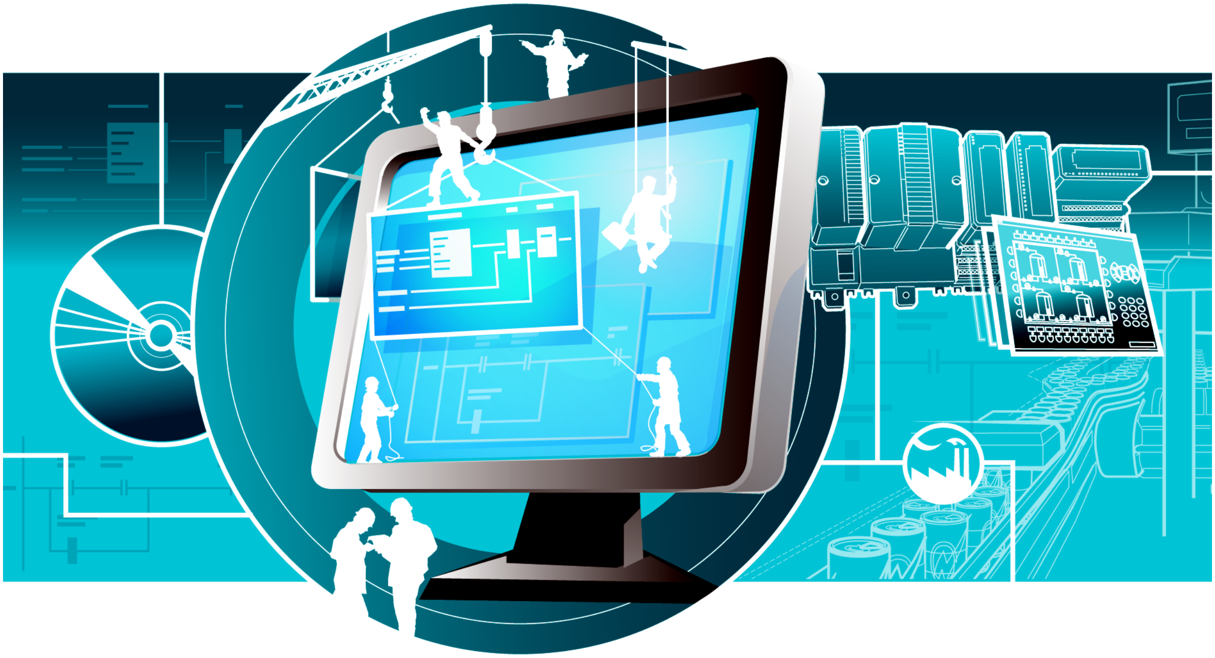
Industrial^{IT}

Compact Control Builder AC 800M

Version 5.0

Basic Control Software

Introduction and Configuration



Industrial^{IT}

Compact Control Builder AC 800M

Version 5.0

Basic Control Software

Introduction and Configuration

NOTICE

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

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

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

The software or hardware described in this document is furnished under a license and may be used, copied, or disclosed only in accordance with the terms of such license.

This product meets the requirements specified in EMC Directive 89/336/EEC and in Low Voltage Directive 72/23/EEC.

Release: June 2006
Document number: 3BSE040935R201 Rev A

TRADEMARKS

TABLE OF CONTENTS

About This Book

Section 1 - Basic Functions and Components

Section 2 - Alarm and Event Handling

Section 3 - Communication

Section 4 - Online Functions

Section 5 - Maintenance and Trouble-Shooting

Appendix A - Array, Queue and Conversion Examples

Appendix B - System Alarms and Events

INDEX

About This Book

General

This manual describes how to use the basic programming and configuration functions that can be accessed via the Project Explorer interface.

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

- Section 1, Basic Functions and Components, describes all the basic functions that are available via system functions, the Basic library, and via 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 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.

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.

Warning, Caution, Information, and Tip Icons

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

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

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

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

Information icon alerts the reader to pertinent facts and conditions.

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

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

Terminology

The following is a list of terms associated with Compact Control Builder. You should be familiar with these terms before reading this manual. The list contains terms and abbreviations that are unique to ABB or have a usage or definition that is different from standard industry usage.

Term/Acronym	Description
Application	Applications contain program code to be compiled and downloaded for execution in a controller.
Control Builder	A programming tool with a compiler for control software. Control Builder is accessed through the Project Explorer interface.
Control Module (Type)	A program unit that supports object-oriented data flow programming. Control modules offer free-layout graphical programming, code sorting and static parameter connections. Control module instances are created from control module types.
Firmware	The system software in the PLC.
Hardware Description	The tree structure in the Project Explorer, that defines the hardware’s physical layout.
Industrial^{IT}	ABB’s vision for enterprise automation.
Industrial^{IT} 800xA System	A computer system that implements the Industrial^{IT} vision.
Interaction Window	A graphical interface used by the programmer to interact with an object. Available for many library types.

Term/Acronym	Description
MMS	Manufacturing Message Specification, a standard for messages used in industrial communication.
OPC/DA	An application programming interface defined by the standardization group OPC Foundation. The standard defines how to access large amounts of real-time data between applications. The OPC standard interface is used between automation/control applications, field systems/devices and business/office application.
Process Object	A process concept/equipment such as valve, motor, conveyor or tank.
Project Explorer	The Control Builder interface. Used to create, navigate and configure libraries, applications and hardware.
Type	A type solution that is defined in a library or locally, in an application. A type is used to create instances, which inherit the properties of the type.

Section 1 Basic Functions and Components

Introduction

Control Builder is a programming tool that contains a compiler, programming editors, standard libraries for developing controller applications and 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 and task supervision. You will find that you can accomplish most of your application development using the basic functions and components presented in this section.

This section is organized in this manner:

- Control Project Templates on page 18 describes the different templates that you can use to create a project.
- Program Organization Units, POU on page 19 introduces the Program Organization Unit (POU) concept.
- System Firmware Functions on page 19 describes firmware functions included in the system, which can be used in any application.
- Hardware on page 21 describes the standard libraries for hardware types
- Basic Library for Applications on page 26 describes the objects of the Basic library, which can be included in any project.
- Application Types and Objects on page 27 introduces the very important, object-oriented, types and objects concept. This subsection also describes how to add your own types and how to create objects (instances) from types.
- Variables and Parameters on page 48 describes how to use parameters and variables to store and transfer values in your control system.
- Library Management on page 75 describes how to work with libraries.

- Hide and Protect Control Module Types, Function Block Types and Data Types on page 87 describes how to hide and protect objects and types, using the Hidden and Protected attributes.
- Task Control on page 89, describes how to set up tasks, to control the execution of your applications.
- Overrun and Latency on page 100 describes how to configure latency control for your tasks.
- Search and Navigation on page 107, 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.
- Compact Flash on page 120, describes how to use Compact Flash (CF) as a mobile memory interface.
- Project Documentation on page 129, describes how to use the Project Documentation function to document standard libraries, your own libraries, and your applications in MS Word format.

Control Project Templates

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

The Compact Control Builder provides the following project templates:

- AC800M
(normal use, for running applications),
- EmptyProject
(rare use, a minimum configuration with only the System folder inserted),
- SoftController
(development use, for simulating applications without a controller).

Program Organization Units, POU

The IEC 61131-3 standard describes programs, function blocks, (control modules)[1] and functions as Program Organization Units (POUs). These units help you organize your control project into code blocks, to minimize code writing, and to optimize code structure and code maintenance.

A POU can best be described as an object type that contains an editor, where you can write code and declare parameters and variables. All POUs can be repeatedly used in a hierarchical structure, except for programs that can only be a 'top-level' POU, inside an application.

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). However, it is important to remember that 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.).

Instead, 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.

Table 1 briefly presents System firmware data types and functions. More information and descriptions can be found in the Control Builder online help.

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

[1] This manual treats control modules as POUs. However, the IEC 61131-3 standard does not discuss control modules.

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.
Array Functions	Handles arrays.
Queue Functions	Handles queues.

Hardware

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

Hardware types are organized and installed as libraries. Thus, it is possible to handle hardware types in a more independent way, which has some advantages:

- The libraries are version handled, thus different versions of the same hardware type may exist, in different versions of the library. This makes it easy to upgrade to newer system versions and to let new and old hardware units coexist.
- New versions of a library, can in a easy way be delivered and inserted to the system.
- Only used hardware types allocate memory in 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.

Standard System Libraries with Hardware

The following standard libraries with hardware are delivered by the system:

Library	Description
BasicHWLib	Basic controller hardware types
CI851PROFIBUSHwLib	Communication interface PROFIBUS DP
CI854PROFIBUSHwLib	Communication interface PROFIBUS DP-V1
ABBDrvNpbaCI851HwLib ABBDrvNpbaCI854HwLib	ABB Drive NPBA and subunits for PROFIBUS
ABBDrvRpbaCI851HwLib ABBDrvRpbaCI854HwLib	ABB Drive RPBA and subunits for PROFIBUS
ABBProcPnlCI851HwLib ABBProcPnlCI854HwLib	ABB Process Panel for PROFIBUS
ABBPnl800CI851HwLib ABBPnl800CI854HwLib	ABB Panel 800 for PROFIBUS

Library	Description
CI855Mb300HwLib	Communication interface MasterBus 300
CI857InsumHwLib	Communication interface INSUM
CI858DriveBusHwLib	Communication interface DriveBus
CI856S100HwLib	Communication interface S100 I/O system and S100 I/O units
S200IoCI851HwLib S200IoCI854HwLib	S200 adapter and S200 I/O units for PROFIBUS
S800ModulebusHwLib	S800 I/O units for ModuleBus
S800CI830CI851HwLib S800CI830CI854HwLib S800CI840CI854HwLib S800CI801CI854HwLib	S800 adapters and S800 I/O units for PPOFIBUS
CI865SattCNHWLib	Communication interface for remote I/O connected via ControlNet
SerialHwLib	Serial communication protocols
CI853SerialComHwLib	RS-232C serial communication interface
CI852FFh1HwLib	Communication interface FOUNDATION Fieldbus H1
PrinterHwLib	Printer unit
ModemHwLib	Modem unit
SerialHwLib COMLIHWLib ModBusHWLib S3964HWLib	Communication protocols

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

Customized Hardware Types

When it is not sufficient, with the standard system libraries, you can create user-defined libraries (see Create Libraries on page 78), to which you can add your own hardware types (customized hardware types) with the Device Import Wizard. The wizard is used to import a device capability description file (for example a *.gsd file), that is, convert the file to a hardware type and insert the type into the user-defined library. (See also Device Import Wizard on page 82 and Supported Device Capability Description Files on page 82.)

In exceptional cases, it may be relevant to insert individual external customized hardware types to a user-defined library. A typical case is if there is a need to use a specific hardware type, that have been converted and used in an earlier version of Control Builder.

The Source Code Report can be used to view which hardware types that are loaded in the project. See Source Code Report on page 125.

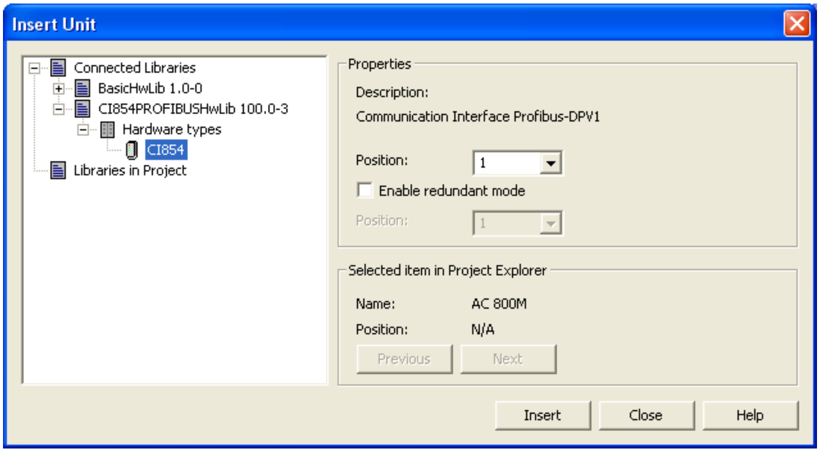
Configuring Controller

To make it possible to configure the controller you first have to insert the libraries into the control project, containing the hardware types (units) to be used in the controller configuration. Furthermore you also have to connect the libraries with the hardware types to the controller. How to add and connect libraries, see Connect Libraries on page 76. General library handling are described in Library Management on page 75.

Add Unit to Hardware in Controller Configuration

To add a new hardware unit into the controller configuration in Project Explorer:

1. Make sure that the library, containing the hardware type to be added, is inserted to the project and connected to the controller.
2. Right-click the unit you want to add a new hardware unit to and select **Insert Unit**. It is not possible to select Insert Unit (dimmed) if the unit cannot contain any subunits or if no more positions are available



Description:
Communication Interface Profibus-DPV1

Position: 1

☐ Enable redundant mode

Position: 1

Name: AC 800M

Position: N/A

Figure 1. Dialog for inserting hardware in a controller configuration.

3. Expand the relevant library folder under Connected Libraries and select the hardware type you wish to add.
   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, you will be offered to connect the library to the controller.

4. Select a position for the hardware unit in the Position drop-down list. The first available position is chosen by default. The Position drop-down list is empty and the **Insert** button is dimmed, if there are no more available positions left.

5. For units supporting redundancy it is possible to check Enable redundant mode check box and select a position for the backup unit.

Some redundant units have a fix position offset. For these units the backup position are automatically calculated by the dialog, and cannot be changed by the user.

6. Click **Insert** button to apply the current changes.
   With the **Previous** or **Next** button it is possible to navigate to another unit in Project Explorer hardware tree.

7. Click **Close** to close the dialog.

Replace Hardware in a Controller Configuration

It is possible to replace a hardware unit in a controller configuration with a different hardware unit. To replace a hardware unit:

1. Make sure that the library, containing the hardware type you want to replace the unit with, is inserted to the project and connected to the controller.
2. Right-click on the unit you want to replace and select **Replace Unit**.

Apart from that it is not possible to change position, this dialog works in the same way as Insert Unit (see above). The system does, as far as possible, retain settings, connections and units in existing sub trees, for example, changing between two similar CPUs can be done without losses.

Basic Hardware

The Basic Hardware library (BasicHwLib) contains standard system hardware types to be used when configuring the AC 800M controller and SoftController. Standard system hardware types are installed together with Control Builder.

Only one version of BasicHwLib can be connected to a controller.

BasicHwLib contains basic controller hardware for AC 800M. The Basic Hardware library contain the following hardware:

- Controllers (AC 800 M and SoftController)
- Compact Flash units
- CPU units
- Ethernet links, serial Com ports and PPP ports
- ModuleBus.

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, see the manual Extended Control Software, Binary and Analog Handling (3BSE035981Rxxxx).

Table 2. 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 Type	Connection module for group start sequences (GroupStartObjectConn).

Application Types and Objects

Types and objects form the basis of your application structure, using the solutions presented in this subsection. To help you get familiar with types and objects, this subsection offers an overview of the following areas:

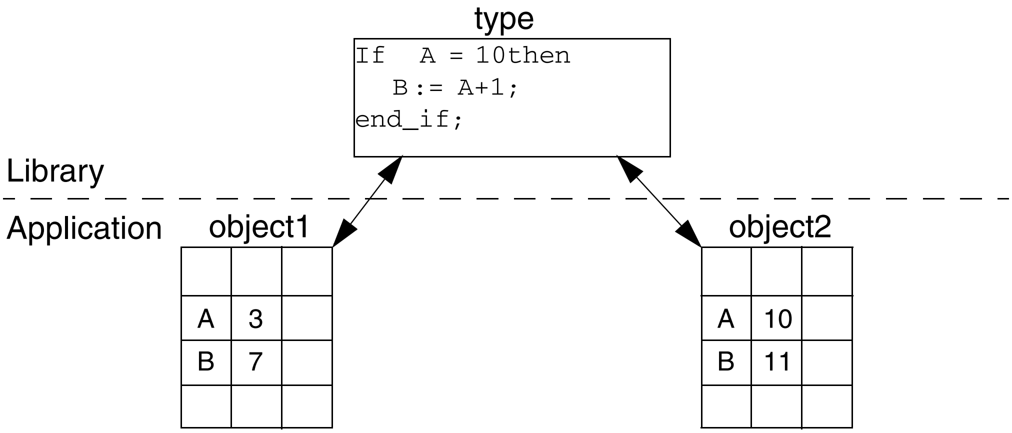
- The type concept, see Types and Objects Concept on page 28.
- The editors that are used to create and configure your types, see Declare a Type in the Editors on page 29.
- Important differences between control module and function block types, see Control Modules or Function Blocks? on page 35.
- How to create types directly in an application, and how to create types in your library for re-use in applications. See Types in Applications on page 36 and Types in Your Own Library on page 37.
- How to create complex types so that they are flexible enough for future upgrades, see Modify Complex Types on page 38.
- What to consider and what to set up before creating types and using them, see Decisions When Creating Types on page 39.
- How to create objects from types and connect the object to the surrounding application or type, see Create and Connect Objects on page 41.
- How different objects are executed, see Function Block Execution on page 45 and Control Module Execution on page 46.
- How to use single control modules as containers for control modules, see Single Control Modules on page 46.

Types and Objects Concept

To represent motors, valves, tanks, etc. that are located in a plant area, and then turn them into manageable units in a control project, requires that you identify and create types (motor types, valve types, mixer types, etc.) and then create a number of objects, based on each of these types.

To help you understand the types and objects concept, this subsection offers a brief presentation of how objects are created from a type and the inherit mechanism that follows. A type is the source (the blue print) for a unit (motor, valve, tank, etc), while an object (instance) represents the unit(s) in libraries and applications. There is an inherit mechanism between a type and all its objects, whereas all objects will have the same performance as the type, and changes performed in the type will affect all objects simultaneously (single source).

A type is a generic solution, which can be used by many objects. Such a solution may contain programming code with variables, functions, connection parameters (textual and graphical), graphical objects and formal instances[1]. Figure 2 tries to give a simplified picture of the relationship between a type located in a library, and two objects created in an application. The type contains the code, whereas each object contains a list of computed variable values. Hence, an object cannot contain any code. Instead, it uses the code inside the type for manipulating its own local variable values.



```
If  A = 10then
  B := A+1;
end_if;
```

A	3	
B	7	

A	10	
B	11	

Figure 2. A simplified example of the relationship between a type an objects.

[1] Formal instances are objects (instances of another type) that are located inside a type. Formal instances are executed when objects based on the type execute in applications.

A type is always static in the sense that it cannot run by itself in applications. In order to execute the code inside the type, you must create an object based on the type (an instance). The object will then execute the code located inside the type. To create an object, point to a type in either a library, or in an application (an object needs a type to exist).

All objects based on the same type have the same characteristics, which means they have equal access to everything in the type. An object does not contain a programming editor or code blocks (only a connection editor), hence you cannot write code inside an object. All logic must be created in the type.

Allocated memory for creating for example, a motor type solution (one motor type and 20 motor objects) is distributed mainly on the programming code inside the type. Therefore, the cost (allocated memory) for each new object (motor) is very small, compared to the type itself. The object only needs to allocate memory for variables, since all code is located and executed from the type. However, the number of objects are still relevant for the total CPU load.

Working with types and objects makes it easier to upgrade, since the inherit mechanism takes care of changes that (often) concern hundreds of objects. A code change, declaring additional connection parameters, etc, can be done once for the type, and is then inherited by all objects simultaneously.

Control Builder also contains a number of structured data types. For more information refer to the Application Programming manual. Hence, a type described in this sub-section is either a function block type or a control module type.

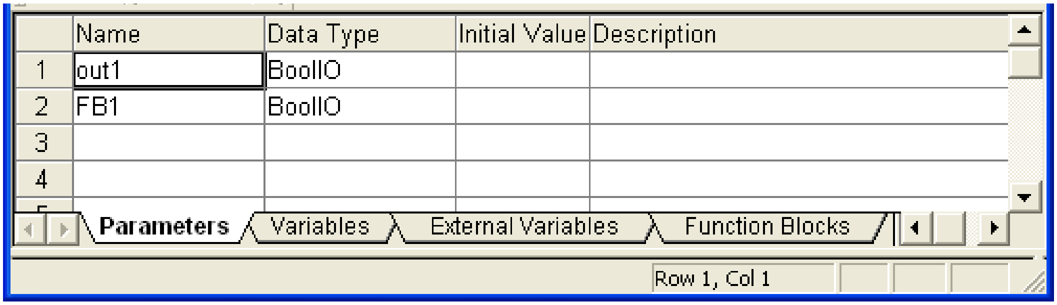
Declare a Type in the Editors

This subsection describes the most common editors used inside a type. All of these editors help you declare the necessary parameters, variables, functions, code, etc. that are needed to make the type work. The editors presented here are opened from a control module type. An Editor contains several declaration panes which can be selected from a tab menu (see Figure 3):

- Declaration pane for declaring parameters (see below).
- Declaration pane for declaring local variables (see Figure 4).
- Declaration pane for declaring function blocks (see Figure 5).
- Programming editor for programming code (see Figure 6).
- Graphical editor, CMD Editor (see Figure 8).

Declaration Pane for Parameters

Figure 3 shows part of an editor for My_MotorType, where the declaration pane for parameters is selected. These parameters can then be used for connecting variables outside the object. To open the declaration pane for parameters, double-click the type.



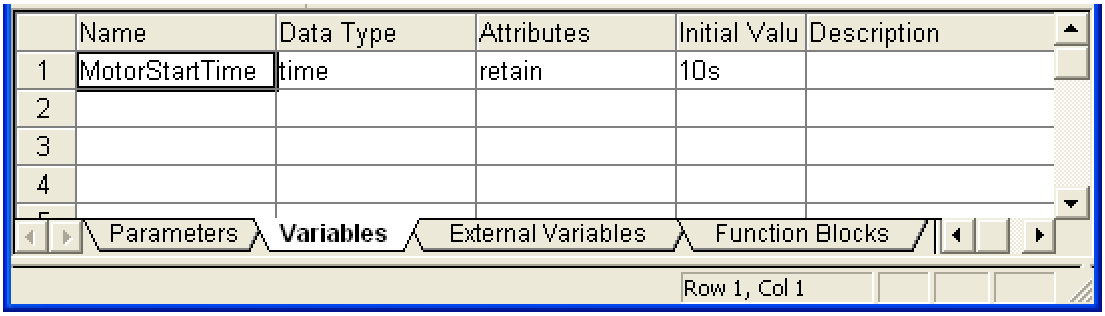
	Name	Data Type	Initial Value	Description
1	out1	BoolIO		
2	FB1	BoolIO		
3				
4				

Figure 3. The declaration pane for creating connection parameters.

Declaration Pane for Local Variables

Figure 4 illustrates a declaration pane for creating local variables inside the type. The local variables can be used by the code inside the type.

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



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

Figure 4. The declaration pane for creating local variables.

Declaration Pane for External Variables

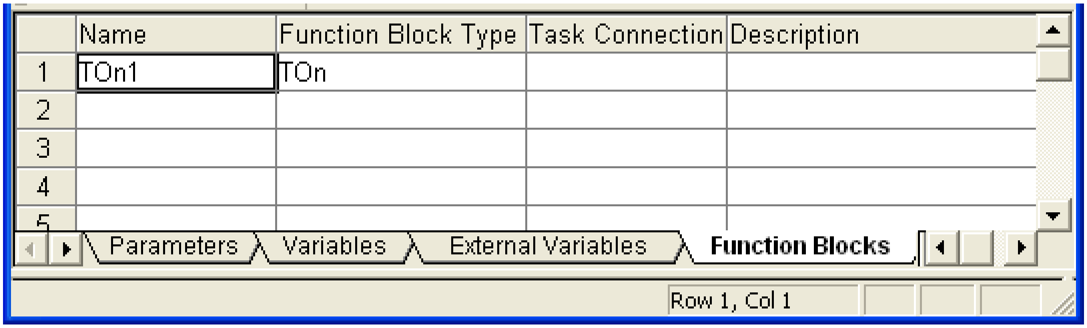
External variables work as pointers to global variables, which means that an object can declare an external variable locally and, via this variable, access the value in a global variable located in the application. External variables and global variables are discussed in External Variables on page 57.

Declaration Pane for Function Blocks

Figure 5 illustrates a declaration pane for declaring function blocks inside the type.

To open the declaration pane for function blocks, double-click the type to open the editor and then select the Function Blocks tab. If the editor is open already, simply select the Function Blocks tab. Name the function block and move the cursor to the Function Block Type column. Press CTRL+J to open a context menu with all function block types available.

You must first connect all the libraries that contain the function block types into your application. Only then will you be able to see available function block types in the context menu (CTRL+J).



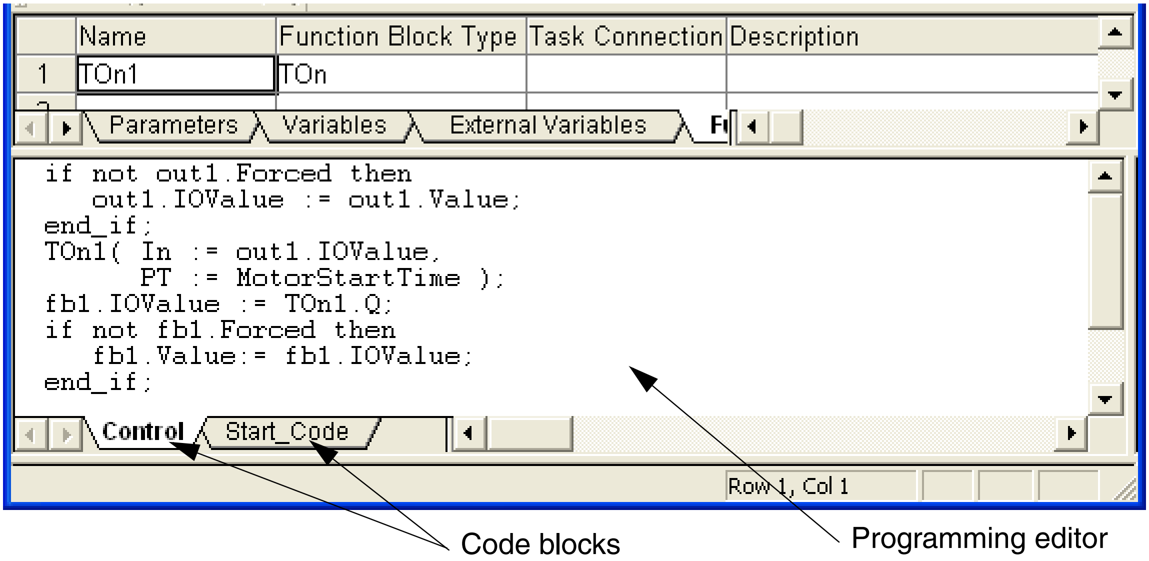
	Name	Function Block Type	Task Connection	Description
1	TOn1	TOn		
2				
3				
4				
5				

Figure 5. Declaration pane for creating function blocks inside a type.

Programming Editor

Figure 6 illustrates part of a programming editor where you can write code in one of the five programming languages, according to the IEC 61131-3 standard. (In our examples, we use Structured Text, ST). The programming editor is always active, and can be reached irrespective of which tab is selected (parameters, variables, function blocks, etc.).

The editor can be expanded with code blocks for structuring the code, like adding new pages to a folder. The code blocks are then executed in a predetermined order, as decided by the compiler (control modules) or from left to right (function blocks).



	Name	Function Block Type	Task Connection	Description
1	TOn1	TOn		

```
if not out1.Forced then
   out1.IOValue := out1.Value;
end_if;
TOn1( In := out1.IOValue,
      PT := MotorStartTime );
fb1.IOValue := TOn1.Q;
if not fb1.Forced then
   fb1.Value:= fb1.IOValue;
end_if;
```

Figure 6. A programming editor with two code blocks.

A brief description of Start_code block and code blocks in general:

- Code block Start_

  A code block with the prefix Start_ is always executed first in an application and *only once*, at application startup (after a warm and cold start, but not after a power failure). It is therefore unsuitable to place functions, function blocks, etc, in a Start_block. This block should be used for initiating alarm strings, converting project constants to strings etc.

However, there are some limitations to the Start_ code block:

- It is only valid for code blocks in control modules.
- It is not valid for code blocks in SFC (Sequential Function Chart).
- The function FirstScanAfterApplicationStart must not be used.
- Function blocks for communication must not be used.

If the application contains a very large amount of code that has to be run in first scan (e.g. alarms in the Start_ code block), the execution time can be so high that overrun occurs. This will lead to that the controller will eventually shut-down.

- Code blocks are very useful for structuring code. By dividing your programming code into a number of code blocks, you can improve 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 which characters that cannot be used in code block names.

- There is no actual limit to how many code blocks that can be created in a type. However, you should only create the absolutely necessary amount of code blocks, since each code block will affect memory consumption and the execution time of the type.

Open Code Block Menu

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

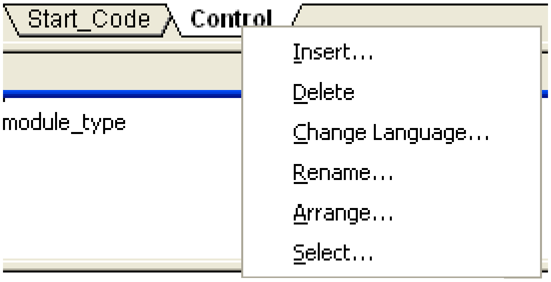


Figure 7. The code blocks Start_Code and Control. Right-click a code block tab to access the context menu.

Graphical Editor

Figure 8 shows part of the graphical editor (CMD Editor). The Control Module Diagram editor (CMD Editor) is a combined function for drawing and programming. (The term diagram refers to the graphical view of control modules and connections.) The editor allows you to create and edit control modules, code, graphics, and to connect variables and parameters.

To open the CMD Editor, right-click the control module type and select CMD Editor.

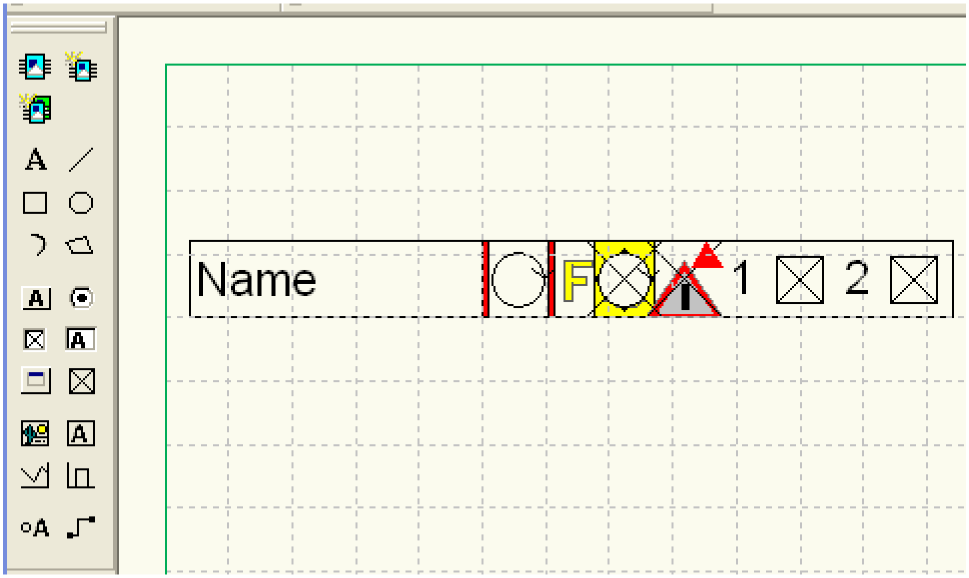


Figure 8. Graphical objects created in the CMD Editor.

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

Control Modules or Function Blocks?

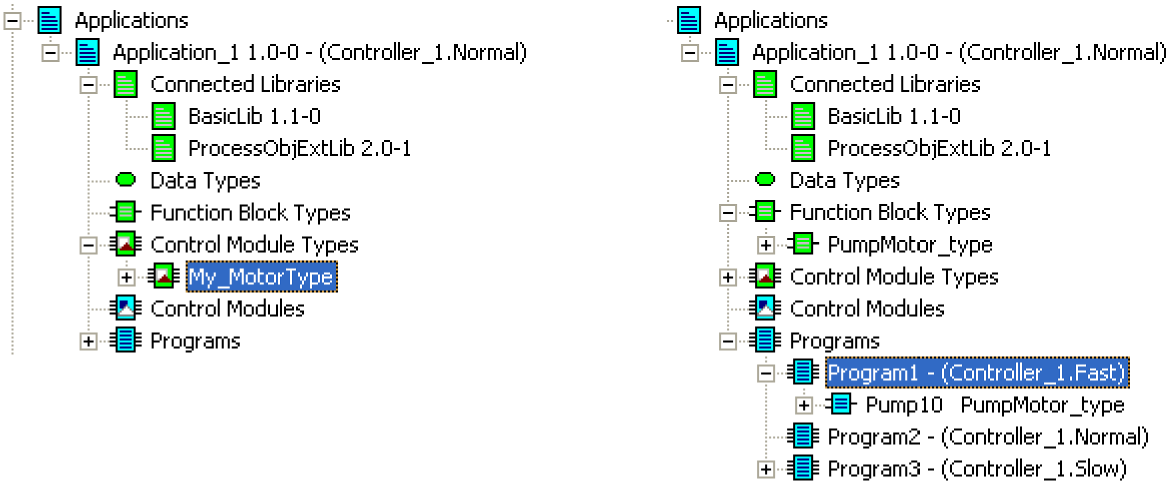
A type can be a control module type or a function block type. Besides that, you can always mix types and objects. For example, a control module can be created inside a function block type (to add graphics), or you can create a function block inside a control module type (to execute a list of basic functions). The following list discusses some differences between control module types and function block types.

- Control modules types may have graphical connections (see Graphical Connections on page 42).
- Control modules types use code sorting (see Control Module Execution on page 46).
- Control modules are always executed by the system and always once per scan, whereas function blocks are always executed from code. Therefore, a function block can be executed, none, one time, or several times per scan. This is the main difference between control module execution and function block execution.
- Parameter values on function blocks are copied (except In_Out parameters, see Function Block Execution on page 45).
- Function block types are required when using extensible parameters (see Extensible Parameters in Function Blocks on page 73).

Other than this, the decision whether to use control module types or function block types depends on the context and environment. More guidelines about the use of control modules and function blocks are given in the Application Programming manual.

Types in Applications

Creating a type in an application is the quickest and easiest way to get started. No libraries have to be created before creating types, and the available methods are still there: you can connect libraries, create your own data types, and select which object type to use (see Decisions When Creating Types on page 39). However, if you choose to create a type directly in an application, it can only be used inside that application.



```
Applications
  Application_1 1.0-0 - (Controller_1.Normal)
    Connected Libraries
      BasicLib 1.1-0
      ProcessObjExtLib 2.0-1
    Data Types
    Function Block Types
    Control Module Types
      My_MotorType
    Control Modules
    Programs
```

```
Applications
  Application_1 1.0-0 - (Controller_1.Normal)
    Connected Libraries
      BasicLib 1.1-0
      ProcessObjExtLib 2.0-1
    Data Types
    Function Block Types
      PumpMotor_type
    Control Module Types
    Control Modules
    Programs
      Program1 - (Controller_1.Fast)
        Pump10   PumpMotor_type
      Program2 - (Controller_1.Normal)
      Program3 - (Controller_1.Slow)
```

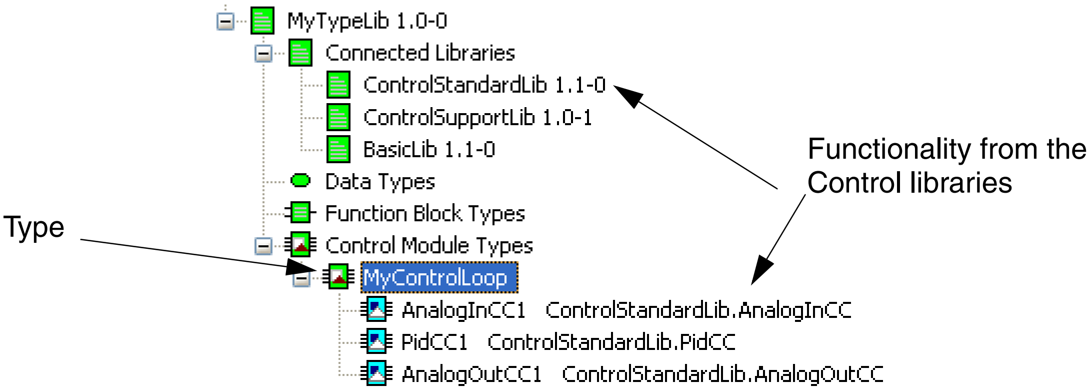
Figure 9. Two examples of a type created especially for Application_1. (Left) A control module type (My_MotorType), (Right) a function blocks type (PumpMotor_type).

To gain access to standard libraries (or own libraries), the first step is to insert them into the control project, see Library Management on page 75, and then connect them to the application. Hence, types in the application may in turn use objects from existing types in those connected libraries.

Types in Your Own Library

A major reason for creating types inside a library, instead of creating them directly in an application, is the possibility to re-use them in other applications (this will not be possible if you create your types directly in an application). If you create types in a library, all necessary functions and programming work can be stored in this library. The library can then be connected to any application.

Provided that a new library has been created, self-defined types can be created in that library (Compact Control Builder does not allow you to create types in a standard library).



```
MyTypeLib 1.0-0
  Connected Libraries
    ControlStandardLib 1.1-0
    ControlSupportLib 1.0-1
    BasicLib 1.1-0
  Data Types
  Function Block Types
  Control Module Types
    MyControlLoop
      AnalogInCC1   ControlStandardLib.AnalogInCC
      PidCC1   ControlStandardLib.PidCC
      AnalogOutCC1   ControlStandardLib.AnalogOutCC
```

Figure 10. A Type (MyControlLoop) created in MyTypeLib library. This example shows a control loop created as a control module type, while the components are ready-made objects from the standard libraries.

Modify Complex Types

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

Refuse Incinerator Type Example

Assume that a plant area has two identical refuse incinerators.

A type solution like this would be complex, but manageable, if we build a Refuse Incinerator type in a library with several underlying types. We could then re-use the type twice (as two objects), in two separate applications, by simply connecting our library to each application.

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, it would be tempting (and normal) to connect the library to both Application_1 and Application_2. It would then be easy to create an Incinerator1 object in Application_1 and an Incinerator2 object in Application_2.

This (type and object) strategy works well, but with one important exception! If the Incinerator2 object running in Application_2 suddenly needs an individual change, perhaps, instead of 10 conveyors, it must be enlarged to 20 conveyors. Then we have to re-enter the library and change the Feeder type inside the Refuse Incinerator type. But, changing anything inside the Refuse Incinerator type would affect both incinerators, (due to the type and object inherit mechanism).

By enlarging the Feeder type with 20 conveyors, both Incinerator objects will be changed and suddenly contain 20 conveyors. This is not what we wanted.

The Refuse Incinerator Type Solution

The strategy to avoid this type of problem is to continue with types and object when building complex type solutions, but once the type solution is ready, consider possible individual (object) changes in the future. If you need to be able to change an individual object, then copy your type on the highest type level (in our example, Refuse Incinerator Type1 and Refuse Incinerator Type2).

In our example, we create an Incinerator10 object in Application_1, based on Refuse Incinerator Type1, and then create an Incinerator20 object in Application_2, based on the new type copy, Refuse Incinerator Type2. This will increase memory consumption in the controller, but you gain the possibility to perform individual changes. For example, you can now change the number of conveyors in the feeder for one of the applications, without affecting the other.

Decisions When Creating Types

This sub-section describes how to set up a type, that is, decisions that have to be made before you start programming code, declaring parameters and variables, etc. Many functions and type solutions have been developed already, and Control Builder helps you set up and access these options before you start programming. Read more about design analysis in the Application Programming manual.

You have to make a number of decisions before you can start programming code, declare parameters, download to controllers, etc. (see Figure 11). One of these decisions is, for example, whether you need to create objects in your own type(s) that are based on other types located in external libraries. Then you must first connect those libraries to your library or application.

Next decision you must make is whether you need to create self-defined structured data types for passing parameters through several layers of objects. The data types that you create will automatically be connected to your library or application. Structured data types are often useful in more complex type solutions, with a deep hierarchical structure. Your last decision is more of a programming choice. Do you need a function block type, or should you use a control module type?It can be said (though this a big generalization) that if you are going to program code in one of the Program's POUs[1], then select function block types, if you are going to program mainly in a graphical editor, and you want automatic code sorting, then select control module types.

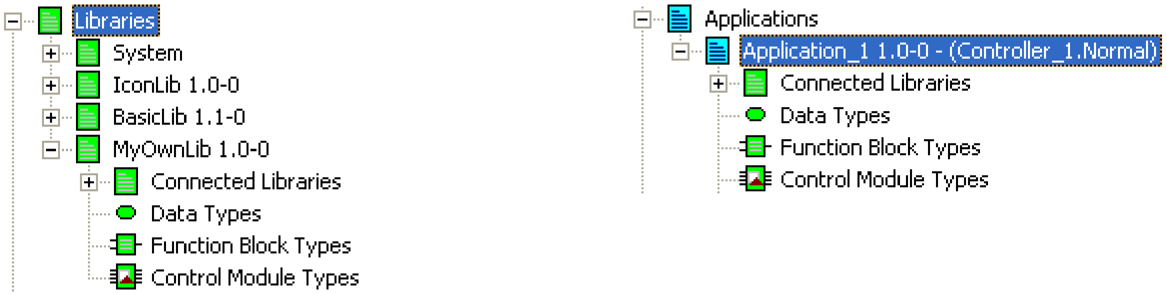


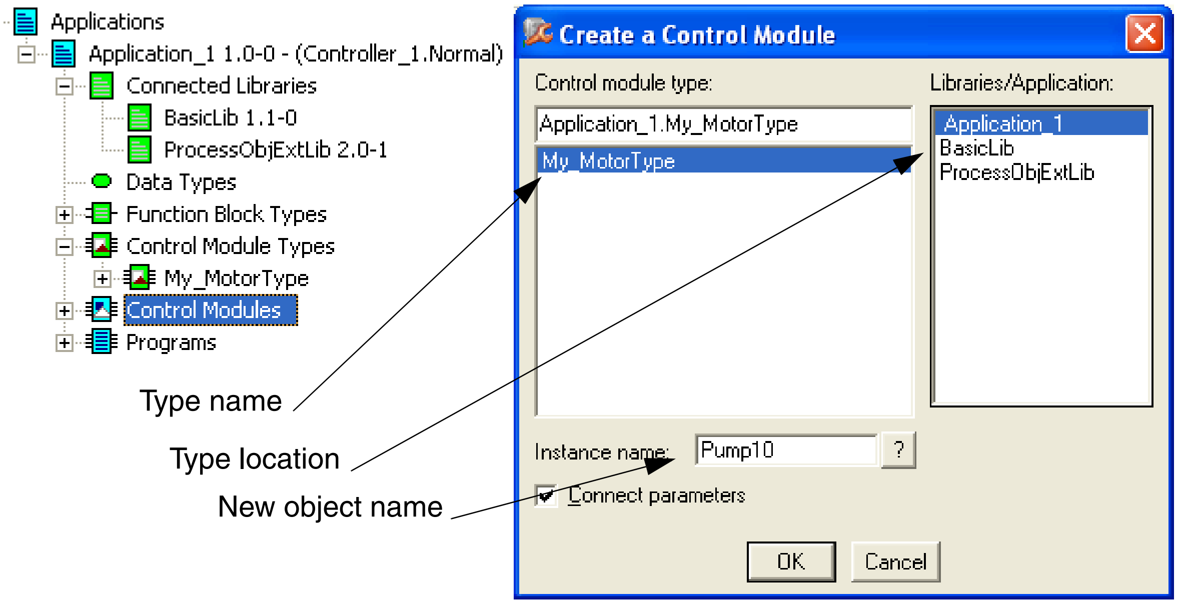
Figure 11. Available settings for setting up an object type, whether it is in libraries, or applications

How to access these methods is described in Control Builder online help. Select one of the folders in Project Explorer and press F1.

[1] See Program Organization Units, POU on page 19.

Create and Connect Objects

An object is a function block or control module based on a type. This means that each time you create a new object Control Builder will prompt you for a type. The type can be located in an inserted library (inserted into the control project) or located in your library, or directly in an application. Either way, a type and its location must always be selected. Once the type has been selected, the next step is to connect the connection parameters.



Applications
- Application_1 1.0-0 - (Controller_1.Normal)
  - Connected Libraries
    - BasicLib 1.1-0
    - ProcessObjExtLib 2.0-1
  - Data Types
  - Function Block Types
  - Control Module Types
    - My_MotorType
  - Control Modules
  - Programs

Create a Control Module

Control module type: Application_1.My_MotorType

My_MotorType

Libraries/Application: Application_1, BasicLib, ProcessObjExtLib

Instance name: Pump10

☑ Connect parameters

Figure 12. Creating an object (Pump10) based on My_MotorType, which located in the application. The object needs the location (Application_1) and type (MyMotorType).

Connections

Control modules can be connected to each other in two different ways: through graphics or through text. Graphical connections are implemented directly in the Control Module Diagram editor and text-based connections in the Connection editor.

Graphical Connections

Graphical nodes and graphical connections allow you to connect control modules in a simple and efficient manner.

It is easy to recognize a control module parameter that can be graphically connected. These parameters have NODE in the beginning of the parameter description. This is 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 13 illustrates three graphical connections for group starting motors. You connect the modules with the Graphical Connection function (located in the CMD Editor).

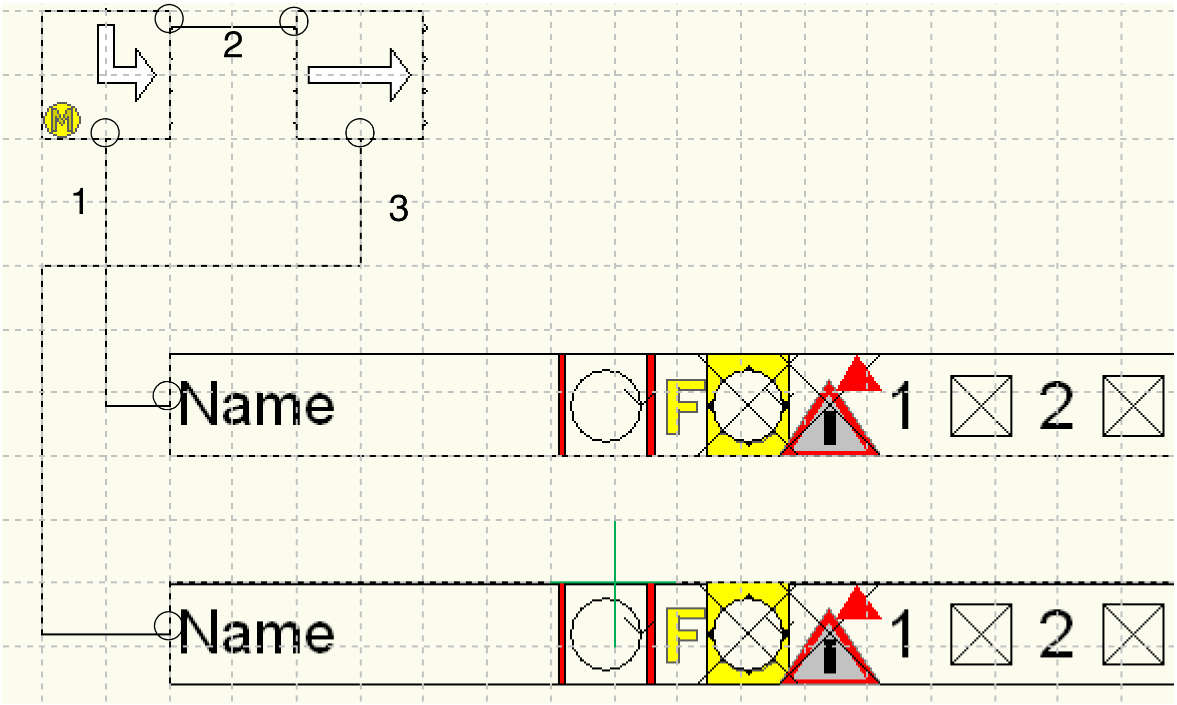


Figure 13. Two motor objects that have been graphically connected with a Start and Next object located in the Group Start library. Note the circles which symbolize the connection nodes.

Textual Connection

You can reach the Connections Editor via the Connections entry, simply right-click the module and select Connections. Parameters can be connected to the actual variables presented in the Connections Editor. It is not possible to connect the same parameter both graphically and textually. 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.

Connect an Object

The Connections editor is a parameter/variable interface between the object and its closest surroundings. The Connections editor displays the parameters that have been declared in the type, seen through the control module object, and you connect surrounding parameters/variables to the object.

If a control module object is created in an application (see Figure 14), then the application can be seen as the closest surroundings, and variables in the application should be connected to the object.

In the same way, when an object is created in a type (located for example in a library), then the type can be seen as the closest surroundings, and parameters/variables in the type should be connected to the object.

If you need to connect parameters to objects located several hierarchical layers away (not the closest surrounding) then structured data types will simplify the connections compared to passing corresponding parameters. For more information on structured data types, see also the Application Programming manual (3BSE044222R101).

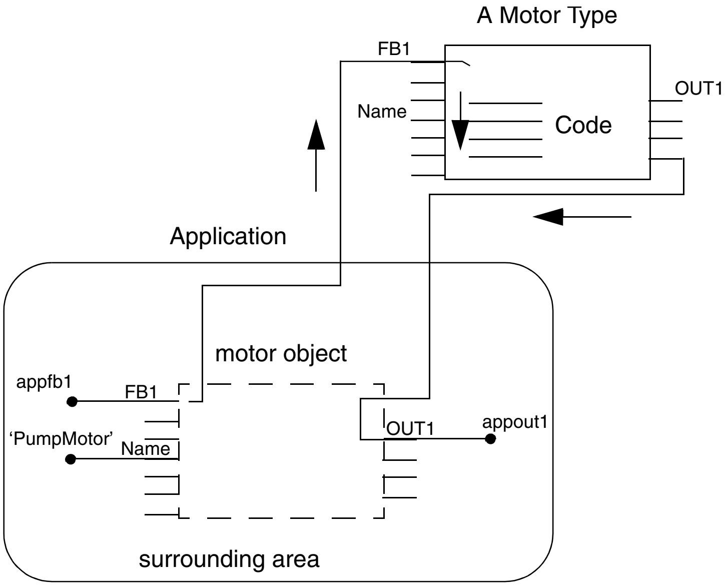


Figure 14. A control module object connected to variables in an application. The application is the 'surrounding area' with the variables appfb1, Name (initial value 'PumpMotor') and appout1 connected to the object.

The connection parameters for the motor object illustrated in Figure 14 connect the parameters (*FB1*, *Name* and *OUT1*) to the variables (*appfb1*, *appout1*, *Name*; *Name* has the initial value PumpMotor) that have been declared in the application. Once the variables have been connected to the object, it is ready to run in the application (see Figure 15).

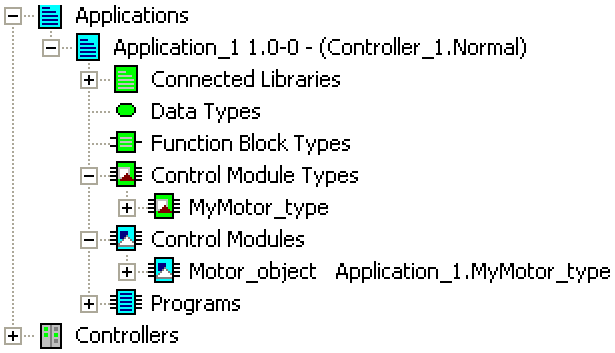

```
⊟ Applications
  ⊟ Application_1 1.0-0 - (Controller_1.Normal)
    ⊞ Connected Libraries
      Data Types
      Function Block Types
    ⊟ Control Module Types
      ⊞ MyMotor_type
    ⊟ Control Modules
      ⊞ Motor_object   Application_1.MyMotor_type
    ⊞ Programs
⊞ Controllers
```

Figure 15. The object 'Motor_object' has been created in the application.

Function Block Execution

Function block parameters have three directions: In, Out and In_out.

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. 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 has been executed, and pass the new values to the surrounding variables outside the function block.

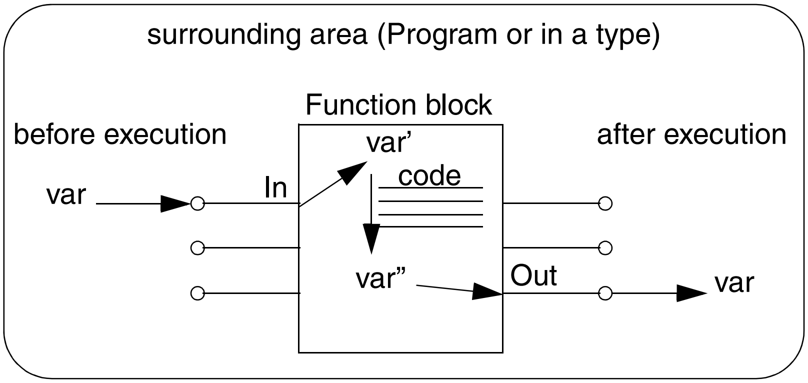


Figure 16. In and Out parameters for a function block. This example illustrates how In and Out parameters copies the variable (var).

In_Out parameters, however, are passed by reference, which means only a reference to the actual variable outside the function block is passed inside the function block. In other words, no local representation of the parameter exists inside the function block. Performing operations on an In_Out parameter inside a function block, thus means performing the operations directly on the actual variable connected to the function block. See also Connecting Variables to I/O Channels on page 68.

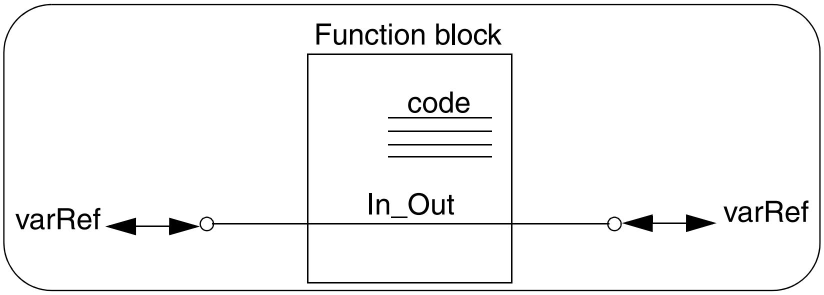


Figure 17. In_Out parameter for a function block. This example illustrates how the In_Out parameter points as reference to the value in the variable varRef.

Control Module Execution

Control modules provide so-called 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 object 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 your code behind the scenes. This is called code sorting.

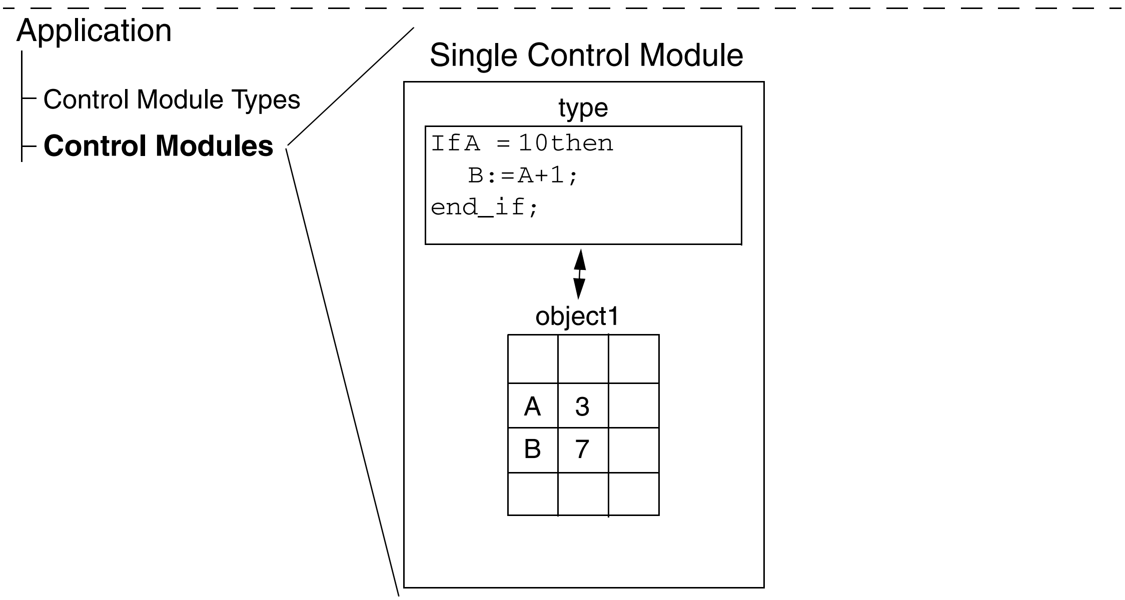
For more information on Code Sorting, see the Application Programming manual.

Single Control Modules

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

Compared to the previous discussions about types and objects, a single control module can be considered as a hybrid of them both (see Figure 18). First of all, you create a single control module as an object that is it must be created under the control module folder (not the control module type folder) in an application.

Once a single control module has been created it start acting as both a type and an object, that is it contain code, editors for declaring parameters, function blocks etc just like a regular type does. But still, it also contains object information like variable values etc like regular objects do. A single control module can never be reusable as a type that can be used to create many objects. However, it can be copied to a new single control module, and then be modified.

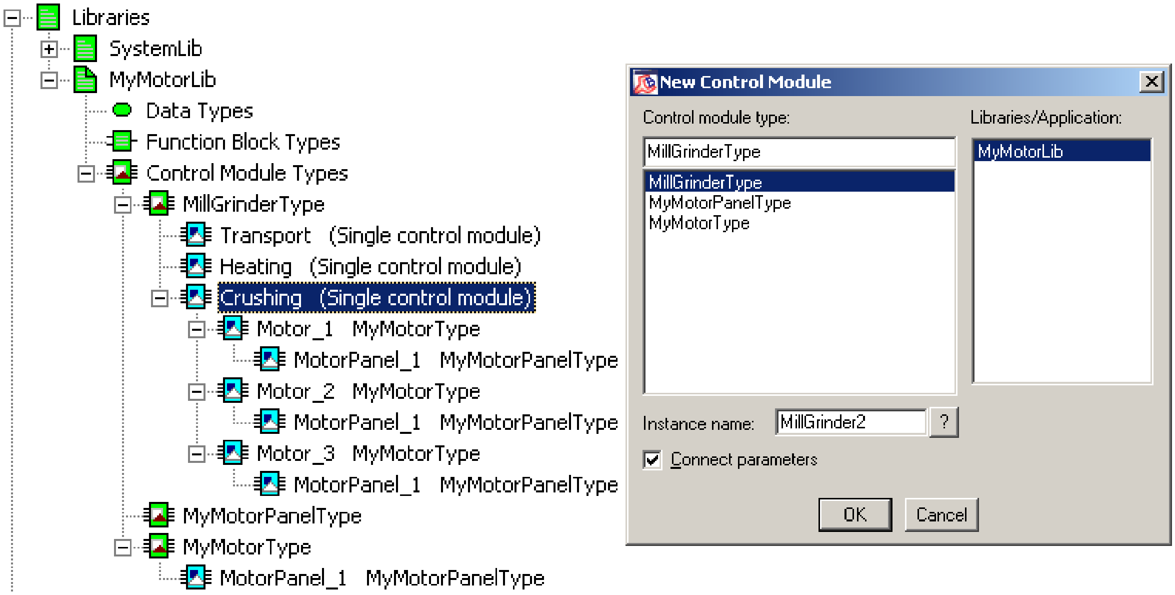


```
IfA =10then
   B:=A+1;
end_if;
```

A	3	
B	7	

Figure 18. A single control module. This module is not reusable, hence intended to be used only once for grouping objects into a single unit.

Single control modules can be used as a framework and attach control module objects inside, like an application does with objects. Figure 19 illustrates this, where three single control modules (Transport, Heating, and Crushing) form the framework for the control modules (Motor_1, etc.).



```
Libraries
  SystemLib
  MyMotorLib
    Data Types
    Function Block Types
    Control Module Types
      MillGrinderType
        Transport   (Single control module)
        Heating   (Single control module)
        Crushing   (Single control module)
          Motor_1   MyMotorType
            MotorPanel_1   MyMotorPanelType
          Motor_2   MyMotorType
            MotorPanel_1   MyMotorPanelType
          Motor_3   MyMotorType
            MotorPanel_1   MyMotorPanelType
      MyMotorPanelType
      MyMotorType
        MotorPanel_1   MyMotorPanelType
```

New Control Module

Control module type: MillGrinderType

MillGrinderType
MyMotorPanelType
MyMotorType

Libraries/Application: MyMotorLib

Instance name: MillGrinder2

☑ Connect parameters

Figure 19. Single control modules form the framework for the control modules.

Variables and Parameters

Variables and parameters are the carriers of data throughout the system. This section contains information designed to help you use parameters and variables in the best way possible:

- Variable and Parameter Concept on page 49 gives an overview of variables and parameters and how they are used.
- Variables on page 51 gives an overview of the different variable types.
- Variable Entry on page 52 describes how to declare variables.
- External Variables on page 57 describes how to define external variables.
- Access Variables on page 58 describes how to define and use access variables.
- Communication between Applications Using Access Variables on page 59 and Communication in an Application Using Global Variables on page 60 describe how communicate between applications.

- Control the Execution of Individual Objects on page 61 describes how to use variables and parameters to control the execution of objects.
- Project Constants on page 64 describes the use of project constants and how to update them.
- I/O Addressing Guidelines on page 68 describes the rules for addressing I/O channels.
- Connecting Variables to I/O Channels on page 68 describes how to connect I/O variables to I/O channels.
- Extensible Parameters in Function Blocks on page 73 describes extensible parameters (these can only be used in function blocks).
- Keywords for Parameter Descriptions on page 74 describes keywords used in description in editors to identify the function of a parameter.

Variable and Parameter Concept

Variables

There are a number of different kinds of variables in Control Builder for storing and computing values. A way of understanding the use of these variables presented throughout this section is perhaps to consider them as carriers on object, application and network levels.

Local variables are mainly used inside objects as carriers of local values. Global variables are declared in the application and holds values that can be reached by any object in the application. Access variables are used as carriers for communication between several applications and controllers in a network.

- Local variables is the most common variable type. They belong to the code and can only be accessed within the same function block, control module or program.
- Global variables on the other hand, are always declared in an application and can be accessed by any function block, control module or program. However, in order 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 57).

- Access variables allow data exchange between controllers, that is, access variables can be accessed by other controllers. You can read more about access variables in section Communication between Applications Using Access Variables on page 59.

In spite of the different variables purposes, they all have one thing in common – a variable holds or carries a value (except an external variable). They are defined by their name and data type, which defines the characteristics of the variable (dint, bool, real, string, etc.).

Parameters

Parameters on the other hand, cannot store any values. Instead, you can assign variables to parameters of function blocks, control modules and functions. Variables store the value of the corresponding (connection) parameters.

Simply put, 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 45 for more details.

Variables

Table 3 lists available variables in Control Builder.

Table 3. 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 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).
Global variable	**Application level.** Can be accessed from anywhere in the code within an application. 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 60.
Access variable	**Network level.** Variable that can be accessed by remote systems for communication between controllers. See also Access Variables on page 58 and Communication between Applications Using Access Variables on page 59.	Access Variable editor of a controller.

Variable Entry

Control Builder helps you 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*. You will receive an error message if such a word is used. For naming guidelines and information on relevant tools, refer to the Application Programming manual.

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

Table 4. 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

Table 4. Simple data types (Continued)

Data type	Description	Bytes allocated by variable	Initial value (default)
string	Character string[1]	10 bytes + string length [n]	“
word	Bit string	4	0
dword	Bit string	4	0
time	Duration	8	T#0s
date_and_time	Date and time of day	8	1979-12-31-00:00:00
real	Real number	4	0.0

[1] 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.

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 you do not define the string length, Control Builder will automatically allocate 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. There are a number of predefined data types in Control Builder (for example BoolIO and RealIO) that are structured data types. You can also create self-defined structured data types, see Decisions When Creating Types on page 39.

In a control module, the word “default” can be used as an initial value for a parameter. 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 5, can be given to a variable.

Table 5. 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 4 on page 52.
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 on disk, and retained after warm or cold restart.(1) Coldretain overrides the retain attributes in a structured data type.

Table 5. Variable attributes (Continued)

Name	Description
constant	You 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.
nosort	This attribute suppresses the code sorting feature for control module types. Do not use the nosort attribute unless you know the data flow characteristics in detail.
state	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 *:old* and *:new* to the variable name.

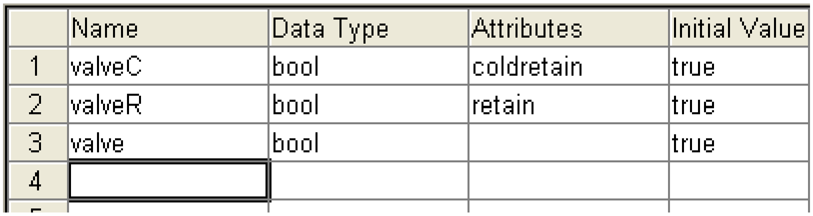
[1] When an application is downloaded the very first time, variables will get their initial data type values, even though they have been declared with the attribute coldretain, and, that the controller has done a cold restart. Hence, no variables can receive their coldretain values before they have been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

You can 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).

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 20 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 20. Three variables with different attributes settings.

According to the attribute settings in Figure 20, 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

You can 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 4 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.

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.

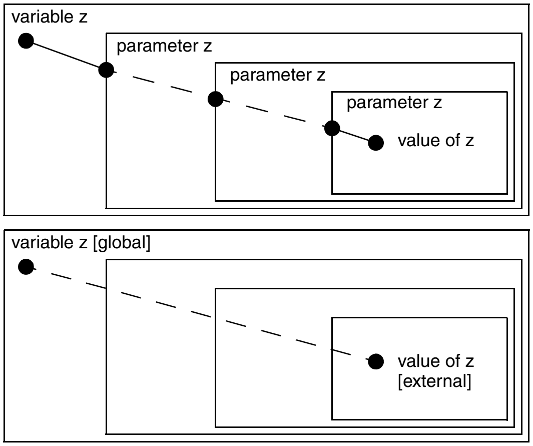


Figure 21. (Top): The variable z can be accessed deep down in the structure, using several parameters. (Bottom): Using external (and global) variables, the variable z is accessed directly, without having to use parameters.

Access Variables

Access variables are needed when the system works as a server. Allowed protocols are MMS, COMLI and SattBus. Variables are declared in the Access Variable Editor under the corresponding 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 your Controller and select **Editor**.

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

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.

If you want to limit the access to an MMS variable, you must set the Attribute to ReadOnly. If the attribute is left blank, both read and write will be 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 COMLI),
- 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.

COMLI

COMLI variables can be accessed in two ways only, either COMLI direct addressing with capital X and the number for boolean, or capital R and the number for registers (R0-R3071) 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 58). This also applies when two applications are downloaded to the same controller (see Figure 22).

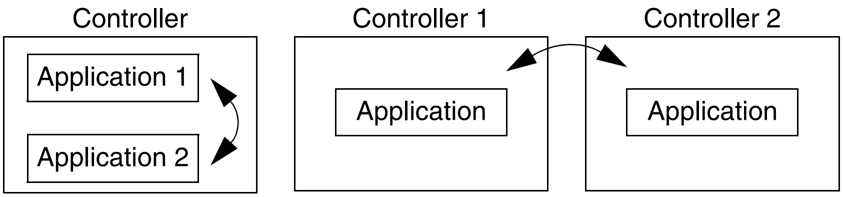


Figure 22. Variables for communication between applications must always be declared as access variables.

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.

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 what you want to achieve, this can be done in two ways:

1. If you need to create a new task and connect this task to all the following objects you should read the sub-section 'Using a Global Variable Connected to an External Variable on page 61.
2. If you need the possibility to choose a new task for each individual object (and for that object only), you should read the sub-section 'Using a Global Variable Connected to a Parameter on page 62.

Using a Global Variable Connected to an External Variable

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

This method is based on declaring a global variable (for example *Speed*) of data type *string*, with the attribute *constant* and the initial value *'SuperFast'*. In order to reach the following 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). 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 you later on need to change the task connection for all the objects (perhaps hundreds of objects), you only need to change the initial value for the global variable in the application (see Figure 23). The present task connection for all objects will point, via the external variable to the task declared by the global variable.

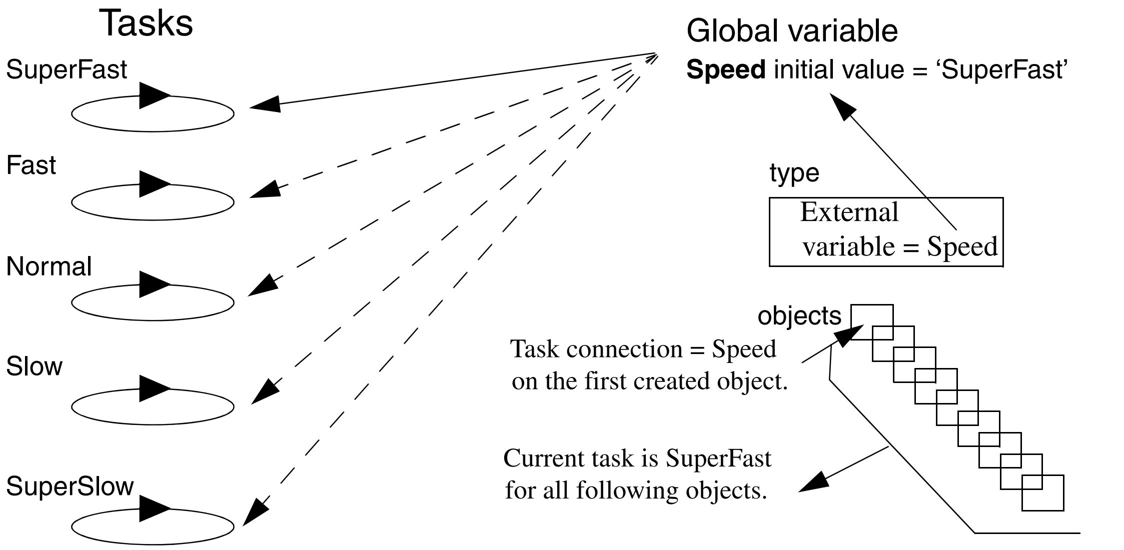

Tasks
SuperFast
Fast
Normal
Slow
SuperSlow
Global variable
Speed initial value = 'SuperFast'
type
External
variable = Speed
objects
Task connection = Speed
on the first created object.
Current task is SuperFast
for all following objects.

Figure 23. 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 you have 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 you can change the task connection on each following formal instance, by simply connecting a parameter to a different global variable. (See Figure 24).

For more information on formal instances, see Types and Objects Concept on page 28.

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.

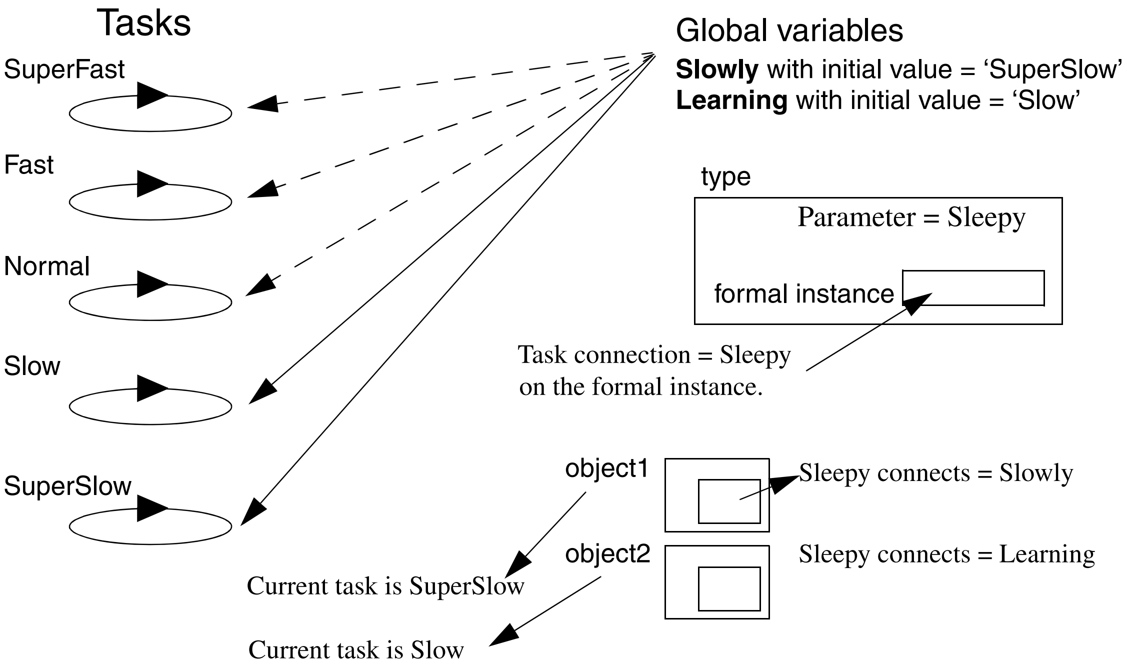


Figure 24. Each object can be connected to a different task via the parameter Sleepy declared in the type and task connected in the formal instance.

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).

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, you can 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.

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 61.

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 you 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. If you have to change such a “constant” value, you do not want to have to change it at every occurrence, but once. 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 your 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 you have created a library that makes extensive use of text strings. Instead of including strings statically in the library, in your own native language, you can use project constants. In this way, you allow 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 (by you) 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 you 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, if you prefer even more levels; “cACMEValveLib.Settings.Labels.Stop”.

Example 2: Combination of Dynamic and Static String Constants

Consider the following function block, in Figure 25, that controls high alarms. *Signal* is of *RealIO* type, *Alarmlevel* is of *real* type, and *Message* is of *string* type.

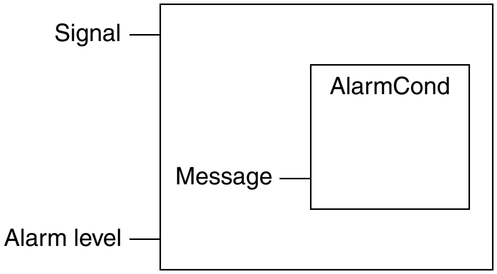


Figure 25. The function block AlarmCond located in the Alarm library.

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

High Level (> 75 ºC)

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. ºC (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, you should 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 you 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, you may use the structure cACMEValveLib.Settings.Labels.HighLevel, as described earlier, if you prefer 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 your 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 your application program.
- Try to divide your 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 your 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

You can only connect one variable 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 45.

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 59.

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 26 has the address Controller_1.0.11.1.1.

Figure 26 illustrates an example of a controller hardware position.

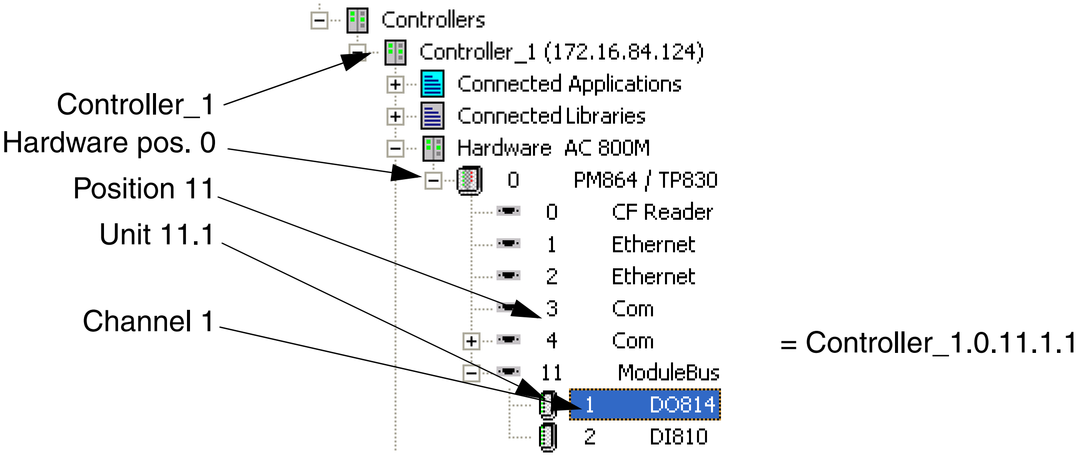


Figure 26. An example of how IO channel addresses are created in a control project.

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

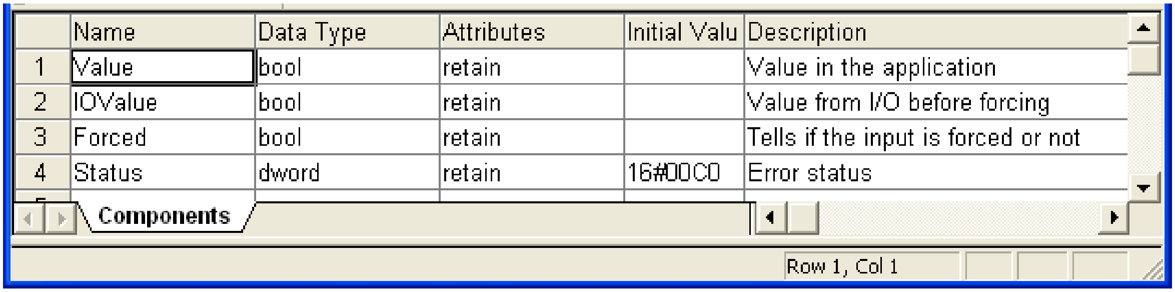
Variables connected to I/O can be of any of the simple data types, *bool*, *dint*, *dword* or *real*, or any of the system-defined I/O data types. For example, an IO unit input can be connected to a variable of *bool* data type or a variable of *BoolIO* data type. For applications that only require a 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.

You can 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.

You 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 you are 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 27 presents as an example the available components inside the structured data type *BoolIO*.



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

Figure 27. Components inside the structured data type BoolIO.

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

By declaring a structured data type, you will get access to more information from the IO channel, which can be read/written in code.

If you choose to declare MyIOVar as a simple data type, *Bool*, you will only be able to access the channel value. In other words, you cannot read or write other values from your 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.

This is a common mistake, due to the fact that the I/O channel only lists *bool*, *real*, etc., in the type column, and does not include the corresponding *BoolIO*, *RealIO*, etc.

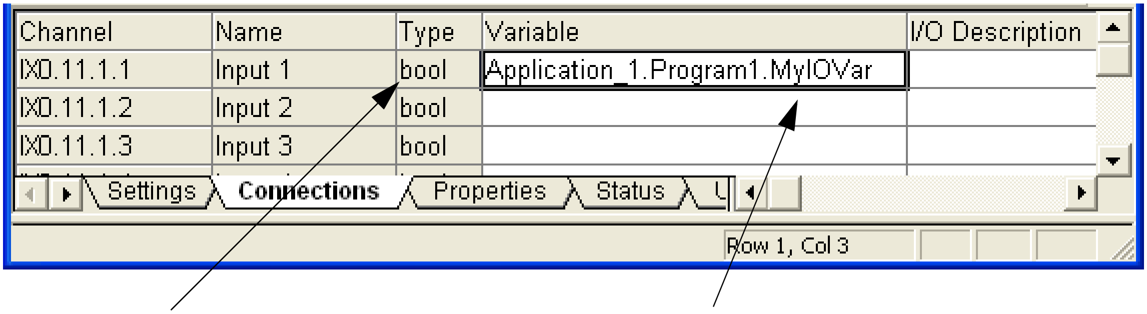
Table 6 shows the (hardware editor) entries to different IO channels. The Type column presents the IO channel data type in the hardware editor, whereas the Variable column presents possible data type connections (simple, structured).

Table 6. Possible variable (data types) connections to IO channels.

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

[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 204.

See Figure 28 and the corresponding structured data types in Table 6.



Channel	Name	Type	Variable	I/O Description
IX0.11.1.1	Input 1	bool	Application_1.Program1.MyIOVar	
IX0.11.1.2	Input 2	bool		
IX0.11.1.3	Input 3	bool		

IO channel of type bool.

MyIOVar of BoolIO (correct connection).

Figure 28. A correct way of connecting IO variables. The structured data type MyIOVar connected to an IO channel.

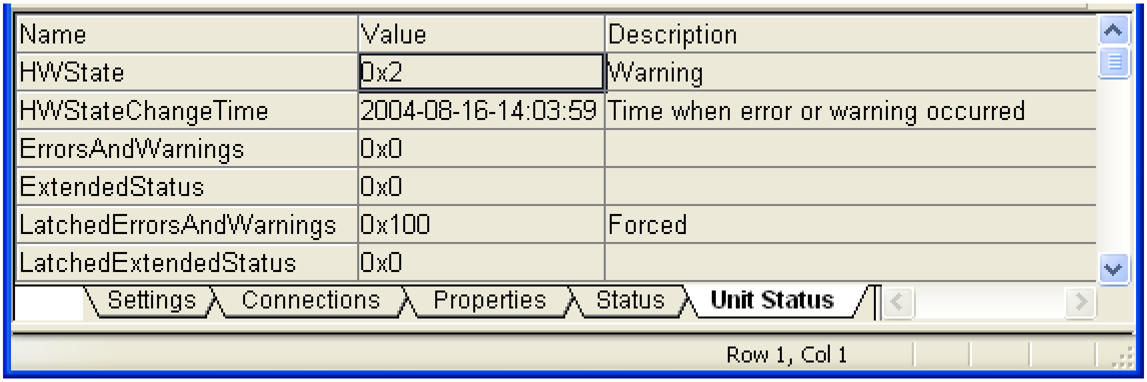
Example of I/O Channel Representation

The IO channel in Figure 28, IX0.11.1.1, interpreted from Table 6, 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. You can connect a variable to Unit Status or selecting the Unit Status tab in the hardware editor.

If you choose 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 will provide you with more extended unit status information. See the contents inside the Unit Status tab in Figure 29.

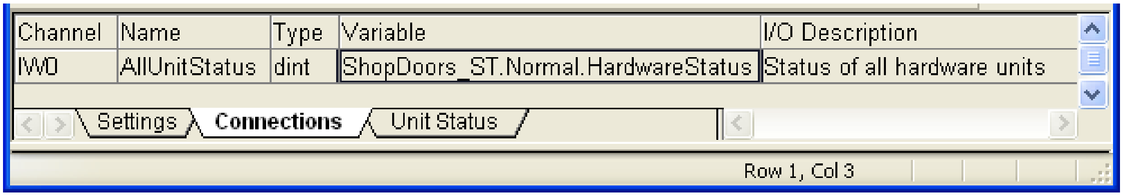


Name	Value	Description
HWState	0x2	Warning
HWStateChangeTime	2004-08-16-14:03:59	Time when error or warning occurred
ErrorsAndWarnings	0x0	
ExtendedStatus	0x0	
LatchedErrorsAndWarnings	0x100	Forced
LatchedExtendedStatus	0x0	

Figure 29. 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, you 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 will provide you with more extended unit status information.



Channel	Name	Type	Variable	I/O Description
IW0	AllUnitStatus	dint	ShopDoors_ST.Normal.HardwareStatus	Status of all hardware units

Figure 30. 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 201.

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 when you declare the function block in the function block tab.

The editor automatically inserts [1] when you specify 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 that can have extensible parameters and the maximum number of parameters for each type, see Control Builder online help.

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 you to organize your parameters and document their purposes.

Table 7. 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.
EDIT	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. Be careful not to connect a variable to a parameter with the keyword EDIT. Use a literal instead.

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.
- Your own libraries, in which you can add your own types. Copies of template types (data types, function block types and control module types), from the standard libraries can be modified and also added into your own libraries.

The following operations are relevant to both library types:

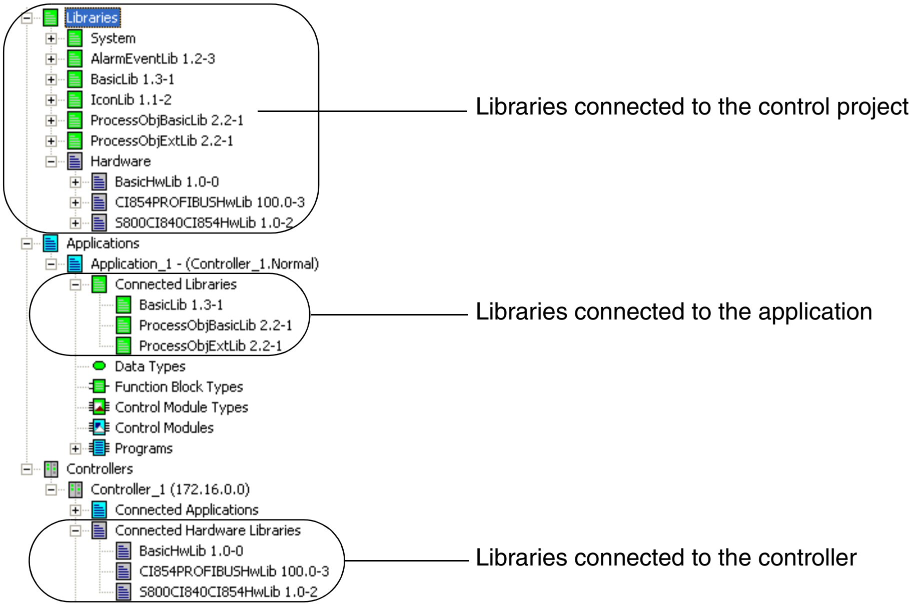
- Libraries must be inserted into the control project in which they are used, see Insert Libraries into Control Projects on page 76.
- A library that contains types for applications must be connected to all libraries and applications that use types from the library. Libraries containing the hardware types (units) used in the controller configuration have to be connected to the controller. See Connect Library to Application, Library or Controller on page 77.
- A library can be disconnected from, an application, library or controller, see Disconnect Libraries on page 77.

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 78.
- The state of a library can be changed, see Library States on page 78.
- The version of a library with hardware types can be changed, see Library Password Protection on page 79.
- Types can be added to a library with hardware types, as long as its state is Open, see Add Types to Libraries Used in Applications on page 80 and Add Customized Hardware Types to Library on page 81.
- A library with hardware types can only be deleted if it is not connected to any application, library or controller.
- A library can be password-protected, see Library Password Protection on page 79.

Connect Libraries

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 31.



Libraries connected to the control project

Libraries connected to the application

Libraries connected to the controller

Figure 31. 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**.

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 folder, right-click and select **Connect Library**.

Replace Connected Library

A connected library can be replaced, for example, when you want 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.
3. Click the **Replace** button to confirm.

Disconnect Libraries

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

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.

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.

Create Libraries

To create a new library:

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

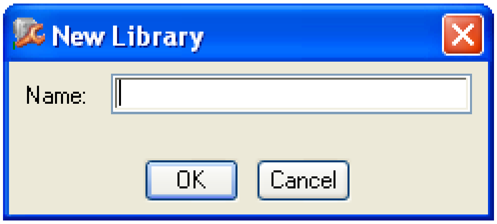


Figure 32. New Library dialog.

2. Enter the name of the new library and click **OK**. The new library is created and inserted into the control project.

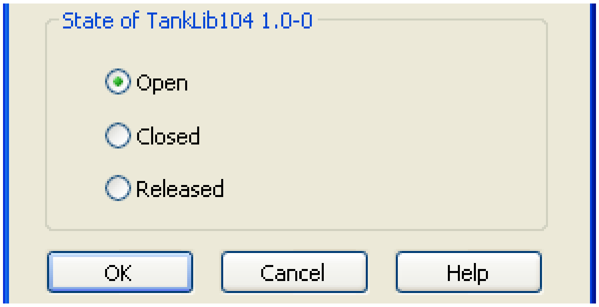
Library States

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.

To change the library state:

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



State of TankLib104 1.0-0

● Open

○ Closed

○ Released

Figure 33. State dialog.

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

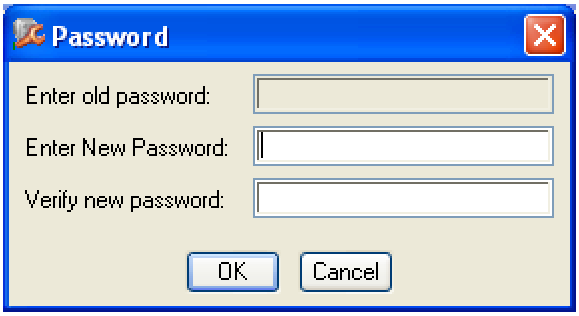
The library state can only change:

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

Library Password Protection

You can password protect your libraries:

1. Right-click the library and select **Properties>Protection**. The Password dialog is displayed.



Password

Enter old password:

Enter New Password:

Verify new password:

Figure 34. Password dialog.

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

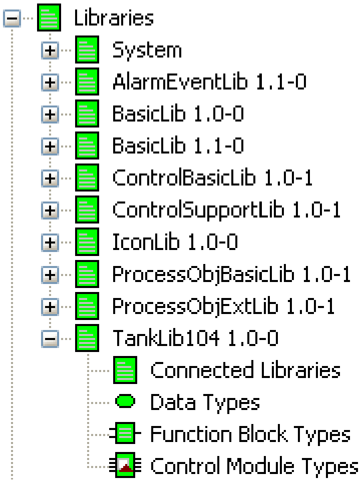
If the library is already password protected, you have to enter the old password before entering a new one. A password may consist of both letters and digits. It must be at least 6 characters long.

3. 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. To add a type to a library:

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



- Libraries
  - System
  - AlarmEventLib 1.1-0
  - BasicLib 1.0-0
  - BasicLib 1.1-0
  - ControlBasicLib 1.0-1
  - ControlSupportLib 1.0-1
  - IconLib 1.0-0
  - ProcessObjBasicLib 1.0-1
  - ProcessObjExtLib 1.0-1
  - TankLib104 1.0-0
    - Connected Libraries
    - Data Types
    - Function Block Types
    - Control Module Types

Figure 35. Library with sub folders.

2. You can now add to the library (see Figure 35):

   a. To connect another library to your library, right-click the Connected Libraries folder and select **Connect Library**.

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

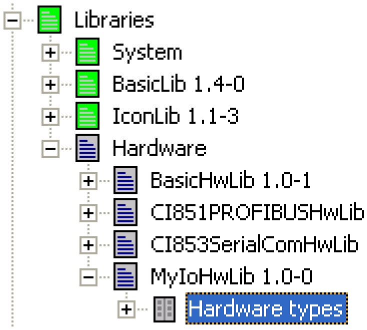
c. To add a type to the library, right-click the folder corresponding to the type you want to add and select the command for creating a new type.

For more information on working with types and project constants, see Application Types and Objects on page 27.

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.



```
Libraries
    System
    BasicLib 1.4-0
    IconLib 1.1-3
    Hardware
        BasicHwLib 1.0-1
        CI851PROFIBUSHwLib 1.0-0
        CI853SerialComHwLib 1.0-0
        MyIoHwLib 1.0-0
            Hardware types
```

Figure 36. Hardware with its libraries.

2. Right-click Hardware types folder under your chosen library, and select **Insert/Replace Hardware Type(s)**.

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 82).

4. The Device Import Wizard starts. Follow the instructions in the wizard.

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.

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 81).
- You can change conversion settings for a previous import, as described in Wizard on page 84.
- 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.

For PROFIBUS GSD-files, *.gsd 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).

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 the Device Import Wizard with PROFIBUS GSD-files, refer to the Industrial IT 800xA - Control and I/O, PROFIBUS DP, Engineering and Configuration manual.

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, below, are associated with the hardware definition file. These files cannot be changed or replaced

File Types Associated with Hardware Types

To display the Additional Files dialog for a hardware type, proceed as follows:

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 85.

The file types, below, are associated with the hardware type and cannot be managed by the user.

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, proceed as follows:

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

The file types, below, are associated with the library.

File Type	Description
Help File	A help file (of *.chm or *.hlp type) can be added, replaced, deleted or extracted, See Help File on page 85
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 or *.hlp 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 F1on the user-defined library or on the customized hardware type, in Project Explorer. For further information about requirements on customized online help, see the Extended Control Software manual.

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

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

Browse to the help file (of *.chm or *.hlp 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.

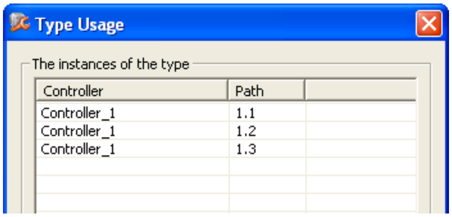
It is not possible to remove a hardware type from a library, if it is used in any hardware configuration.

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**.



Controller	Path
Controller_1	1.1
Controller_1	1.2
Controller_1	1.3

Figure 37. Type Usage for a selected hardware type.

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

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

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 you set the protected attribute, the type interface will be read-only to other users, meaning that only parameter connection is possible. The complete type structure will still be protected from external exposure.

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 be set in Project Explorer.

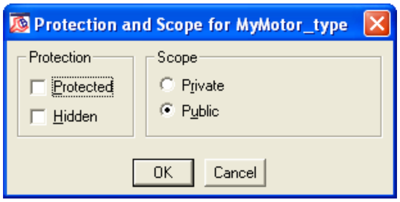
For self-made libraries with password protection, you must enter the password before you make an override, see Library Password Protection on page 79

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

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.



Protection and Scope for MyMotor_type

Protection: □ Protected □ Hidden

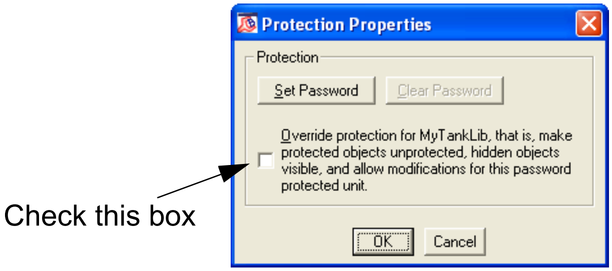
Scope: ○ Private ● Public

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.



Protection Properties

Protection

□ Override protection for MyTankLib, that is, make protected objects unprotected, hidden objects visible, and allow modifications for this password protected unit.

Check this box

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

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 POUs (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 programs (one task per program). The tasks serve as 'work schedulers' for the programs 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 Application Programming, Introduction and Design manual (3BSE044222R101).

If a program 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.

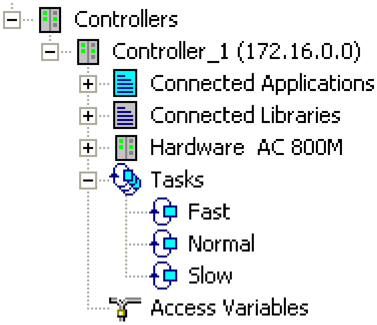
Task Connections

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

Create a New Task

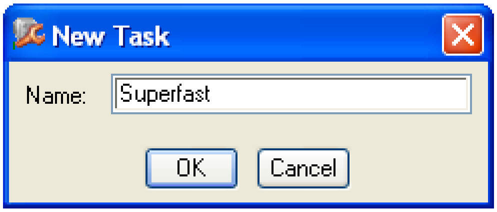
To create and configure a new task:

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.



Name: Superfast

4. Click **OK**.

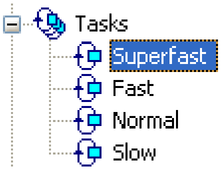
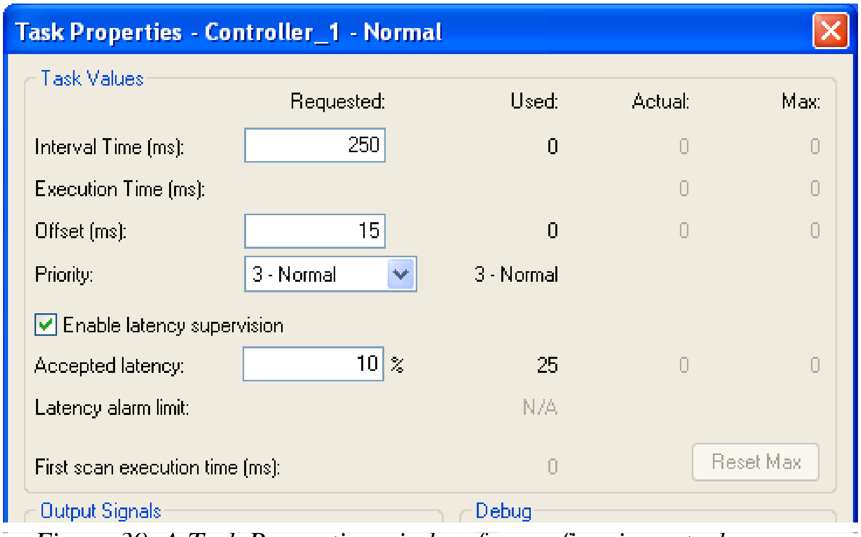


Figure 38. A new task has been created.

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.



Task Properties - Controller_1 - Normal

Task Values

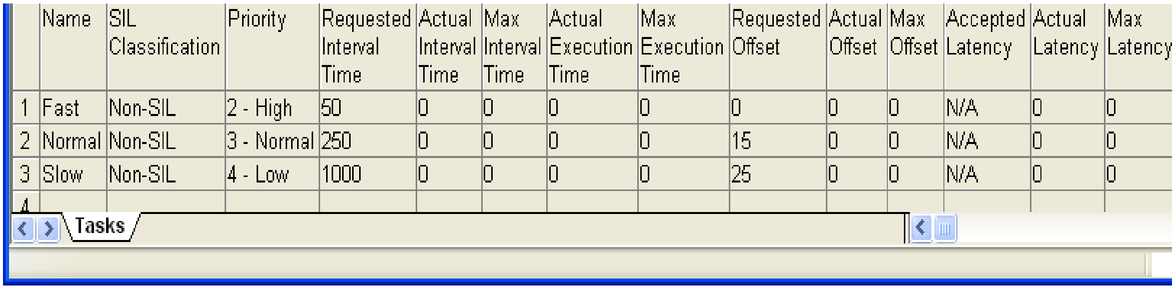
	Requested:	Used:	Actual:	Max:
Interval Time (ms):	250	0	0	0
Execution Time (ms):			0	0
Offset (ms):	15	0	0	0
Priority:	3 - Normal	3 - Normal		

☑ Enable latency supervision

Accepted latency:	10 %	25	0	0
Latency alarm limit:		N/A		
First scan execution time (ms):		0	Reset Max	

Figure 39. A Task Properties window for configuring a task.

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



	Name	SIL Classification	Priority	Requested Interval Time	Actual Interval Time	Max Interval Time	Actual Execution Time	Max Execution Time	Requested Offset	Actual Offset	Max Offset	Accepted Latency	Actual Latency	Max Latency
1	Fast	Non-SIL	2 - High	50	0	0	0	0	0	0	0	N/A	0	0
2	Normal	Non-SIL	3 - Normal	250	0	0	0	0	15	0	0	N/A	0	0
3	Slow	Non-SIL	4 - Low	1000	0	0	0	0	25	0	0	N/A	0	0
4														

The Task Overview window lists all the tasks with each property settings. To change the settings for a certain task:

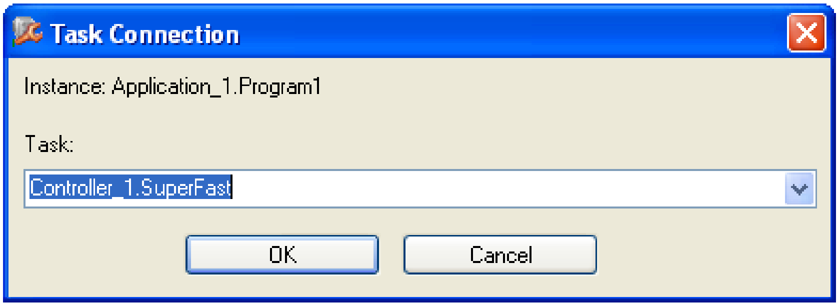
8. Select a task in the Task Overview window and open **Tools > Task Properties**.

Right-click a task directly in the hardware tree and select **Properties** to open the Task Properties window directly.

Connect a Task to a Program

To connect the task SuperFast to Program1:

1. Right-click **Program1** and select **Task Connection**. A ‘Task Connection’ dialog opens.



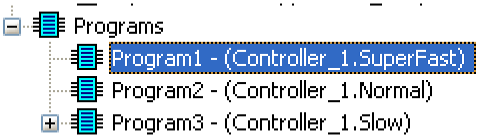
Task Connection

Instance: Application_1.Program1

Task:

Controller_1.SuperFast

2. Select a task from the drop-down menu (here SuperFast) and click **OK**.



Programs
Program1 - (Controller_1.SuperFast)
Program2 - (Controller_1.Normal)
Program3 - (Controller_1.Slow)

Figure 40. 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 61.

Task Execution

There are three 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 95.
- *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 96.

All POUs connected to a task execute with the same *priority*, *interval time* and *offset*.

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.

Interval Time

The interval time, during which the program is executed, is set in the Task Properties dialog. 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 41 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.

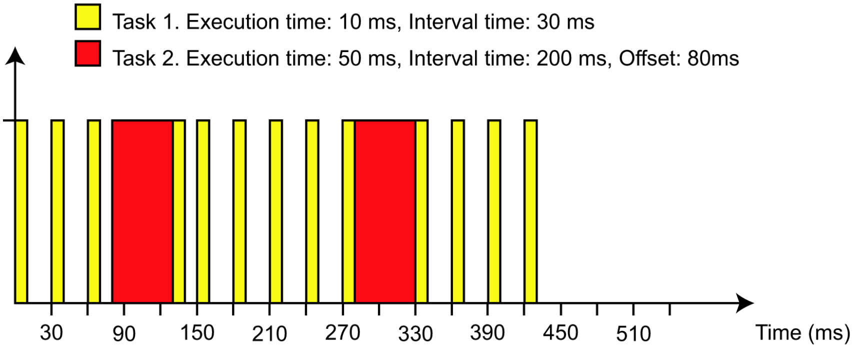


Figure 41. Execution of two tasks with the same priority.

In Figure 42, 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.

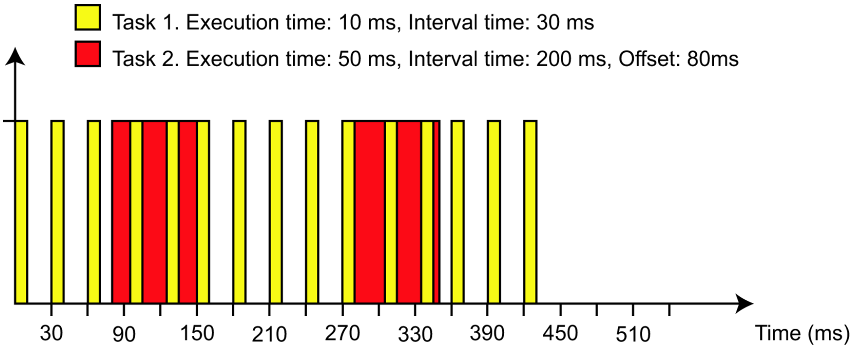


Figure 42. Execution of two tasks with different priorities.

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.

Turning off Task Collision warnings

You can turn off the task collision warning from the Project Explorer.

1. Righ-click the Project item and select **Settings > Compilation Warnings** from the context manu. A Compilation warnings dialog 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.

In Figure 43 and Figure 44, 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 43), 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.

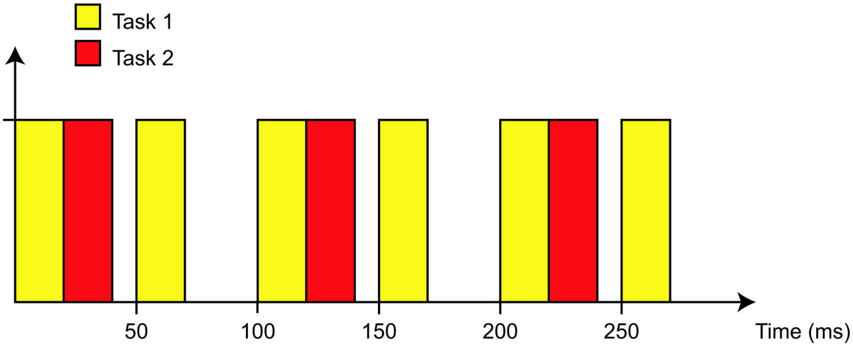


Figure 43. No offset. The two tasks have the same priority, but different interval times (50 and 100 ms).

If task 2 is assigned an offset, as in Figure 44, neither task is delayed, and the actual interval time for task 2 will not be affected by task 1.

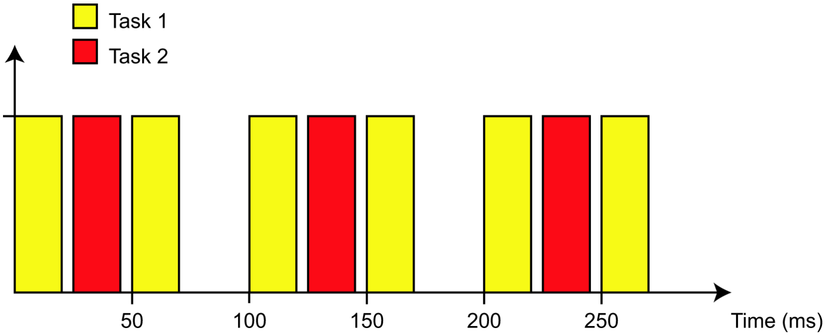


Figure 44. Offset is set on task 2. The two tasks have the same priority, but different interval times (50 and 100 ms) and are thus executed at the requested times.

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. When a task has been executed, the start of its next execution is synchronized to time 0 (the time when the controller began to execute). This means that if the time is t when the task has finished execution, the task will be scheduled to execute at time $(n + 1)$ * (interval time). However, if (interval time) – d, is less than 10 ms, then the task will be scheduled to execute at time $(n + 2)$ * (interval time).

$$t = n * \text{(interval time)} + d, \quad 0 <= d < \text{interval time},$$

If offset > 0, then If offset > d, the start of the next execution will be at a time n* (interval time) + offset. If offset < d, the start of the next execution will be at a time $(n + 1)$*(interval time) + 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.

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 45).

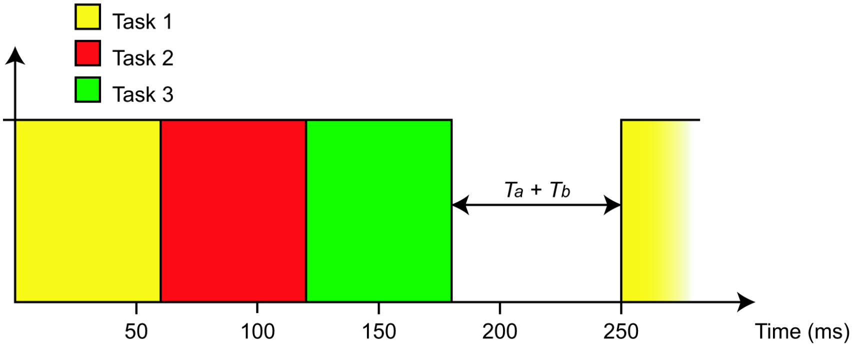


Figure 45. The result of having no offset for three tasks with long execution times. The gap (T_a+T_b) is the time available for the execution of other functions, for example communication.

The offset mechanism can be used to make the time gaps more frequent (see Figure 46).

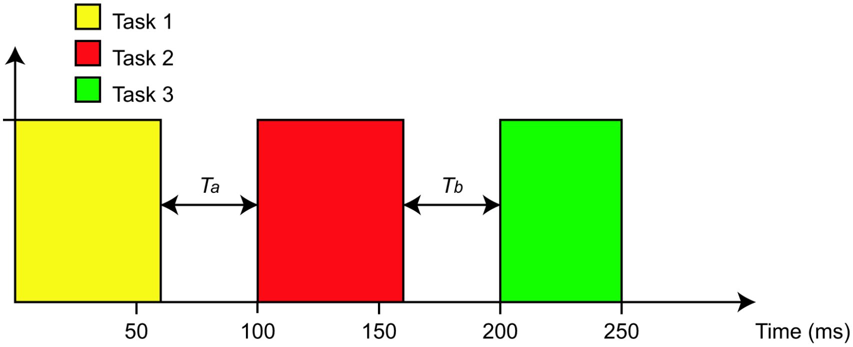


Figure 46. The result of assigning offset to tasks 2 and 3, is that the time available for the execution of other functions occurs more often (T_a).

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 Application Programming manual.

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, while the Latency function is configured per task via the Task Properties dialog. Both Overrun and the Latency function uses the Error Handler to report any errors.

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:

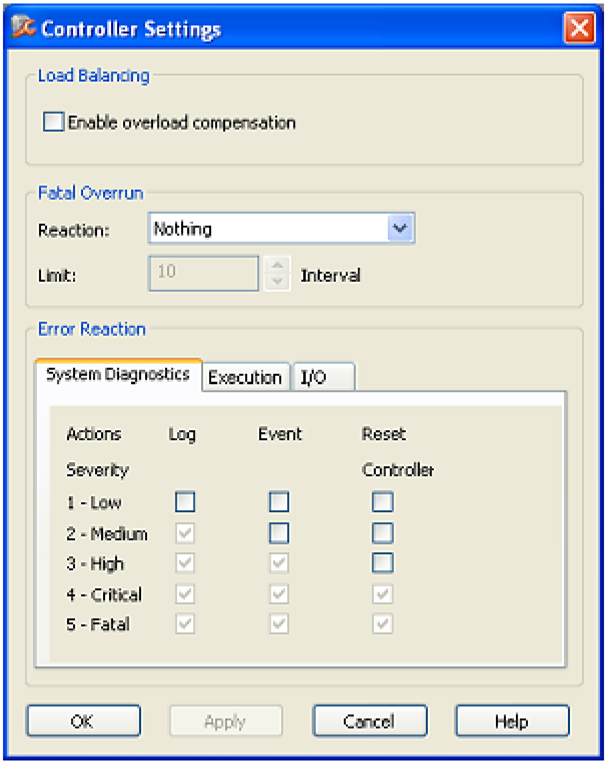
- Nothing,
- Stop Application,
- Reset Controller.

In an AC 800M 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 105.

Configuring Overrun Supervision

Overrun supervision is set for each controller in the Controller Settings dialog. 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 opens.



Controller Settings

Load Balancing

□ Enable overload compensation

Fatal Overrun

Reaction: Nothing

Limit: 10 Interval

Error Reaction

System Diagnostics | Execution | I/O

Actions Severity	Log	Event	Reset Controller
1 - Low	□	□	□
2 - Medium	☑	□	□
3 - High	☑	☑	□
4 - Critical	☑	☑	☑
5 - Fatal	☑	☑	☑

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 Error Handler Settings in Controllers on page 217).

7. Click **OK**.

If overrun errors occur, re-program the faulty task to decrease load.

Latency Supervision

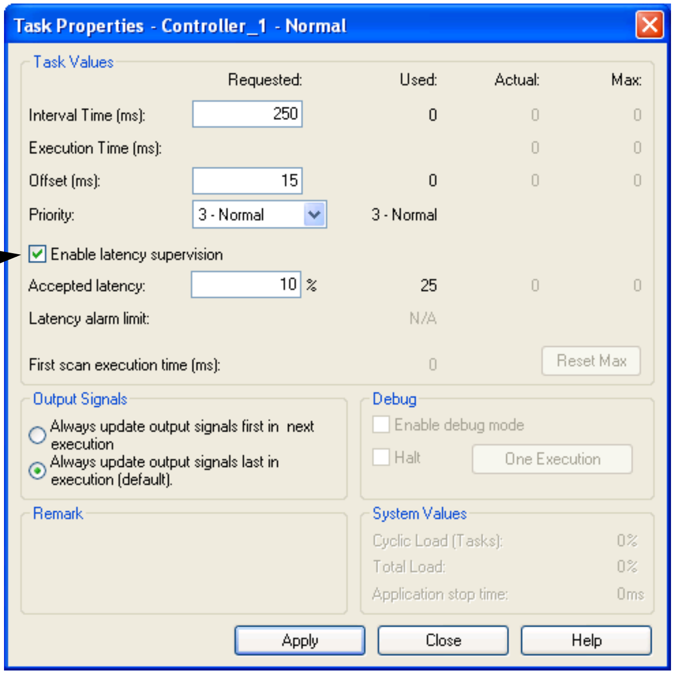
Latency occurs when the execution of a task is delayed, that is, the task starts to execute later than scheduled. The latency function will supervise your tasks (start on time on each cyclic load), and detect if a task starts sooner or later than scheduled.

Latency is activated in the Task Properties dialog, where you set the acceptable latency in percent (accepted latency in percentage of the cycle 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. To select Latency Supervision for a task, follow these steps:

1. Expand the Hardware tree, until you find **Tasks**.
2. Right-click a task and select **Properties** from the pop-up menu. A ‘Task Properties’ dialog opens.



Enable Latency supervision check box

Task Properties - Controller_1 - Normal

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		
☑ Enable latency supervision				
Accepted latency:	10 %	25	0	0
Latency alarm limit:		N/A		
First scan execution time (ms):		0		

Output Signals

○ Always update output signals first in next execution

● Always update output signals last in execution (default).

Debug

□ Enable debug mode

□ Halt

Remark

System Values

Cyclic Load (Tasks): 0%

Total Load: 0%

Application stop time: 0ms

3. Select Latency, (check **Enable latency supervision** check box).
4. 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.
5. Click **Apply**. Note how the actual latency time changes if the accepted latency percentage exceeds 10 %.
6. Click **OK**.

If latency error occurs, re-program the faulty task to decrease load.

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.

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 * IntervalTime).

This means that if IntervalTime is set to 100 ms or higher (100 * 100 ms = 10 seconds), tasks will be aborted if they have not been executed within 10 seconds.

If IntervalTime has been set to <100 ms, tasks will be aborted if they are not executed within (100 * 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%.

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 (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 (right-click the task in Project Explorer, and select **Properties**).

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.

Search and Navigation

The Search and Navigation function makes it possible for the user to search for symbols (see Symbol and Definition on page 111) in a project, by using advanced queries, for example, to find out where a certain variable is used in an application.

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 can also be generated (see Reports on page 119).

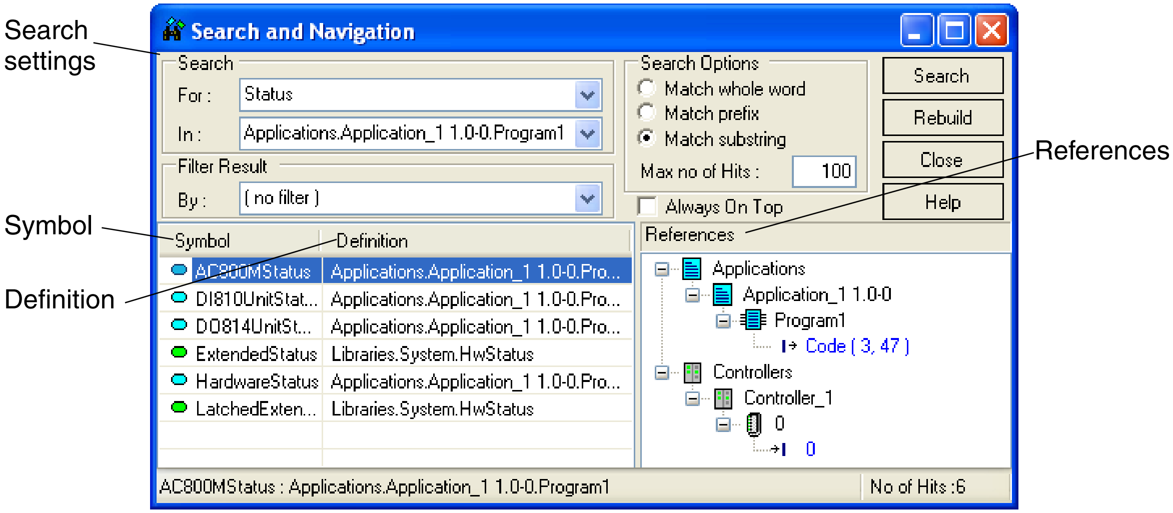
The Search and Navigation function is available in offline, online and test mode. For information on search and navigation in online mode, see Search and Navigation in Online and Test Mode on page 210.

Search and Navigation Dialog

The Search and Navigation dialog mainly consists of Search settings, Symbol, Definition and References. All Search settings are remembered and will be applied next time the dialog is used (until Control Builder is shut down).

The Search and Navigation dialog 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.



Search
For : Status
In : Applications.Application_1 1.0-0.Program1

Filter Result
By : (no filter)

Search Options
○ Match whole word
○ Match prefix
● Match substring
Max no of Hits : 100
□ Always On Top

Symbol	Definition
AC800MStatus	Applications.Application_1 1.0-0.Pro...
DI810UnitStat...	Applications.Application_1 1.0-0.Pro...
DO814UnitSt...	Applications.Application_1 1.0-0.Pro...
ExtendedStatus	Libraries.System.HwStatus
HardwareStatus	Applications.Application_1 1.0-0.Pro...
LatchedExten...	Libraries.System.HwStatus

References
- Applications
  - Application_1 1.0-0
    - Program1
      - Code (3, 47)
- Controllers
  - Controller_1
    - 0
      - 0

AC800MStatus : Applications.Application_1 1.0-0.Program1 No of Hits :6

Figure 47. The Search and Navigation dialog

Search Settings

The Search part of the dialog 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 116) and the **Rebuild** button rebuild the Search data base (see Search Data on page 119).

Search For:

In the Search For text field you enter the symbols to search for (see Symbol and Definition on page 111). 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.

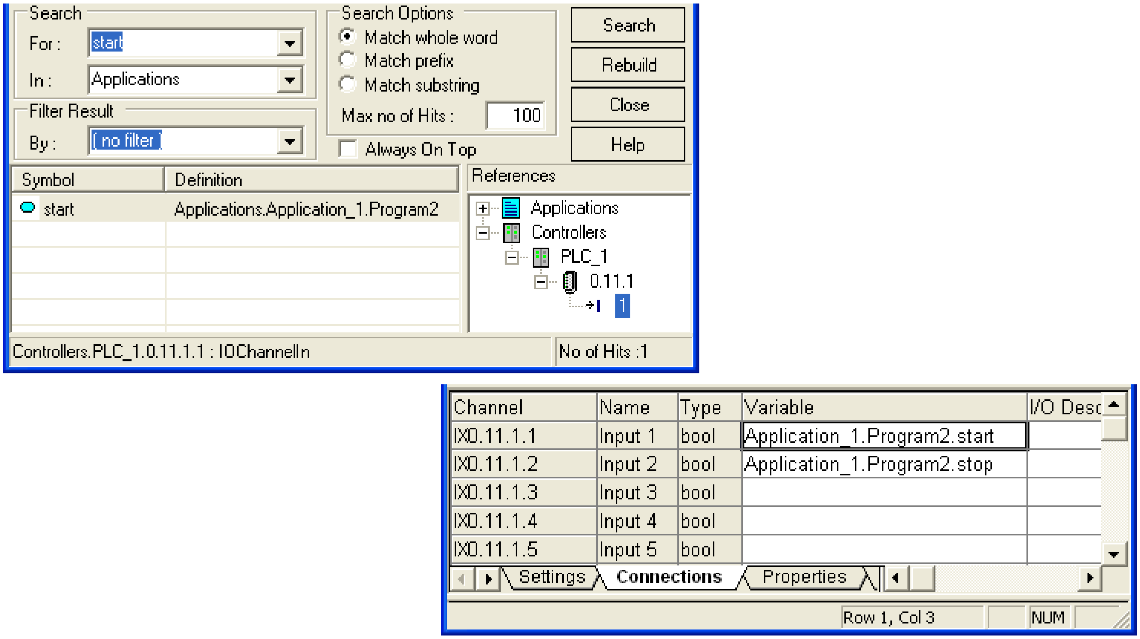
The text in the Search and Navigation Dialog on page 108, *Applications.Application_1 1.0-0.Program1* performs a search in Program1 of Application_1. This search also finds symbols from libraries, because the *HWStatus* data type is used in Program1.

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 48, a search for the variable “start” is performed to find out which I/O channel it is connected to. “start” is connected to channel 1 in hardware on position 0.11.3. By double-clicking on I/O channel (1), in References pane, you navigate to the I/O unit editor there “start” is connected.



Search

For : start

In : Applications

Filter Result

By : (no filter)

Search Options

● Match whole word

○ Match prefix

○ Match substring

Max no of Hits : 100

□ Always On Top

Symbol	Definition
start	Applications.Application_1.Program2

References

- Applications
- Controllers
  - PLC_1
    - 0.11.1
      - 1

Controllers.PLC_1.0.11.1.1 : IOChannelIn

No of Hits :1

Channel	Name	Type	Variable	I/O Desc
IX0.11.1.1	Input 1	bool	Application_1.Program2.start	
IX0.11.1.2	Input 2	bool	Application_1.Program2.stop	
IX0.11.1.3	Input 3	bool		
IX0.11.1.4	Input 4	bool		
IX0.11.1.5	Input 5	bool		

Row 1, Col 3

NUM

Figure 48. (Part of Search and Navigation dialog 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

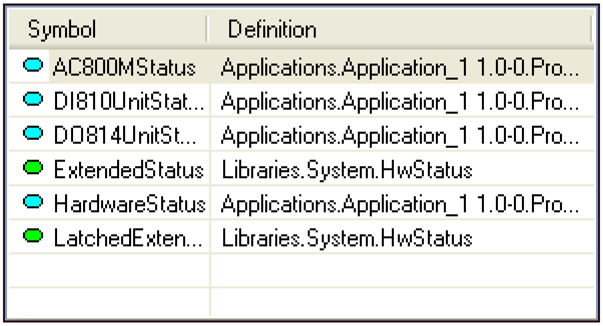
A click on the **Search** button performs the search according to the settings. The search result will be shown.

Always on Top

If Always on Top is checked, the Search and Navigation dialog is placed in front of all other Windows dialogs.

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	Applications.Application_1 1.0-0.Pro...
● DI810UnitStat...	Applications.Application_1 1.0-0.Pro...
● DO814UnitSt...	Applications.Application_1 1.0-0.Pro...
● ExtendedStatus	Libraries.System.HwStatus
● HardwareStatus	Applications.Application_1 1.0-0.Pro...
● LatchedExten...	Libraries.System.HwStatus

Figure 49. 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.

Examples of symbols are:

- hardware channels, access variables, project constants, variables, global variables, external variables, parameters, extensible parameters, 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

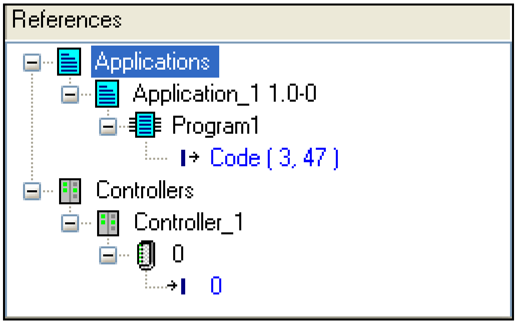
Right-click a Definition to get the context menu selections.

- **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 119.

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.



References

- Applications
  - Application_1 1.0-0
    - Program1
      - Code (3, 47)
- Controllers
  - Controller_1
    - 0
      - 0

Figure 50. The References part of the Search and Navigation dialog.

In the example in Figure 50, the *AC 800M* symbol is used at two locations:

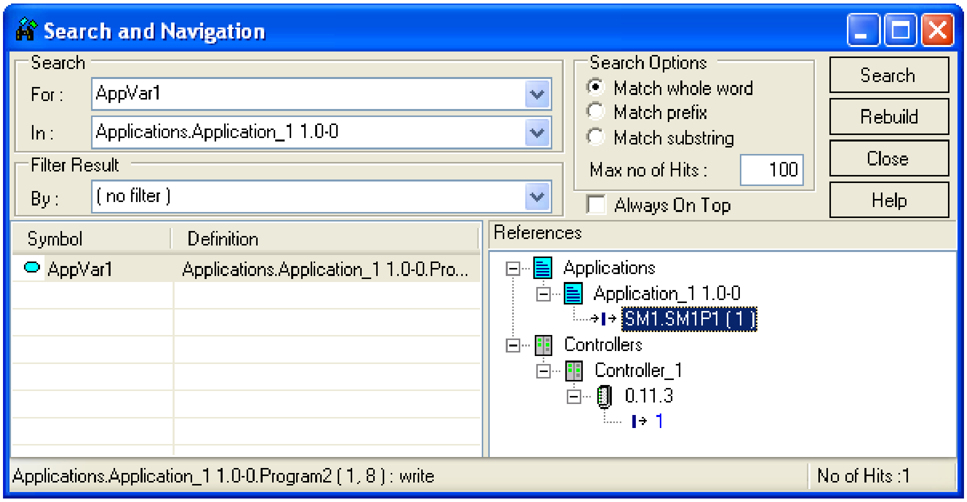
- at line 3, position 47, in Code code block of Program1.
- in channel 0 of unit at position 0 in Controller_1.

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.

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 or a function block.



Search and Navigation

Search
For : AppVar1
In : Applications.Application_1 1.0-0

Filter Result
By : (no filter)

Search Options
● Match whole word
○ Match prefix
○ Match substring
Max no of Hits : 100
□ Always On Top

Symbol	Definition
AppVar1	Applications.Application_1 1.0-0.Pro...

References
- Applications
  - Application_1 1.0-0
    - SM1.SM1P1 (1)
- Controllers
  - Controller_1
    - 0.11.3
      - 1

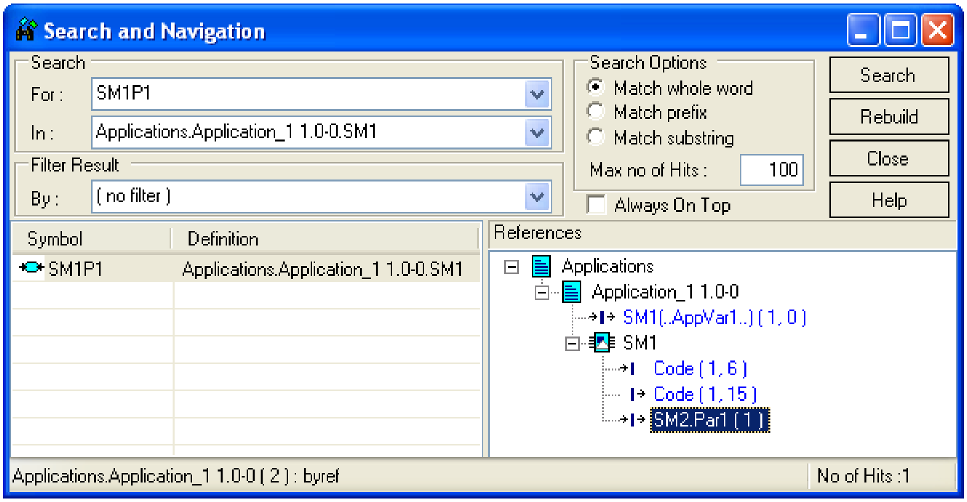
Applications.Application_1 1.0-0.Program2 (1, 8) : write
No of Hits :1

Figure 51. A search for Variable “AppVar1” in Applications.

In the example in Figure 51, *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 52. A new search is performed.

The Execute Search Instantly check box (see Execute Search Instantly on page 117) has to be checked. If it is not checked, the user must click the **Search** button.



Search and Navigation

Search
For : SM1P1
In : Applications.Application_1 1.0-0.SM1

Filter Result
By : (no filter)

Search Options
● Match whole word
○ Match prefix
○ Match substring
Max no of Hits : 100
□ Always On Top

Symbol	Definition
SM1P1	Applications.Application_1 1.0-0.SM1

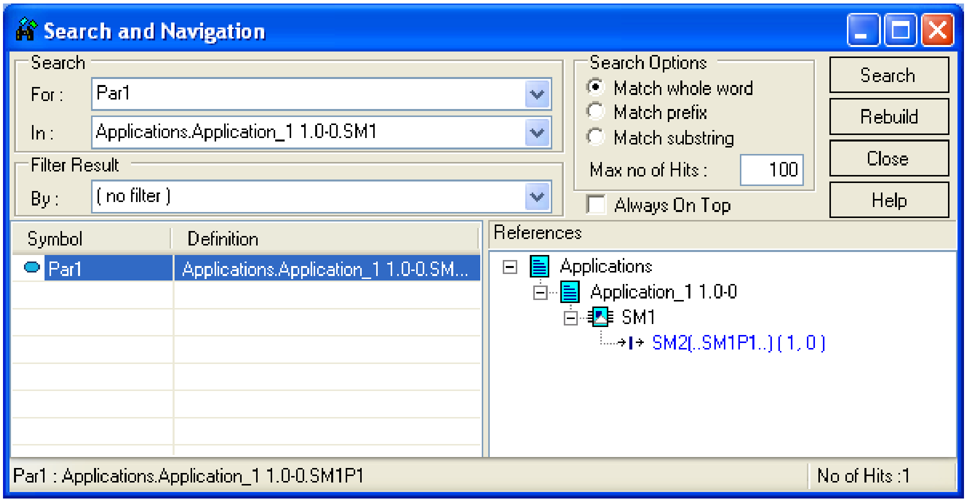
References
- Applications
  - Application_1 1.0-0
    - SM1(..AppVar1..) (1, 0)
    - SM1
      - Code (1, 6)
      - Code (1, 15)
      - SM2.Par1 (1)

Applications.Application_1 1.0-0 (2) : byref
No of Hits :1

Figure 52. 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**.



Search

For : Par1

In : Applications.Application_1 1.0-0.SM1

Filter Result

By : (no filter)

Search Options

● Match whole word

○ Match prefix

○ Match substring

Max no of Hits : 100

□ Always On Top

Symbol	Definition
Par1	Applications.Application_1 1.0-0.SM...

References

- Applications
  - Application_1 1.0-0
    - SM1
      - →|→ SM2(..SM1P1..) (1, 0)

Par1 : Applications.Application_1 1.0-0.SM1P1

No of Hits :1

Figure 53. 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.
∘A	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)

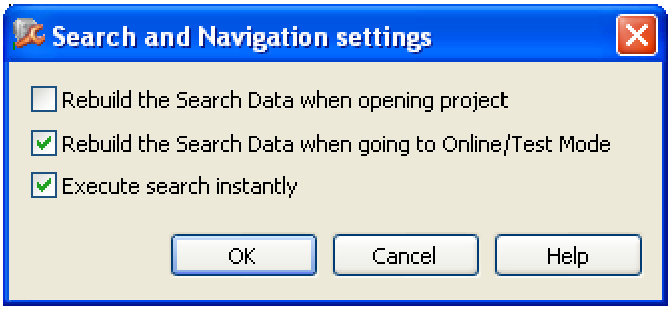
When navigating to an editor or a dialog the window already can be active, but minimized as well as hidden behind other windows.

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 has two settings regarding update of the search database and one setting for a “quick-search” function.

Select **Tools > Setup > Station > Search and Navigation Settings** to display the Search and Navigation settings dialog.



Search and Navigation settings

☐ Rebuild the Search Data when opening project

☑ Rebuild the Search Data when going to Online/Test Mode

☑ Execute search instantly

Figure 54. The Search and Navigation settings dialog with default setting.

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 M. This check box is by default not checked.

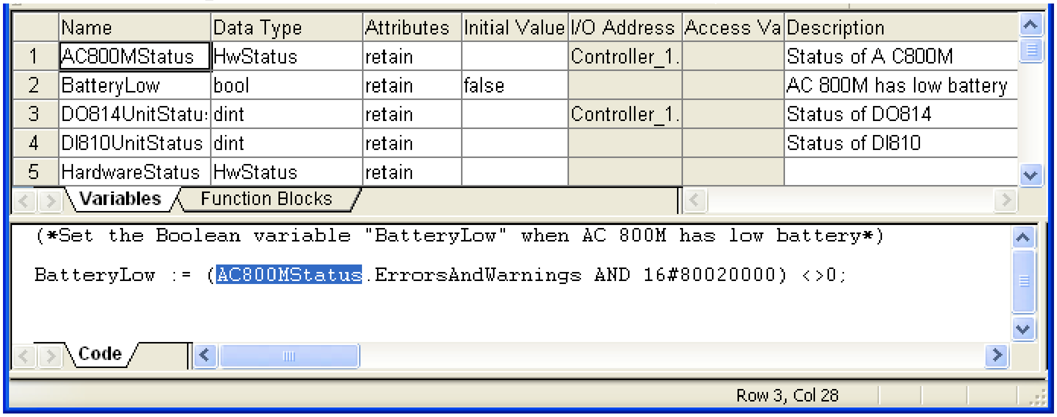
Rebuild the Search Data when Going to Online/Test Mode

When this option is checked, search data is rebuilt when Control Builder M 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 checked.

Execute Search Instantly

When this option is checked, the Search and Navigation dialog 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:



	Name	Data Type	Attributes	Initial Value	I/O Address	Access Va	Description
1	AC800MStatus	HwStatus	retain		Controller_1.		Status of A C800M
2	BatteryLow	bool	retain	false			AC 800M has low battery
3	DO814UnitStatu:	dint	retain		Controller_1.		Status of DO814
4	DI810UnitStatus	dint	retain				Status of DI810
5	HardwareStatus	HwStatus	retain				

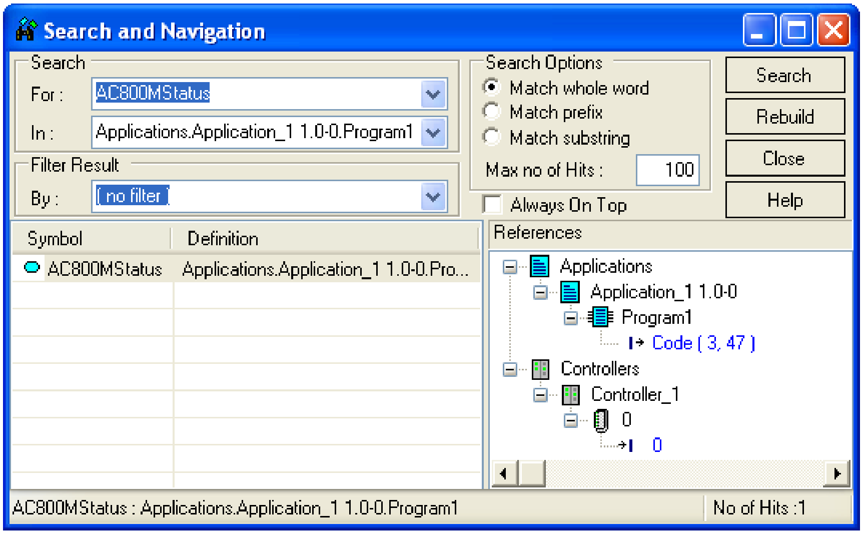
```
(*Set the Boolean variable "BatteryLow" when AC 800M has low battery*)

BatteryLow := (AC800MStatus.ErrorsAndWarnings AND 16#80020000) <>0;
```

Row 3, Col 28

Figure 55. Selection of the AC800MStatus in Program1.

1. Click on the *AC800MStatus* variable in code block *Code* in *Program1*.
2. Select **Edit > Search** (or **Alt-F12**).



Search and Navigation

Search

For : AC800MStatus

In : Applications.Application_1 1.0-0.Program1

Filter Result

By : (no filter)

Search Options

● Match whole word

○ Match prefix

○ Match substring

Max no of Hits : 100

□ Always On Top

Symbol	Definition
AC800MStatus	Applications.Application_1 1.0-0.Pro...

References

```
Applications
  Application_1 1.0-0
    Program1
      Code ( 3, 47 )
Controllers
  Controller_1
    0
      0
```

AC800MStatus : Applications.Application_1 1.0-0.Program1 No of Hits :1

Figure 56. The search result after performing above steps.

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.

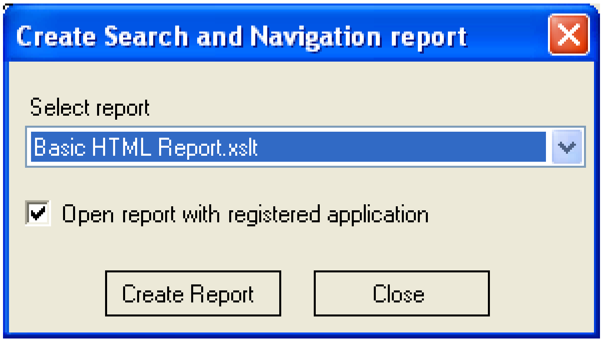
It is possible to perform a manual rebuild of the Search data base, in offline mode. The Search data base can be rebuild in the following ways:

- 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....**



Create Search and Navigation report

Select report

Basic HTML Report.xslt

☑ Open report with registered application

Figure 57. 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.

Compact Flash

Compact Flash (CF) is a memory card that can be easily inserted to the card slot located at the front of AC 800M controllers.

The memory card keeps the application saved in unpredictable power supply environments, which rapidly wear out the controller's battery strength. It is also convenient for transporting new or updated applications over great distances without depending on traditional battery support.

For more information about the AC 800M controller, see the subsection ‘Product Overview’ in the AC 800M Controller Hardware manual.

The card will be activated and read after a long controller reset (or power failure) and your application(s) can be loaded into the new controller host without performing a monitored application download from a Control Builder station.

Compact Flash does not support distributed applications; hence you cannot use the memory card in a controller that run distributed applications.

Saving Cold Retain Values on Files

The cold retain values used by Compact Flash can either be saved cyclic via 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 CF card. Thus, not be confused with the cold retain values saved by Control Builder or OPC Server during a download.

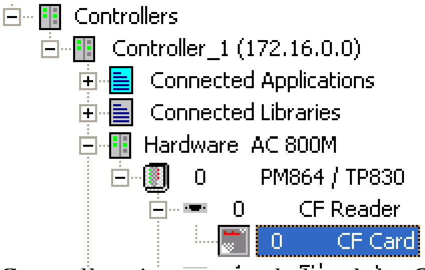
Read more about the SaveColdretain function block type in Control Builder online help.

Adding Compact Flash to Hardware

From the Project Explorer:

Make sure that BasicHwLib is inserted under Hardware and that it is connected to the controller.

1. Expand the **Controllers** item until you reach the **CF Reader** item (Figure 58).
2. Right-click the **CF Reader** and select **Insert Unit** from the context menu. A dialog will open.
3. Select **CF Card** in the dialog and click **Insert**.
4. Click **Close**.



```
Controllers
  Controller_1 (172.16.0.0)
    Connected Applications
    Connected Libraries
    Hardware  AC 800M
      0       PM864 / TP830
        0       CF Reader
          0       CF Card
```

Figure 58. The Controllers item expanded and the CF Card connected to the CF Reader item.

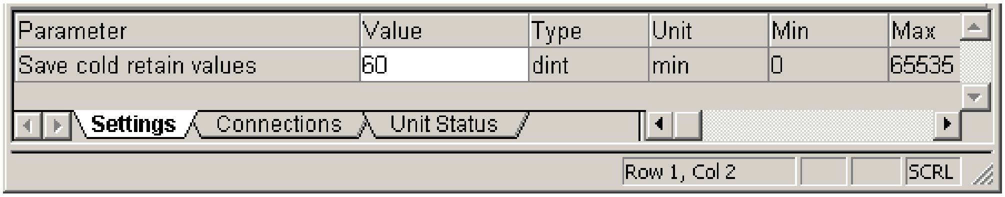
Setting Up Cyclic Save of Cold Retain Values

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 122.

This subsection will describe how to save cold retain values cyclic. Provided that you have added the CF Card to your Hardware tree according to Figure 58, do the following:

1. Double-click the **CF 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 59.



Parameter	Value	Type	Unit	Min	Max
Save cold retain values	60	dint	min	0	65535

Figure 59. Settings for Save cold retain values (default 60 min.).

To prevent Compact Flash for 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 CF 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 Compact Flash memory card.

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 **Tool > Save “ColdRetain” Values**. A ‘Save “ColdRetain” Values’ dialog will open.
2. Click **Save**. The cold retain values have been saved with your application and you are now ready to download to the Compact Flash memory card.

Downloading the Application to Compact Flash

Before you can download your application to Compact Flash, you must connect an external Compact Flash Writer to your Control Builder PC. The writer is normally connected to the PCs USB port.

From the Project Explorer, make sure your project is in offline mode:

1. Insert a Compact Flash card in the Writer slot.
2. Right-click controller and select **Compact Flash** from the context-menu. A 'Pick removable media' dialog window will open.
3. Select Writer and click **OK**. The Control Builder will write the application to the Compact Flash card.

In case the Control Builder source code files is to be placed on the 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 cards, one in each CPU). Copy (in Windows Explorer) the downloaded application (two folders) from the CF 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

Configuration Load means to load a controller configuration, all applications and their corresponding cold retain values from Compact Flash. After a configuration load, the application can read all the critical process (cold retain) values that was stored on Compact Flash.

If or when a control system is breaking-down due to power failure, and no battery backup in the controller is available, Compact Flash can re-boot the control system with the latest and the most efficient cold retain values.

In case of a redundant processor unit configuration, it is recommended to insert a CF in both CPUs.

Application Version Check

If the application version in the controller is not identical with the version in Compact Flash or vice verse; a warning message will alert and no more cold retain values can be saved.

Reports

Difference Report

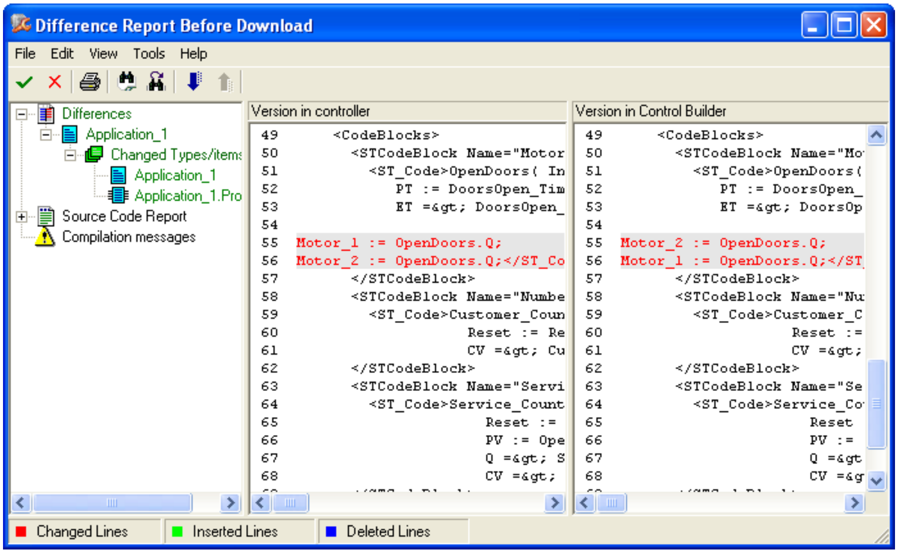
If the Difference Report function is enabled, the Difference Report Before Download dialog displays (in the same dialog):

- Difference report,
- Source code report.

Based on the information presented in the reports you can either accept or reject the changes, if you want the download to be carried out or cancelled.

The function is enabled/disabled by right-click the control project folder (root object) and select **Settings > Difference Report**.

Difference report shows the difference between data downloaded to the controller and the data present in Control Builder, see Figure 60. 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).



```
    <CodeBlocks>
      <STCodeBlock Name="Motor
        <ST_Code>OpenDoors( In
           PT := DoorsOpen_Tim
           ET =&gt; DoorsOpen_

Motor_1 := OpenDoors.Q;
Motor_2 := OpenDoors.Q;</ST_Co
      </STCodeBlock>
      <STCodeBlock Name="Numbe
        <ST_Code>Customer_Coun
                   Reset := Re
                   CV =&gt; Cu
      </STCodeBlock>
      <STCodeBlock Name="Servi
        <ST_Code>Service_Count
                     Reset :=
                     PV := Ope
                     Q =&gt; S
                     CV =&gt;
```

```
    <CodeBlocks>
      <STCodeBlock Name="Mo
        <ST_Code>OpenDoors(
           PT := DoorsOpen_
           ET =&gt; DoorsOp

Motor_2 := OpenDoors.Q;
Motor_1 := OpenDoors.Q;</ST
      </STCodeBlock>
      <STCodeBlock Name="Nu
        <ST_Code>Customer_C
                   Reset :=
                   CV =&gt;
      </STCodeBlock>
      <STCodeBlock Name="Se
        <ST_Code>Service_Co
                     Reset
                     PV :=
                     Q =&gt
                     CV =&g
```

Figure 60. Difference report before download

The difference report presents found differences, see Table 8.

Table 8. Differences presented in difference report.

Data	Example
Application data	User defined types, start values, execution order, connected libraries.
Controller configuration data	Access variables, hardware units, hardware definition files (hwd files), task properties, connected applications, settings from external configuration tool (such as Fieldbus Builder FF), controller settings (error handler), communication interval settings.
Project constants	
System variable *EnableStringTransfer*	

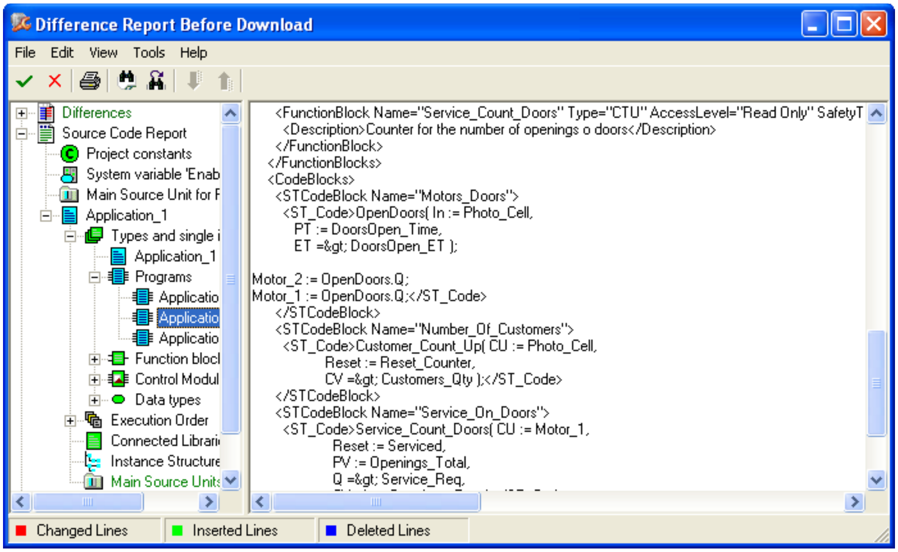
To reduce the compilation time during download of a project to a controller, it is possible to exclude the start values from the difference report. The start value analysis is enabled/disabled via **Tools > Difference Report Settings**.

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.

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.



```
    <FunctionBlock Name="Service_Count_Doors" Type="CTU" AccessLevel="Read Only" SafetyT
     <Description>Counter for the number of openings o doors</Description>
    </FunctionBlock>
   </FunctionBlocks>
   <CodeBlocks>
    <STCodeBlock Name="Motors_Doors">
     <ST_Code>OpenDoors( In := Photo_Cell,
       PT := DoorsOpen_Time,
       ET =&gt; DoorsOpen_ET );

Motor_2 := OpenDoors.Q;
Motor_1 := OpenDoors.Q;</ST_Code>
    </STCodeBlock>
    <STCodeBlock Name="Number_Of_Customers">
     <ST_Code>Customer_Count_Up( CU := Photo_Cell,
            Reset := Reset_Counter,
            CV =&gt; Customers_Qty );</ST_Code>
    </STCodeBlock>
    <STCodeBlock Name="Service_On_Doors">
     <ST_Code>Service_Count_Doors( CU := Motor_1,
             Reset := Serviced,
             PV := Openings_Total,
             Q =&gt; Service_Req,
```

Figure 61. 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**).

The source code report presents information as shown in Table 9.

Table 9. Information presented in source code report .

Data	Example
Application data	User defined types, execution order, connected libraries.
Controller configuration data	Access variables, hardware units, hardware definition files (hwd files), task properties, connected applications, controller settings (error handler), communication interval settings, structural changes, simulation mark, signature.

Table 9. Information presented in source code report (Continued).

Data	Example
Project constants	
System variable *EnableStringTransfer*	

Information about execution order 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 128.

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 128.

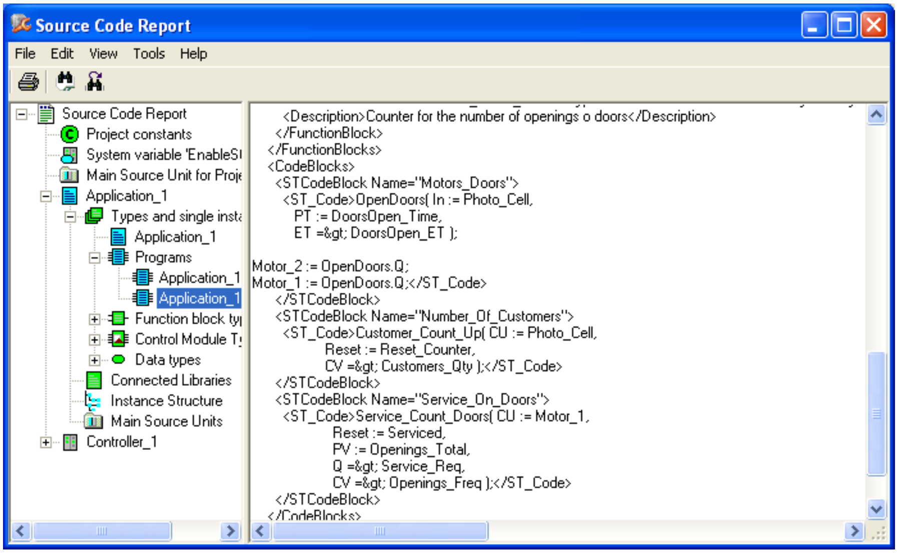
Reports Generated at Download

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 124 and Source Code Report on page 125.

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**.



```
      <Description>Counter for the number of openings o doors</Description>
    </FunctionBlock>
  </FunctionBlocks>
  <CodeBlocks>
    <STCodeBlock Name="Motors_Doors">
      <ST_Code>OpenDoors( In := Photo_Cell,
         PT := DoorsOpen_Time,
         ET =&gt; DoorsOpen_ET );

Motor_2 := OpenDoors.Q;
Motor_1 := OpenDoors.Q;</ST_Code>
    </STCodeBlock>
    <STCodeBlock Name="Number_Of_Customers">
      <ST_Code>Customer_Count_Up( CU := Photo_Cell,
                Reset := Reset_Counter,
                CV =&gt; Customers_Qty );</ST_Code>
    </STCodeBlock>
    <STCodeBlock Name="Service_On_Doors">
      <ST_Code>Service_Count_Doors( CU := Motor_1,
                 Reset := Serviced,
                 PV := Openings_Total,
                 Q =&gt; Service_Req,
                 CV =&gt; Openings_Freq );</ST_Code>
    </STCodeBlock>
  </CodeBlocks>
```

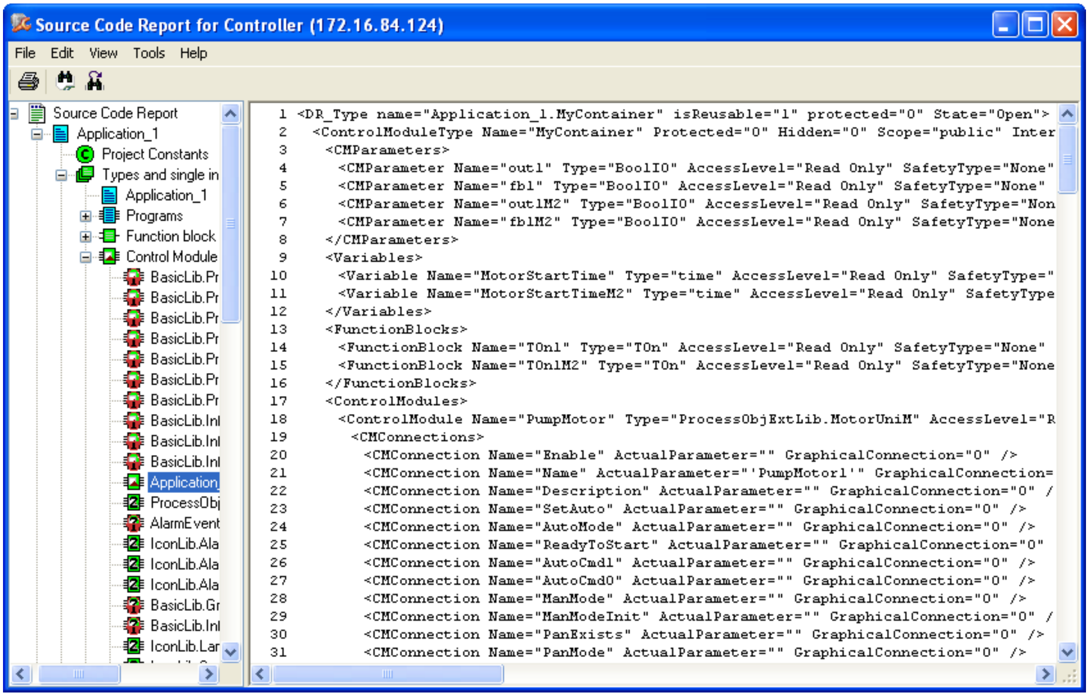
Figure 62. Source code report generated without prior compilation

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**.



```
<DR_Type name="Application_1.MyContainer" isReusable="1" protected="0" State="Open">
 <ControlModuleType Name="MyContainer" Protected="0" Hidden="0" Scope="public" Inter
  <CMParameters>
   <CMParameter Name="out1" Type="BoolIO" AccessLevel="Read Only" SafetyType="None"
   <CMParameter Name="fb1" Type="BoolIO" AccessLevel="Read Only" SafetyType="None"
   <CMParameter Name="out1M2" Type="BoolIO" AccessLevel="Read Only" SafetyType="Non
   <CMParameter Name="fb1M2" Type="BoolIO" AccessLevel="Read Only" SafetyType="None
  </CMParameters>
  <Variables>
   <Variable Name="MotorStartTime" Type="time" AccessLevel="Read Only" SafetyType="
   <Variable Name="MotorStartTimeM2" Type="time" AccessLevel="Read Only" SafetyType
  </Variables>
  <FunctionBlocks>
   <FunctionBlock Name="TOn1" Type="TOn" AccessLevel="Read Only" SafetyType="None"
   <FunctionBlock Name="TOn1M2" Type="TOn" AccessLevel="Read Only" SafetyType="None
  </FunctionBlocks>
  <ControlModules>
   <ControlModule Name="PumpMotor" Type="ProcessObjExtLib.MotorUniM" AccessLevel="R
    <CMConnections>
     <CMConnection Name="Enable" ActualParameter="" GraphicalConnection="0" />
     <CMConnection Name="Name" ActualParameter="'PumpMotor1'" GraphicalConnection=
     <CMConnection Name="Description" ActualParameter="" GraphicalConnection="0" /
     <CMConnection Name="SetAuto" ActualParameter="" GraphicalConnection="0" />
     <CMConnection Name="AutoMode" ActualParameter="" GraphicalConnection="0" />
     <CMConnection Name="ReadyToStart" ActualParameter="" GraphicalConnection="0"
     <CMConnection Name="AutoCmd1" ActualParameter="" GraphicalConnection="0" />
     <CMConnection Name="AutoCmd0" ActualParameter="" GraphicalConnection="0" />
     <CMConnection Name="ManMode" ActualParameter="" GraphicalConnection="0" />
     <CMConnection Name="ManModeInit" ActualParameter="" GraphicalConnection="0" /
     <CMConnection Name="PanExists" ActualParameter="" GraphicalConnection="0" />
     <CMConnection Name="PanMode" ActualParameter="" GraphicalConnection="0" />
```

Figure 63. Source code report generated for project in controller.

Project Documentation

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.

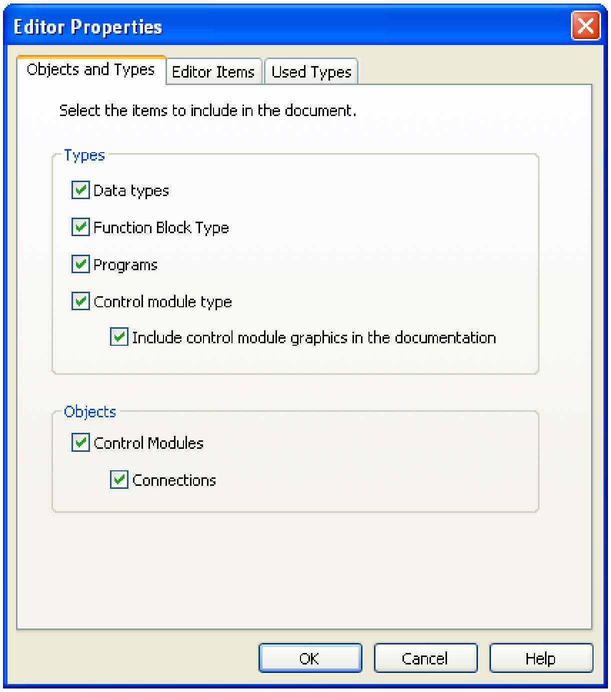
All project documentation will be connected to a standard template. But you can create templates of your own for the documentation.

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.

Printing Project Documentation

To print documentation, in Project Explorer:

1. Right-click any object in the tree view and select **Documentation**. A ‘Documentation’ dialog will open.
2. Click **More** to filter information. An ‘Edit Properties’ dialog opens.



Select the items to include in the document.

Types

- ☑ Data types
- ☑ Function Block Type
- ☑ Programs
- ☑ Control module type
  - ☑ Include control module graphics in the documentation

Objects

- ☑ Control Modules
  - ☑ Connections

Figure 64. Editor Properties dialog for filter options.

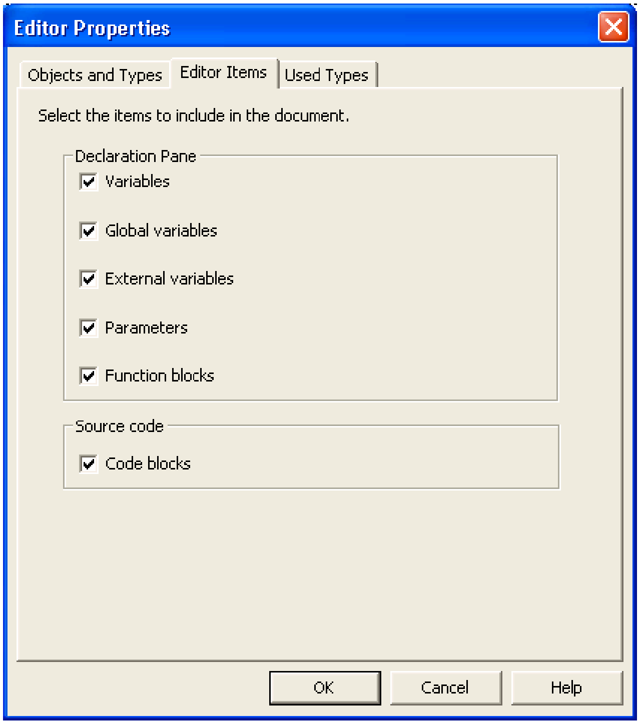
The Editor Properties dialog inside the Documentation function, contains three main areas, which are represented by tabs in the dialog, see Figure 64.

- 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



Editor Properties

Select the items to include in the document.

Declaration Pane

☑ Variables

☑ Global variables

☑ External variables

☑ Parameters

☑ Function blocks

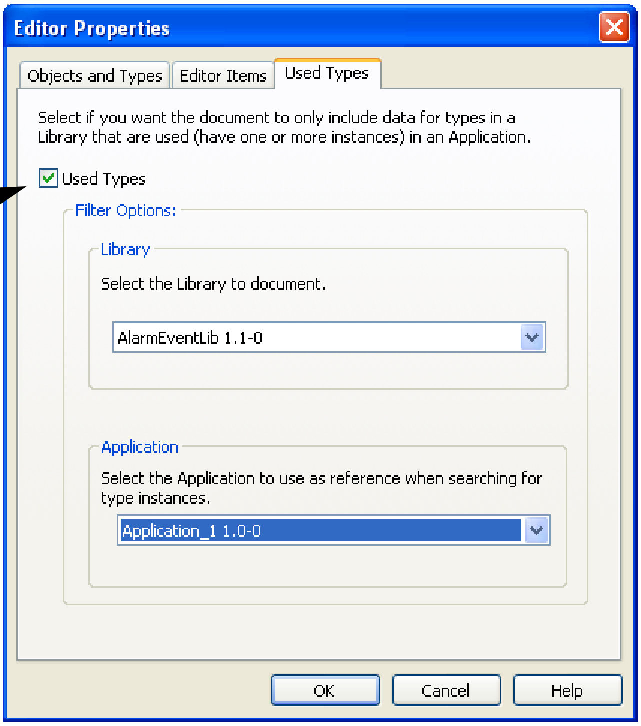
Source code

☑ Code blocks

Figure 65. Editor item 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 and Source code.

Used Types



Select if you want the document to only include data for types in a Library that are used (have one or more instances) in an Application.

☑ Used Types

Filter Options:

Library

Select the Library to document.

AlarmEventLib 1.1-0

Application

Select the Application to use as reference when searching for type instances.

Application_1 1.0-0

Used Types must be checked.

Figure 66. 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 66).

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, please refer to the Extended Control Software manual, 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 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 159.

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 135.
- 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 144 and System Alarm and Event Generation on page 154.

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 156 and Sequence of Events (SOE) on page 147.

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 Condition State Example on page 163. For details on how to use alarm and event functions, see Alarm and Event Functions on page 170. 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 Basic Control Software.

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 136.

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 144.
- 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 144.

There are also system generated alarms and events, see System Alarm and Event Generation on page 154.

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.

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 67, the function block type AlarmCond is used to illustrate the function of the different parameters.

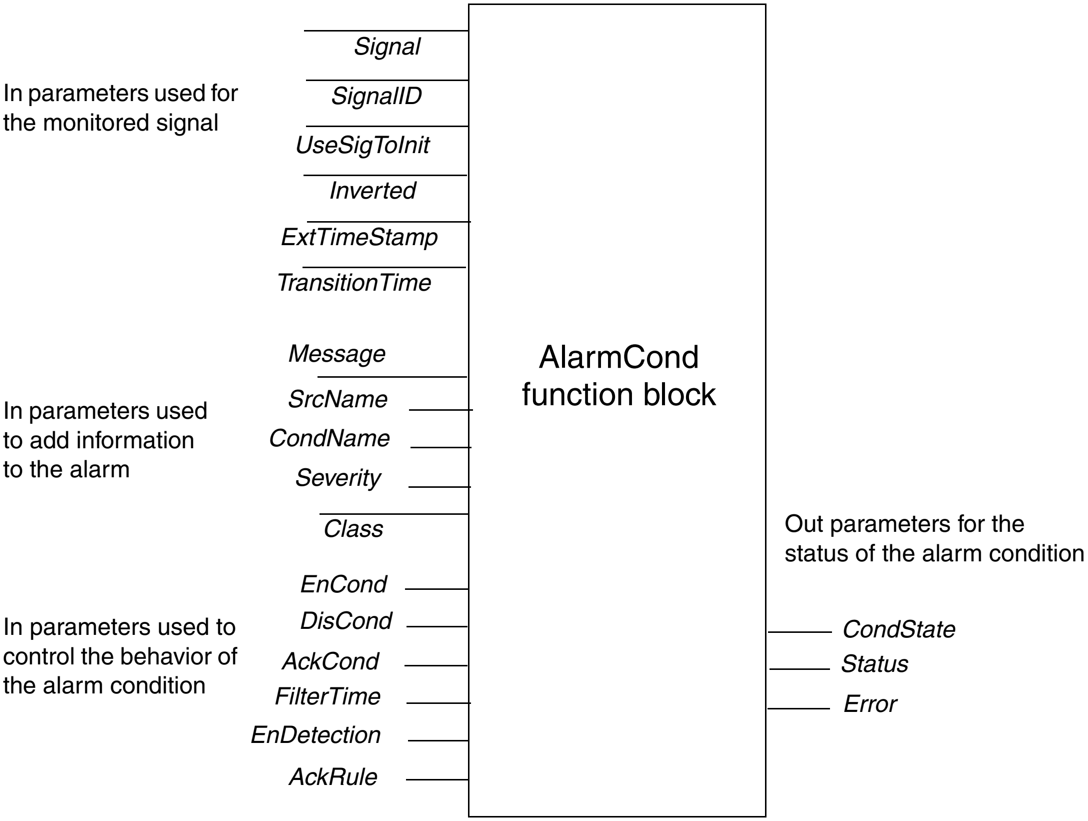

Signal
In parameters used for the monitored signal
SignalID
UseSigToInit
Inverted
ExtTimeStamp
TransitionTime
Message
AlarmCond function block
SrcName
In parameters used to add information to the alarm
CondName
Severity
Class
Out parameters for the status of the alarm condition
EnCond
DisCond
In parameters used to control the behavior of the alarm condition
AckCond
FilterTime
EnDetection
AckRule
CondState
Status
Error

Figure 67. 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 171.

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 144) 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.

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 disabl*e. 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, autodisabling is turned off and alarms can be activated an unlimited number of times.

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 two 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.

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.

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 139. 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 155.

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 Compact Control Builder.

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 Library to represent *all* I/O channels. 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 library, see online help and the Extended Control Software manual.

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 Extended Control Software manual.

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 146.

Alarms and events can be disabled from the interaction windows and from OPC AE, as well as from the application, via interaction parameters.

Inhibit Parameters

The inhibit function is present in the following standard library types.

- Signal library:
  - SignalInReal,
  - SignalReal,
  - SignalInBool,
  - SignalBool.
- Standard Control library:
  - Level6CC,
  - Level4CC,
  - Level2CC.

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 IO)

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 only. For information on enabling/disabling and configuring SOE, see online help for S800 I/O.

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.

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 149) 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.

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.

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 155. This is useful when creating batch records.
- How to handle system alarms and events, see System Alarms and Events on page 156.
- Internal, remote, and external time stamps (Sequence of Events, SOE), including time synchronization, see Time Stamps on page 156.

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 Condition State Example on page 163.

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 the Getting Started manual (3BSE041584R101).

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 Industrial IT, 800xA - Control and I/O, Communication, Protocols and Design (3BSE035982Rxxxx)

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.

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. .

For detailed information on how to configure OPC Server for AC 800M, please refer to the OPC Server for AC 800M manual.

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 five 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 68.

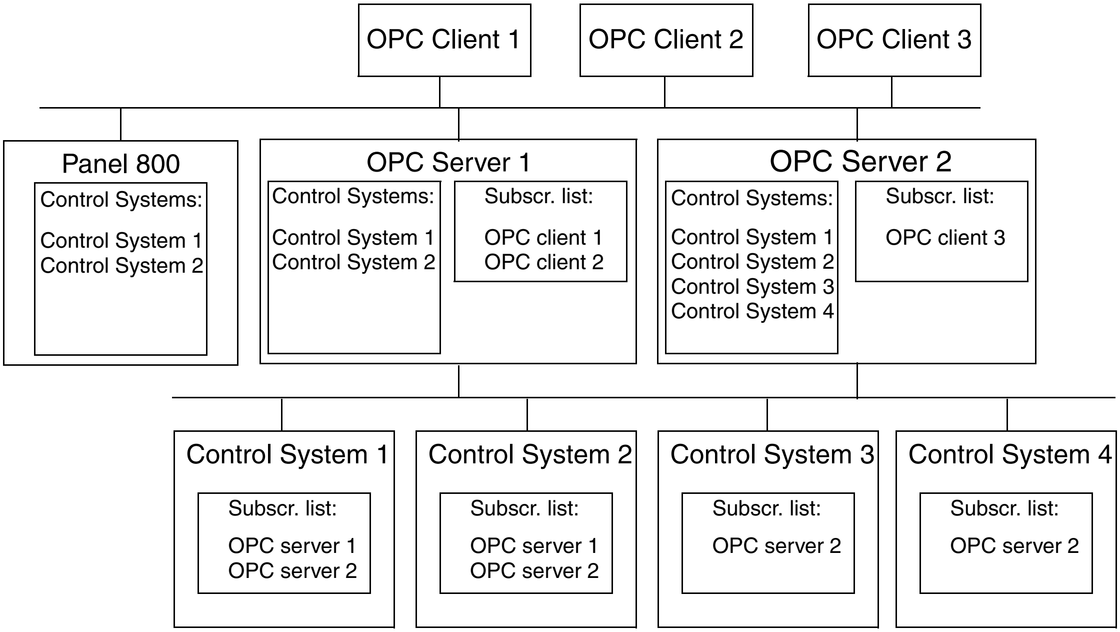


Figure 68. Example of a control network configuration.

Information about how to configure individual OPC servers is found in the OPC Server for AC 800M manual, and in the online help, which can be opened from the OPC server panel.

Buffer Configuration

Alarm and event handling requires a number of buffers. Memory for these buffers must be allocated in the controllers. These settings have to be made for each controller, in the Project Explorer CPU settings tab, see Table 10. Also, see System Diagnostics on page 170.

Table 10. 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 * AE Local printer event queue size * 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 * AE Event subscription queue size * AE Max number of event subscriptions
AE Max number of event subscriptions	Number of subscribing OPC Servers
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

Table 10. Memory planning for buffer configuration (Continued)

Parameter	Comment
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.

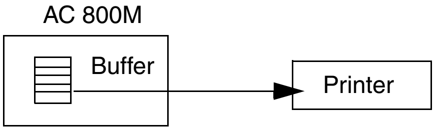


Figure 69. Example of a controller and local printer configuration.

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.

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.

Condition State Example

The following example shows how to use the condition state parameter (*CondState*) to control a pump.

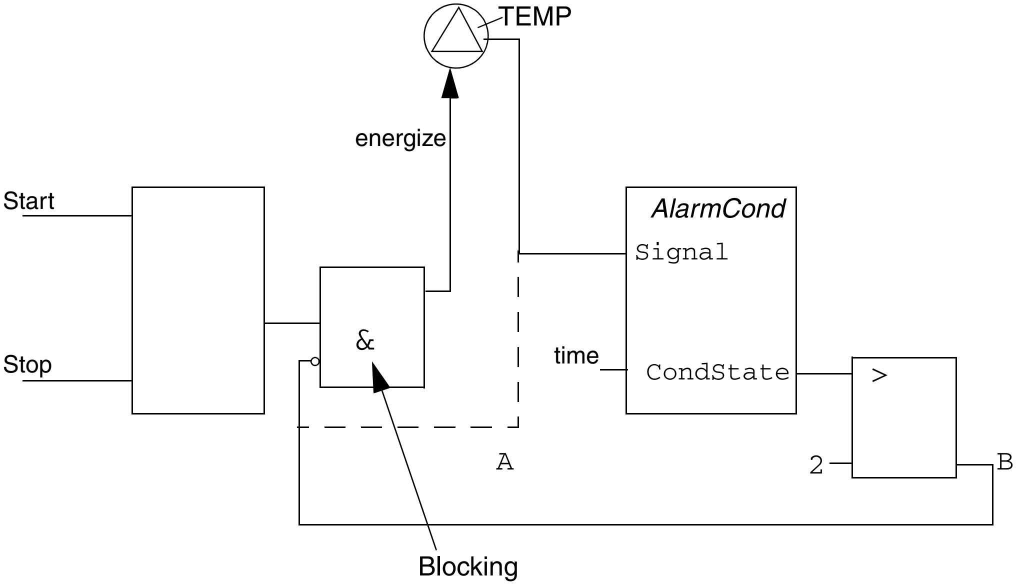


Figure 70. Manipulating the condition state using I/O.

Figure 70 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 142). 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 71):

- 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 *LTLLAct* 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. *LTLLStat* 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 *LTLLAct* 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. *LTLLStat* 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. *LTLLStat* will be set and the lamp will be lit.

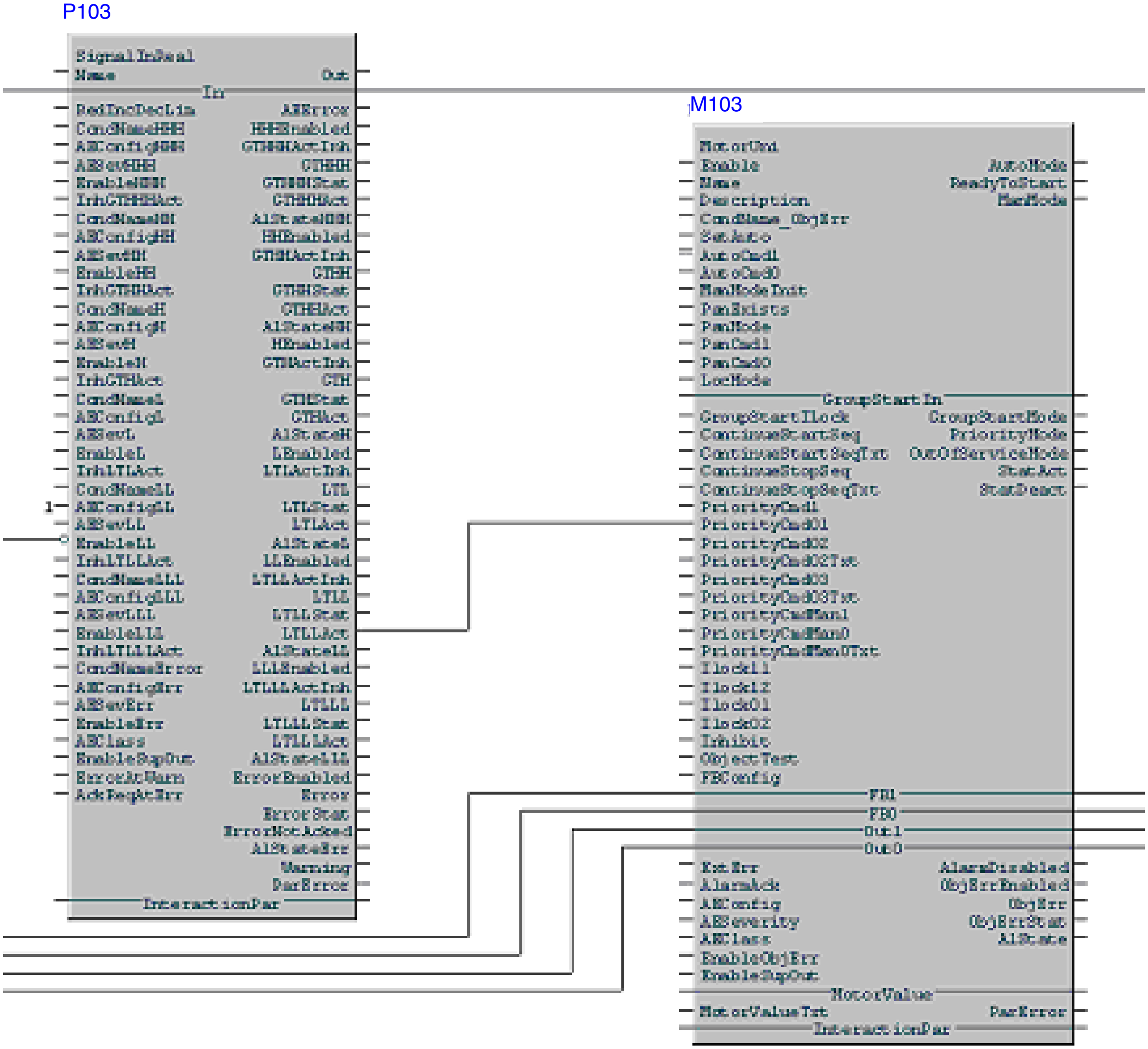


Figure 71. Example of how to implement inhibition of an alarm.

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 166,
- Structured Data – Example 1 on page 168,
- Structured Data – Example 2 on page 168.

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* 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 170.
- 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 171.

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.

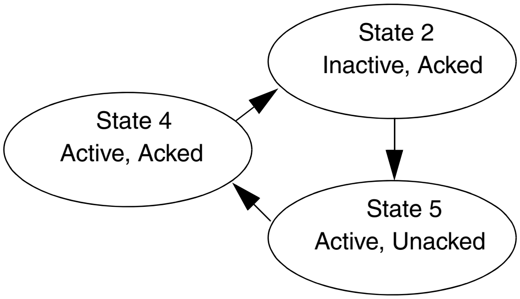


Figure 72. State diagram for enabled alarm conditions with AckRule 1, part 1.

In Figure 72 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.

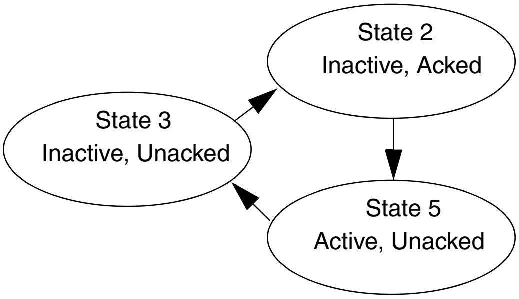


Figure 73. State diagram for enabled alarm conditions with AckRule 1, part 2.

In Figure 73 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.

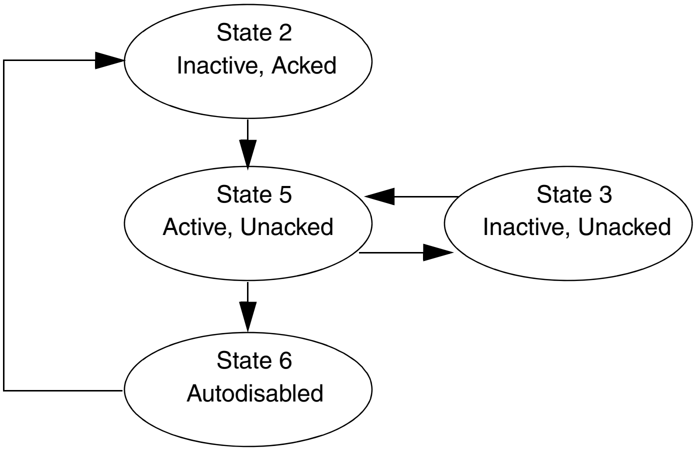


Figure 74. State diagram for enabled alarm conditions with AckRule 1, part 3.

The third instance occurs when an alarm switches between active and inactive without being acknowledged. In Figure 74, 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

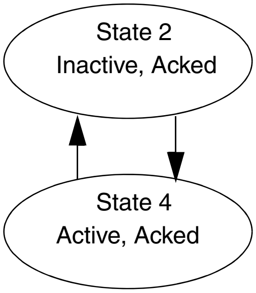


Figure 75. State diagram for enabled alarm conditions with AckRule 2.

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

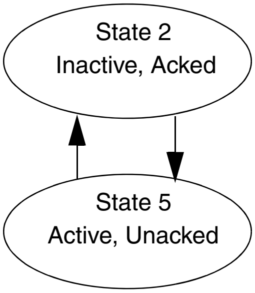


Figure 76. State diagram for enabled alarm conditions with AckRule 3.

Regardless of the signal being monitored, alarm conditions with AckRule 3 changes immediately to is 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

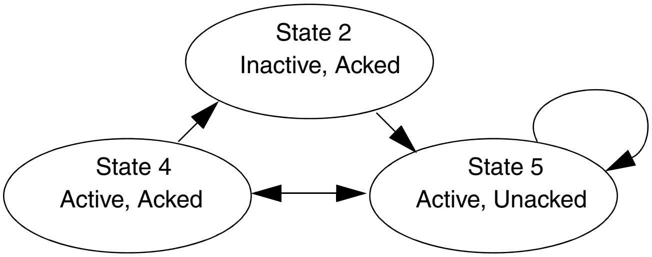


Figure 77. State diagram for enabled alarm conditions with AckRule 5, part 1.

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 77 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.

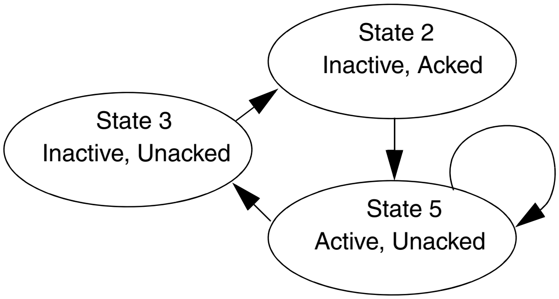


Figure 78. State diagram for enabled alarm conditions with AckRule 5, part 2.

The second instance is shown in Figure 78 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.

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 Communication manual.

This section is split into the following parts:

- Communication Libraries on page 178 gives a brief overview of the Communication standard libraries.
- Control Network on page 185 describes Control Network, which is used to communicate between controllers, engineering stations, and external devices.
- Variable Communication on page 186 describes variable communication briefly, and contains references to more detailed information.
- Reading/Sending Data on page 189 describes reading and sending data.
- Fieldbus Communication on page 195 describes the supported fieldbus protocols briefly.

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 Communication manual, 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.

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.

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 Measuremen	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.
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
Contactor 1	nvoOpCount1	In	Number of switch cycles for contactor 1.
Contactor 2	nvoOpCount2	In	Number of switch cycles for contactor 2.
Contactor 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

Function/Object in MCU	NV name in MCU	Dir.	Description
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 (MMSCommLib) 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.

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.

Normally, communication between controllers has to be defined using access variables and function block types and/or control module types from the MMS Communication library.

For more information on MMS communication, see the Communication manual.

ModBus Communication Library

The ModBus 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.

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 Communication manual.

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.

Serial Communication Library

The Serial Communication library (SerialCommLib) contains function block types for communication with external devices via serial channels with user-defined protocols, for example printers, terminals, scanner pens. You can write an application which controls the characters sent and checks that the correct answer is received, using serial channel handling function blocks.

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 (2x5+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.

Supported Protocols

Table 11 lists all supported protocols.

Table 11. Protocols and 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(3)	COM3 (TP830), CI853
PROFIBUS DP-V1	CI854
DriveBus	CI858
INSUM	CI857

(1) Both master and slave
(2) Master only
(3) Master only

For more information on supported protocols, see the Communication manual.

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 Communication manual. 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 Getting Started manual.

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 Automation System Network manual.

Statistics and Information on Communication

Statistics concerning all MMS communication 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>Show MMS Variables** shows which MMS variables are present in the selected remote system
- **Tools>Maintenance>Show MMS Connections** shows all connections, including information on the type of connection, the destination system, and a number of statistics.

There is also a function block type System Diagnostics that is stored in the Basic library. This function block will (among other things) show 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. 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 and SattBus protocols. Paths to local variables are given using the syntax `ApplicationName.ProgramName.FunctionblockName.VariableName`

For more information about variable communication, see Variables and Parameters on page 48.

Variables are not updated in synchronization with IEC 61131 code. This must be taken into account when designing variable communication.

StartAddr

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 12. StartAddr parameter setting

Protocoll	IEC 61131-3 Direct Addressing	Direct Addressing (Octal, 8# only)
ModBus	%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 See also Siemens 3964R Addresses	%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:

Protocoll	IEC 61131-3 Direct Addressing	Direct Addressing (Octal, 8# only)
ModBus	%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 See also Siemens 3964R Addresses	%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.

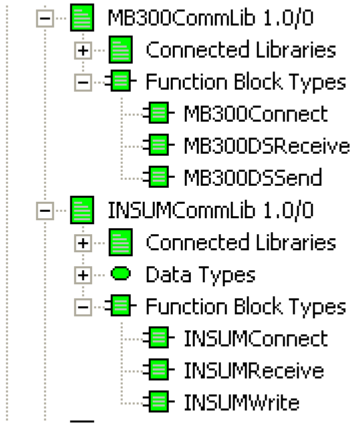
- **Read Types**
  Read types read data (often an access variable) from a target system. The source system (the communication channel) is indicated by the *Id* parameter, which is passed from the corresponding connect function block or control module.

- **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:



```
MB300CommLib 1.0/0
    Connected Libraries
    Function Block Types
        MB300Connect
        MB300DSReceive
        MB300DSSend
INSUMCommLib 1.0/0
    Connected Libraries
    Data Types
    Function Block Types
        INSUMConnect
        INSUMReceive
        INSUMWrite
```

Figure 79. 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.

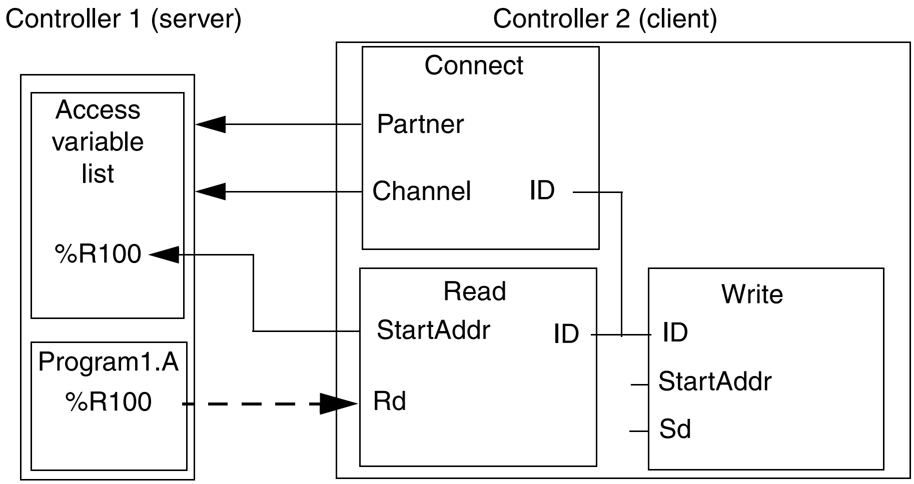


Figure 80. Variable read by controller 2, from controller 1.

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.

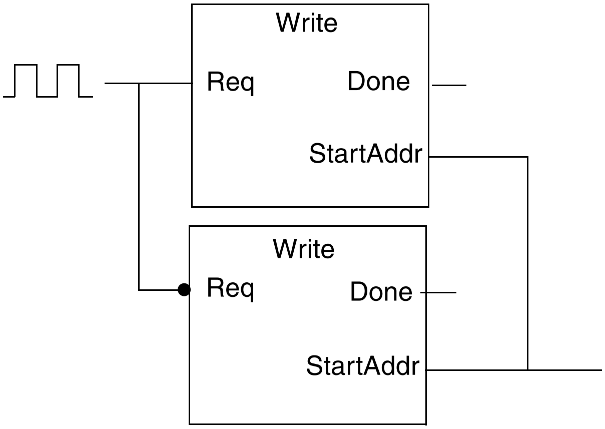


Figure 81. Resetting the Req parameter using two function blocks.

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 193,
- The publisher/subscriber (also called subscriber/provider) concept, see Publisher/Subscriber Communication on page 194.
- There is also the choice between cyclic and asynchronous communication, see Cyclic vs. Asynchronous Communication on page 195.

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 82 shows the principle.

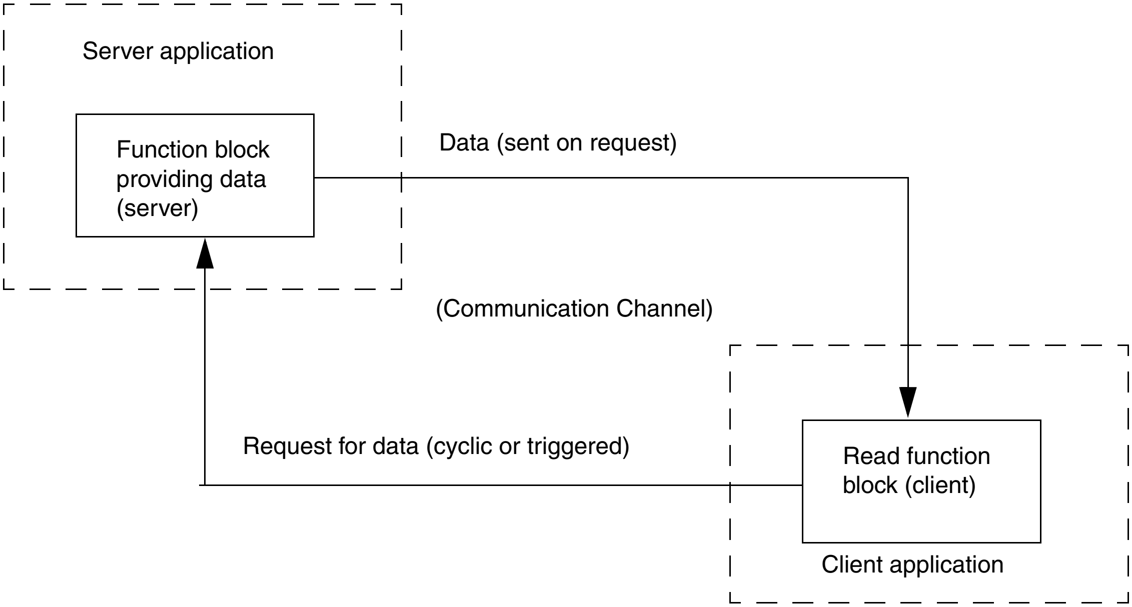

Server application
Function block providing data (server)
Data (sent on request)
(Communication Channel)
Request for data (cyclic or triggered)
Read function block (client)
Client application

Figure 82. 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 83 shows the publisher/subscriber principle.

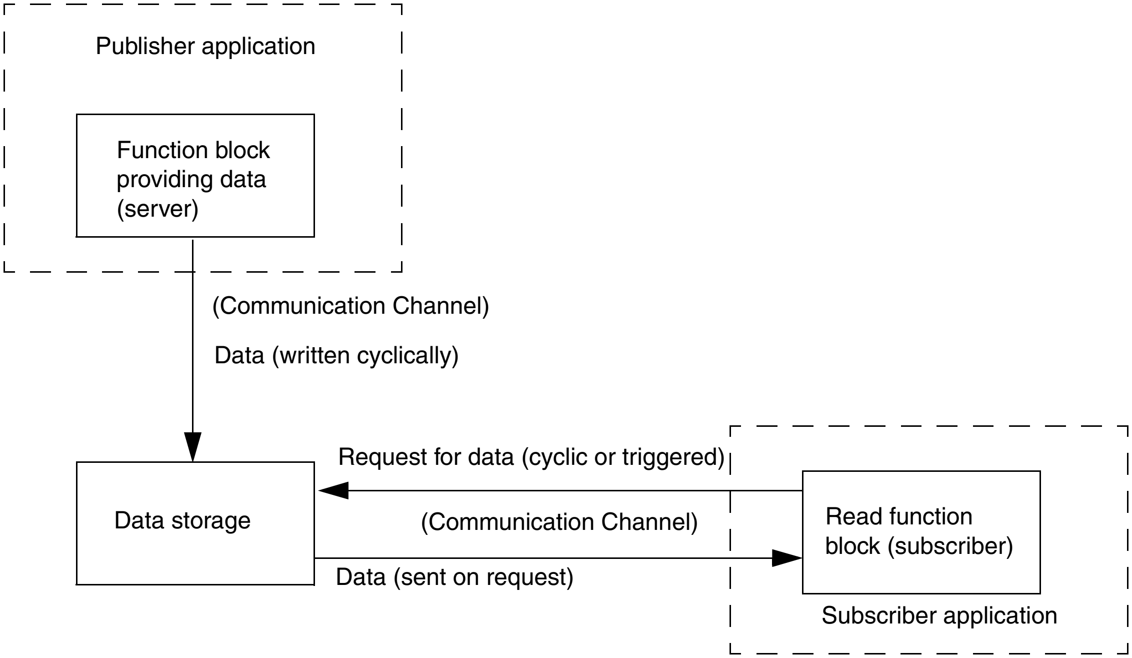

Publisher application
Function block providing data (server)
(Communication Channel)
Data (written cyclically)
Request for data (cyclic or triggered)
Data storage
(Communication Channel)
Read function block (subscriber)
Data (sent on request)
Subscriber application

Figure 83. 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 Application Programming manual.

For information on how to make part of your code execute with a different interval, see Control the Execution of Individual Objects on page 61.

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, please 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.

The following fieldbuses are supported:

- **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.

- **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.

The protocols used by the supported fieldbuses are described in detail in the Communication 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.

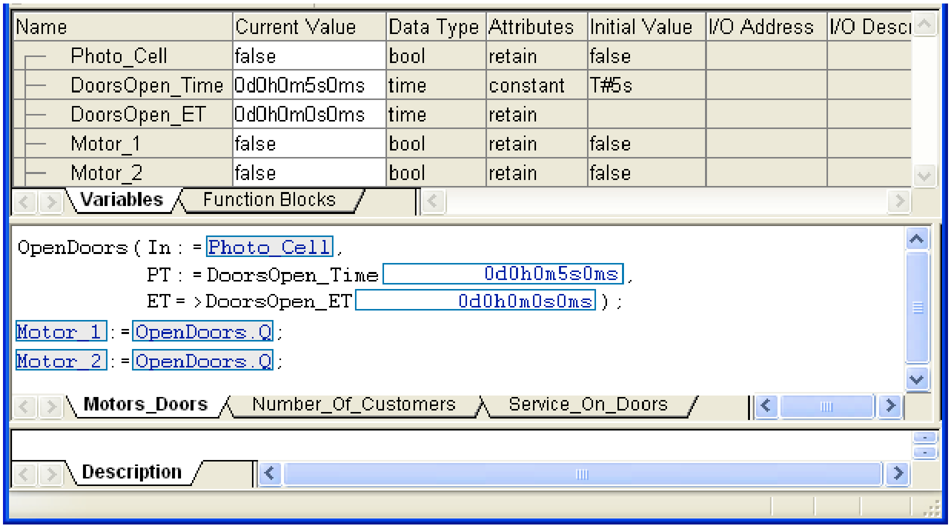
The following functions are available in online mode and test mode:

- Online editors, see Online Editors on page 198.
- Dynamic display of I/O channels and forcing, see Dynamic Display of I/O Channels and Forcing on page 199.
- Scaling analog signals, see Scaling Analog Signals on page 201.
- Unit status and channel status, see Supervising Unit Status on page 201.
- Hardware and task status indications, see Status Indications on page 205.
- Tasks, see Tasks on page 207.
- Interaction windows, see Interaction Windows on page 207.
- Status and error messages, see Status and Error Messages on page 209.
- Reports and analysis, see Search and Navigation in Online and Test Mode on page 210.
- Project documentation, see Project Documentation on page 211.

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, the program editor, the hardware configuration editor and the 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 Objects on page 27) you want to view. You can also select the POU, click the right mouse button and select **View.**



Name	Current Value	Data Type	Attributes	Initial Value	I/O Address	I/O Desci
Photo_Cell	false	bool	retain	false		
DoorsOpen_Time	0d0h0m5s0ms	time	constant	T#5s		
DoorsOpen_ET	0d0h0m0s0ms	time	retain			
Motor_1	false	bool	retain	false		
Motor_2	false	bool	retain	false		

```
OpenDoors ( In : = Photo_Cell,
            PT : = DoorsOpen_Time   0d0h0m5s0ms,
            ET = > DoorsOpen_ET   0d0h0m0s0ms ) ;
Motor_1 : = OpenDoors.Q;
Motor_2 : = OpenDoors.Q;
```

Figure 84. Part of Program 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, in any of the 1131 programming languages.
- The lower description pane displays descriptions of the types and POUs.

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**.

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.

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 201).

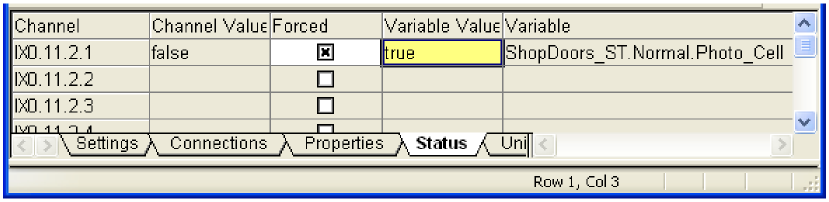
Normally, only channels with variable connections to application programs can be forced. However, if no variable is connected, you have to change the parameter *Copy unconnected channels* under the Settings tab for the current controller to obtain a status update. The I/O channels you can copy are *None*, *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.

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 *IOValue* 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
IX0.11.2.1	false	☒	true	ShopDoors_ST.Normal.Photo_Cell
IX0.11.2.2		□		
IX0.11.2.3		□		
IX0.11.2.4		□		

Figure 85. 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".

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.

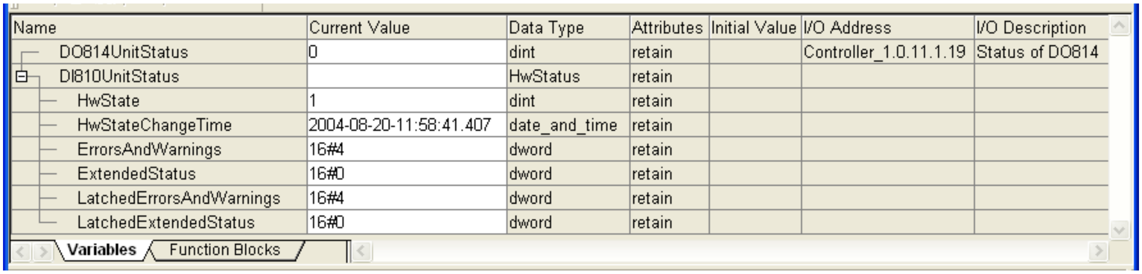
Supervising Unit Status

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 86, the *DO814UnitStatus* variable of *dint* data type is connected to *UnitStatus* of DO814 (unit status is 0=OK!). The *DO810UnitStatus* 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 86. The UnitStatus connection gives access to the status of individual hardware units.

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 6 on page 71 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 87 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.

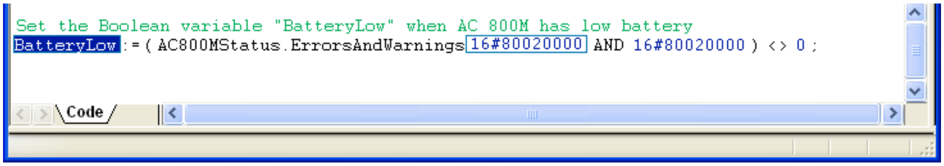
For more information on error codes, see Control Builder online help.

By combining *AC800MStatus.ErrorsAndWarnings* with the bit pattern 80020000[1] 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:

[1] Typed in ST editor in hexadecimal notation as 16#80020000.

```
(*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 87.



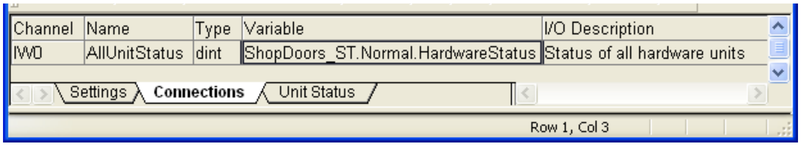
```
Set the Boolean variable "BatteryLow" when AC 800M has low battery
BatteryLow := ( AC800MStatus.ErrorsAndWarnings 16#80020000 AND 16#80020000 ) <> 0 ;
```

Figure 87. 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 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.

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

Figure 88. 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 87).

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 6 on page 71), 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 6 on page 71). You cannot use both methods simultaneously.

ISP and OSP values are not set for variables connected to *All Inputs*/*All Outputs*!

ISP/OSP (**I**nput/**O**utput **S**et as **P**redetermined) 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 6 on page 71).

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 70 and the variable example given in Figure 28 on page 72.

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.

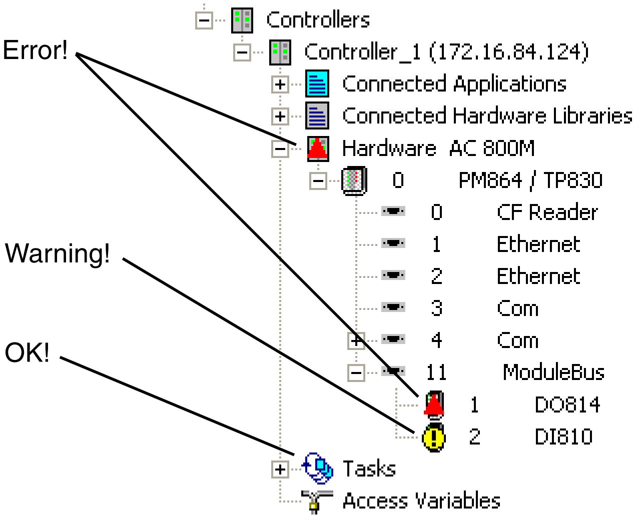


Figure 89. Status indications of hardware and tasks in Project Explorer.

- 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 104 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 Getting Started manual. The warning is described in the Remark field of the Task Properties dialog. See Task Control on page 89 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

ℹ 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. 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

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 Getting Started manual.

If debug mode is used in a running plant, task execution will be stopped.

You can also select Always update output signals last in next execution, or select Always update output signal first in next execution.

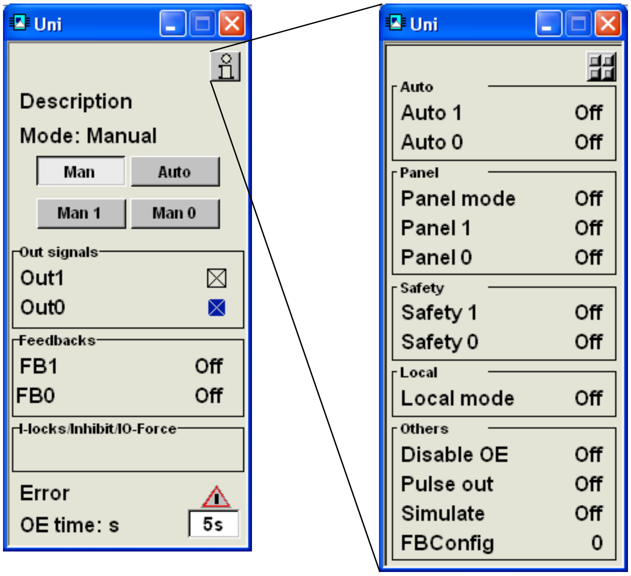
For further basic information about tasks, see Task Control on page 89. For Latency information, see Latency Supervision on page 102. See also Control Builder online help for how to carry out task changes.

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 program editor containing a control module.
- An online program editor containing a function block (compare with item 2 above).
- From interaction window objects in a control module.



Description
Mode: Manual

Out signals
Out1 ☒
Out0 ☒

Feedbacks
FB1 Off
FB0 Off

I-locks/Inhibit/IO-Force

Error
OE time: s 5s

Auto
Auto 1 Off
Auto 0 Off

Panel
Panel mode Off
Panel 1 Off
Panel 0 Off

Safety
Safety 1 Off
Safety 0 Off

Local
Local mode Off

Others
Disable OE Off
Pulse out Off
Simulate Off
FBConfig 0

Figure 90. 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.

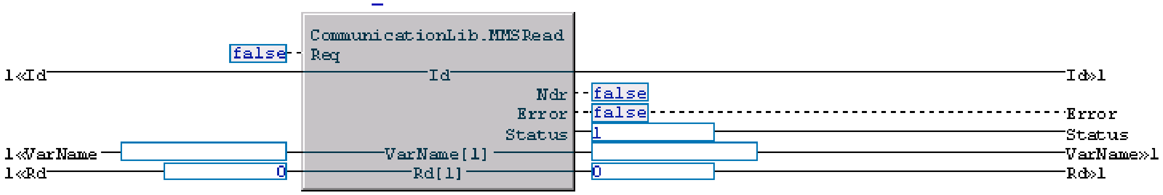


Figure 91. A function block with Status parameter and Error parameter (operation successful=1).

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.

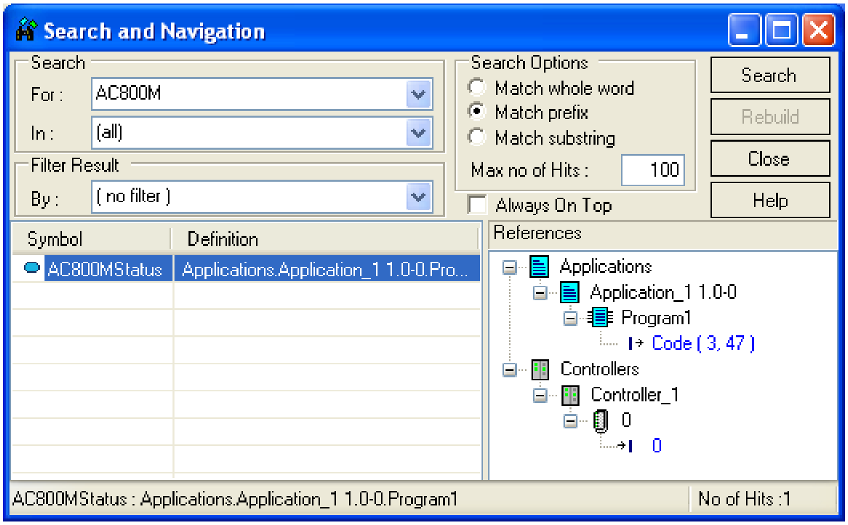
Search and Navigation in Online and Test Mode

The Search and Navigation function makes it possible for the user to search for symbols (see Symbol and Definition on page 111) in the Project Explorer, by using advanced queries. In online mode, for example, an online error can 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 Explorer are shown as well. 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 can also be generated (see Reports on page 119).

There are some differences between offline and online/test mode. For dialog settings and more specific explanations about dialog contents, see description of Search and Navigation on page 107 in offline mode.



For : AC800M
In : (all)
By : (no filter)

○ Match whole word
● Match prefix
○ Match substring

Max no of Hits : 100

□ Always On Top

Symbol	Definition
AC800MStatus	Applications.Application_1 1.0-0.Pro...

References:

- Applications
  - Application_1 1.0-0
    - Program1
      - Code (3, 47)
- Controllers
  - Controller_1
    - 0
      - 0

AC800MStatus : Applications.Application_1 1.0-0.Program1 No of Hits :1

Figure 92. The Search and Navigation function in online mode.

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 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.

  In online mode, it is also possible to navigate from the Search and Navigation function to the corresponding object in the Project Explorer.
- The **Rebuild Search Data** menu selection does not exist in online mode. It is also not possible to click the **Rebuild** button (dimmed) in the Search and Navigation dialog. The Control Builder will, however, update search data before entering online or test mode.

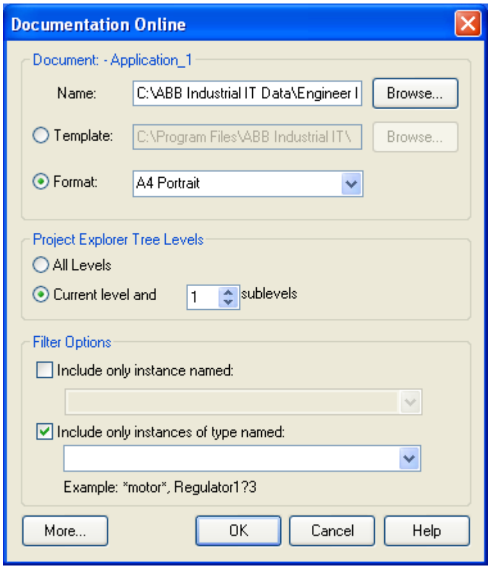
Project Documentation

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 version, the values of variables, parameters, etc. are included. 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.



Documentation Online

Document: - Application_1

Name: C:\ABB Industrial IT Data\Engineer I Browse...

○ Template: C:\Program Files\ABB Industrial IT\ Browse...

● Format: A4 Portrait

Project Explorer Tree Levels

○ All Levels

● Current level and 1 sublevels

Filter Options

☐ Include only instance named:

☑ Include only instances of type named:

Example: *motor*, Regulator1?3

More... OK Cancel Help

Figure 93. The Documentation Online dialog.

3. See Control Builder online help for information about dialog settings and selections.

See Project Documentation on page 129 for information about Project Documentation in offline mode.

Section 5 Maintenance and Trouble-Shooting

This section provides important information for maintenance and trouble-shooting Compact 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:

- Backup and Restore on page 214 gives a short overview of backup and restore.
- Error Handler Configuration on page 216 describes how to configure handling and logging of system alarms and events, using the Error Handler.
- Trouble-Shooting on page 221 lists a number of error symptoms, and suggest actions upon these.
- Error Reports on page 247 describes how to write a complete error report, so that the support engineers get a complete picture of an error situation.

Backup and Restore

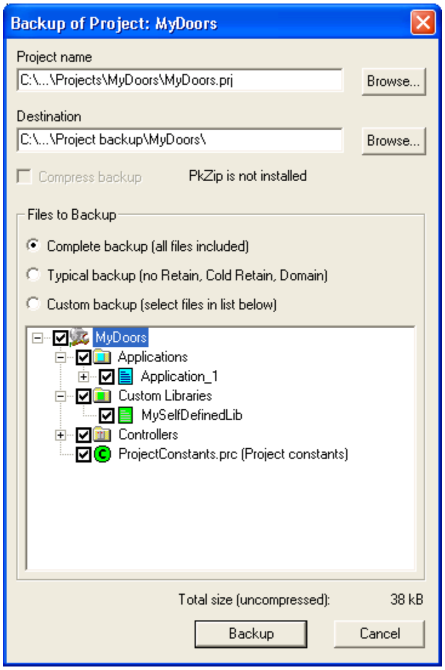
Introduction

This function provides a backup of your project, and enables you to move a project from one Control Builder station to another with the restore function. You can choose to backup a complete project or select parts of the project.

Backup

Compact Control Builder suggest the current project in Project Explorer for backup, or you can browse via a button to another project on your hard disk. Furthermore, Control Builder suggests a destination folder, named Project Backup which will be created next to the Project folder.

To Backup a project, select (in Project Explorer) **Tools>Maintenance>Project Backup**. A Project Backup dialog window will open, (Figure 94).



Backup of Project: MyDoors

Project name: C:\...\Projects\MyDoors\MyDoors.prj

Destination: C:\...\Project backup\MyDoors\

□ Compress backup PkZip is not installed

Files to Backup

● Complete backup (all files included)
○ Typical backup (no Retain, Cold Retain, Domain)
○ Custom backup (select files in list below)

- ☑ MyDoors
  - ☑ Applications
    - ☑ Application_1
  - ☑ Custom Libraries
    - ☑ MySelfDefinedLib
  - ☑ Controllers
  - ☑ ProjectConstants.prc (Project constants)

Total size (uncompressed): 38 kB

Figure 94. Backup menu in Project Explorer.

Complete Backup

This option includes all files that are needed to restore the project on another computer.

Typical Backup

A Typical backup of a project includes all source code files that are needed to restore the project on another computer. However, Retain, Cold Retain or Domain files will not be included with this option.

Custom Backup

Advanced users may want to choose explicitly which files to back up. This option provides a list of all files included in the project.

Restore

The restore function is used to install a backup, for example, after a disk crash, or when moving a project to another system. You may decide to make a Complete, Typical or Custom restore.

When you are restoring a complete project, all the project-related files are copied from the backup folder to the project folder.

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 OPC Server for AC 800M manual 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.

Error Handler Configuration

The Error Handler is used to configure controller behavior on system alarms and events of different severities, and how different errors are logged.

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.Error Handler settings are slightly different for High Integrity and non-High Integrity controllers:

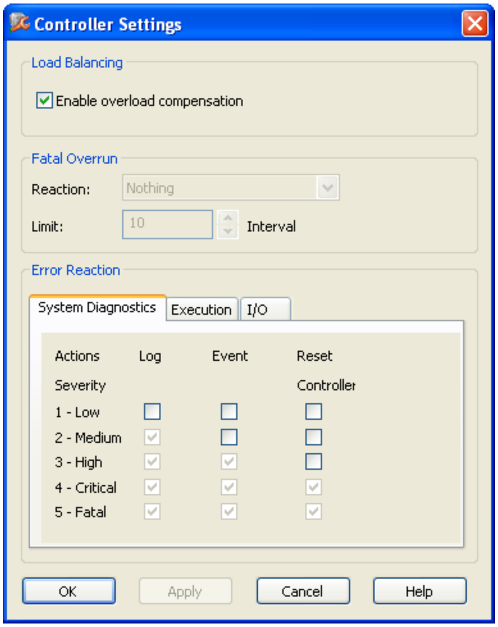
- Error Handler Settings in Controllers on page 217 describes how to configure the Error Handler in a controller.

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.

Error Handler Settings in Controllers

Figure 95 shows the Controller Settings dialog for a AC 800M controller. It is displayed by right-clicking the controller in Project Explorer and selecting **Properties > Controller Settings**.



☑ Enable overload compensation

Reaction: Nothing

Limit: 10 Interval

Actions Severity	Log	Event	Reset Controller
1 - Low	□	□	□
2 - Medium	☑	□	□
3 - High	☑	☑	□
4 - Critical	☑	☑	☑
5 - Fatal	☑	☑	☑

Figure 95. Controller Settings dialog for an AC 800M controller.

If load balancing is enabled, overrun and latency supervision is automatically disabled, see Overrun and Latency on page 100.

The default setting for a 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 95).

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 (Nothing is default). The default setting for the limit is 10 overruns.

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, Reset Controller for the corresponding severity). Note that severity Fatal and Critical will always lead to a controller reset.

If settings are inconsistent, you will receive a warning when trying to save the new settings.

For a controller, the Error Reaction part lets the user set the following, see Table 13.

Table 13. Error Reaction. This part of the dialog is used to set controller actions at system alarms of different severity.

Severity	Log	Event	Reset Controller
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 alarms of different severities are received by the Error Handler in a controller.

There are three tabs in the Error Reaction part of the dialog:

- The System Diagnostics tab contains settings for system alarms generated by the System Diagnostics module, for example, CEX module errors, protocol handler errors.
- The Execution tab contains settings for system alarms generated during execution of IEC-61131 code, for example, latency errors, CRC check failures.
- The I/O tab contains settings for I/O module errors.

The following definitions have been used for the severity of system errors when designing error handling for different modules:

- **1 Low**
  Minor, of diagnostic or informative sort. Does not affect system integrity or the functionality of the reporting module.
- **2 Medium**
  An error, such as I/O channel failure, communication failed, or similar, has occurred. Does not affect system integrity, but affects functionality in the reporting module.
- **3 High**
  Severe error, but not critical, for example I/O module failure. May affect system integrity. Functionality in the reporting module is affected. Redundancy may maintain the system integrity
- **4 Critical**
  A severe error has occurred, for example, a task has stalled, ModuleBus stalled, I/O cluster down. Will affect system integrity, since the reporting module has failed. Redundancy may maintain the safety of the system.
- **5 Fatal**
  Systematic software errors have been found. The whole reporting subsystem has failed. Redundancy will not maintain the system integrity. This severity is only used when there is no possibility to safely continue using a backup PM.

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 231) 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 below table.

Table 14. 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:`	Error detected by	`ErrorHandler PM:` =Processor Module
`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)

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 247 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.

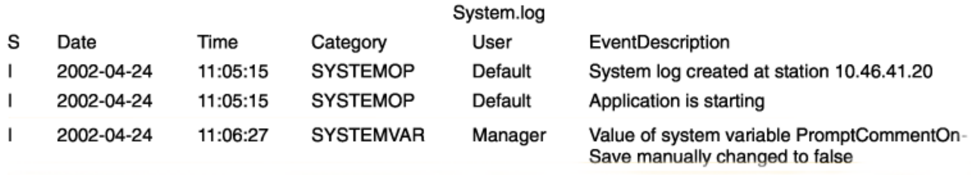
- What is the problem?
- Is it a known problem? Check the available information (for example, Release Notes and Product Bulletins) and discuss it with colleagues.
- Has the system worked previously (with the same hardware)? If so, the problem may have occurred due to poor installation or due to setup 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 234) are very useful for troubleshooting and contain crucial information for analyzing malfunctions.

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 96 shows an example of the system log.



```
                                        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 96. An example of the system log

The path and file name of the System log are given in Table 15.

Table 15. The System log file path.

Denomination	Path/Note
Control Builder System Log	**Path / File name** C:\ABB Industrial IT Data\Engineer IT Data[1]\ Compact Control Builder \LogFiles\System.log **Note** Only one version of this file exists.

[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, automatically creates a *session log* file on the hard disk. 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. 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~~

The paths and file names of the session logs are given in Table 16.

Table 16. Session log file paths.

Denomination	Path/Note
Control Builder session log	**Path / File name** C:\ABB Industrial IT Data\Engineer IT Data[1]\ Compact Control Builder\LogFiles\Session.log **Note** Session log files stored from the last 9 sessions: Session.log Session.log_bak1, Session.log_bak2, *Session.log_bakn*.... Session.log_bak9
OPC Server session log	**Path / File name** C:\ABB Industrial IT Data\Control IT Data[1]\OPC Server for AC 800M\ LogFiles\Session.log **Note** Session log files are stored from the last 9 sessions: Session.log Session.log_bak1, Session.log_bak2, *Session.log_bakn*.... Session.log_bak9

Table 16. Session log file paths. (Continued)

Denomination	Path/Note
MMS Server session log	**Path / File name** C:\ABB Industrial IT Data\Control IT Data[1]\ MMS Server for AC 800M\ Session.log **Note** Session log files are stored from the last 9 sessions: Session.log Session.log_bak1, Session.log_bak2, *Session.log_bakn....* Session.log_bak9
SoftController session log	**Path / File name** C:\ABB Industrial IT Data\Control IT Data[1]\ SoftController \ Session.log **Note** Session log files are stored from the last 9 sessions: Session.log Session.log_bak1, Session.log_bak2, *Session.log_bakn....* Session.log_bak9

[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 forth 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 17.

Table 17. The Control Builder start log file path.

Denomination	Path/Note
Control Builder Start log	**Path / File name** C:\ABB Industrial IT Data\Engineer IT Data[1]\ Compact Control Builder\LogFiles\startlog.txt **Note** The nine latest Start log files are saved: startlog.txt startlog.txt_bak1,startlog.txt_bak2, *startlog.txt_bakn....* startlog.txt_bak9

[1] The default working directory is shown.

Field bus parameter log files

During compilation and simulation, CI851, CI854 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 18.

Table 18. The Field bus parameter log files path.

Denomination	Path/Note
CI851 parameter log file	**Path / File name** C:\ABB Industrial IT Data\Engineer IT Data[1]\ Compact Control Builder\LogFiles\Profibus_DP_Calculation.txt
CI854 parameter log file	**Path / File name** C:\ABB Industrial IT Data\Engineer IT Data[1]\ Compact Control Builder\LogFiles\Profibus_DPV1_Calculation.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 Compact Control Builder> 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 System information report file are shown in Table 19.

Table 19. The Control Builder system information report file path.

Denomination	Path/Note
Control Builder System information report	**Path / File name** C:\ABB Industrial IT Data\Engineer IT Data[1]\ Compact Control Builder\LogFiles\ SystemInformation.txt

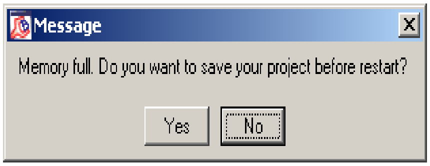
[1] The default working directory is shown.

Heap Statistics Log

There are *heap statistics* log files for Control Builder, SoftController, and OPC Server for AC 800M. Every time a message “memory full” occurs (see Figure 97) 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.



Memory full. Do you want to save your project before restart?

Figure 97. The “memory full” message.

The paths and file names of the *heap statistics* log files are given in Table 20

Table 20. The heap statistics log file paths.

Denomination	Path/Note
Control Builder heap statistics log	**Path / File name** C:\ABB Industrial IT Data\Engineer IT Data(1)\ Compact Control Builder\LogFiles\heapstat.dat **Note** The file is intended to be stored and be included in an error report.

[1] A product, for example, a Control Builder, an OPC server, or a 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.

Table 20. The heap statistics log file paths.

Denomination	Path/Note
OPC Server for AC 800M heap statistics log	**Path / File name** C:\ABB Industrial IT Data\Control IT Data[1]\OPC Server for AC 800M \LogFiles\heapstat.dat **Note** The file is intended to be stored and included in an error report.
SoftController heap statistics log	**Path / File name** C:\ABB Industrial IT Data\Control IT Data[1]\SoftController \heapstat.dat **Note** The file is intended to be stored and included in an error report.

[1] The default working directory is shown.

Controller System Log

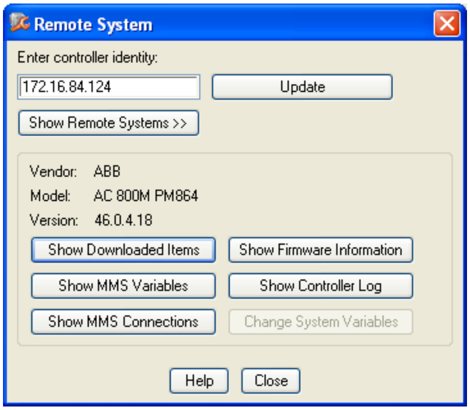
Controllers have a circular log buffer that can hold a certain amount of information, normally all information that has been generated during the last 5 to 8 start-ups.

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 98.



Remote System

Enter controller identity: 172.16.84.124

Vendor: ABB
Model: AC 800M PM864
Version: 46.0.4.18

Figure 98. The Remote System dialog box.

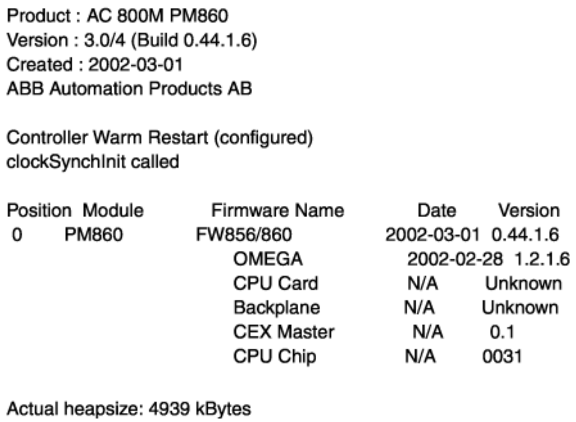
Enter the controller identity (the IP address) and click on the **Show Controller Log** button to show the Controller System Log.

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 (see the path in Table 21).

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 99 below, is an excerpt of the controller system log.



```
Product : AC 800M PM860
Version : 3.0/4 (Build 0.44.1.6)
Created : 2002-03-01
ABB Automation Products AB

Controller Warm Restart (configured)
clockSynchInit called

Position  Module          Firmware Name          Date       Version
 0        PM860          FW856/860          2002-03-01  0.44.1.6
                             OMEGA              2002-02-28  1.2.1.6
                             CPU Card             N/A        Unknown
                             Backplane            N/A        Unknown
                             CEX Master            N/A        0.1
                             CPU Chip             N/A        0031

Actual heapsize: 4939 kBytes
```

Figure 99. 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 21

Table 21. The controller system log and controller integrity log file paths .

Denomination	Path/Note
Controller System log	**Path / File name** **All controllers:** C:\ABB Industrial IT Data\Engineer IT Data[1]\Compact Control Builder \LogFiles\Controller_a_b_c_d.log (BackupCPU_a_b_c_d.log) **Note** a_b_c_d is the IP address of the controller. See Controller System Log on page 231. The nine latest Controller System logs are saved: Controller_a_b_c_d.log Controller_a_b_c_d.log_bak1, Controller_a_b_c_d.log_bak2, etc Controller_a_b_c_d.log_bak9

[1] The default working directory is shown.

Crash Dumps for Analysis and Fault-Localization

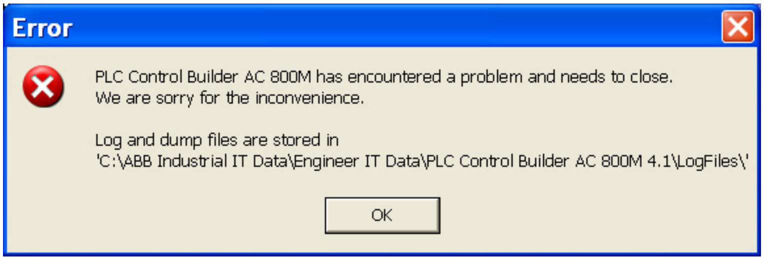
If a crash occurs (in Control Builder, OPC Server, SoftController, MMS Server for AC 800M, two new files are generated at the same location as the session log files. The first one is a dump file and the second is a rewritten session log file. These two files contain crucial information that should be delivered to the support personnel

If a 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:

ControlBuilderStd 2006-05-19 16.20.29.184.dmp

ControlBuilderStd 2006-05-19 16.20.29.184 Session.LOG

After these two files have been generated, an error message window will show up on the screen.



Error

PLC Control Builder AC 800M has encountered a problem and needs to close.
We are sorry for the inconvenience.

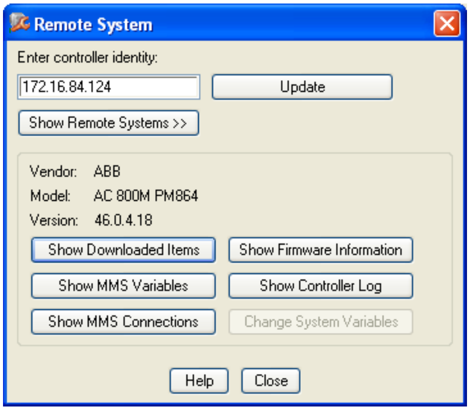
Log and dump files are stored in
'C:\ABB Industrial IT Data\Engineer IT Data\PLC Control Builder AC 800M 4.1\LogFiles\'

Figure 100. Error message, showing where to find the Crash Dump and Session Dump.

Remote Systems Information

A connected remote control system[1] 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 101.



Remote System

Enter controller identity: 172.16.84.124

Vendor: ABB
Model: AC 800M PM864
Version: 46.0.4.18

Figure 101. 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 22 below. Click on a button in the dialog to retrieve information.

Table 22. 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 Getting Started manual.
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 231.
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 22. The available remote system dialog functions. (Continued)

Menu Item	Function
Change System Variables	Shows a dialog box where the system variable values in a controller can be changed.

For further information, refer to Control Builder online help. Use the Help button in the Remote System dialog, see Figure 101.

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 23.

For further information, refer to Control Builder online help.

Table 23. The menu items of the Analysis tool.

Menu Item	Function
Disable Double-buffering	*Not useful for troubleshooting*
Disable Information Zoom	*Not useful for troubleshooting*
Disable Clipping	*Not useful for troubleshooting*
Image Selector Info in Online Mode	*Not useful for troubleshooting*
Image Selector Information	*Not useful for troubleshooting*
Show control modules in Online Mode	*Not useful for troubleshooting*

Table 23. The menu items of the Analysis tool. (Continued)

Menu Item	Function
Write Variable Memory	Used for counting modules and instances as described in Exceeding the Maximum Number of Instances on page 238.
Write Exported Variables	*Not useful for troubleshooting*
Write Variables in View	*Not useful for troubleshooting*
Write Heap Statistics	Creates the Control Builder Heap Statistics Log, described in Heap Statistics Log on page 230.
Application Information	Shows information on the application selected in Control Builder. The application name, creation date and checksum are shown. The checksum may be useful, for instance, to check if two applications loaded into two different engineering stations have the same status.
Start log	Shows the Control Builder Start log, described in Control Builder Start Log on page 227.
System log	Shows the Control Builder System log, described in System Log File on page 222.

Exceeding the Maximum Number of Instances

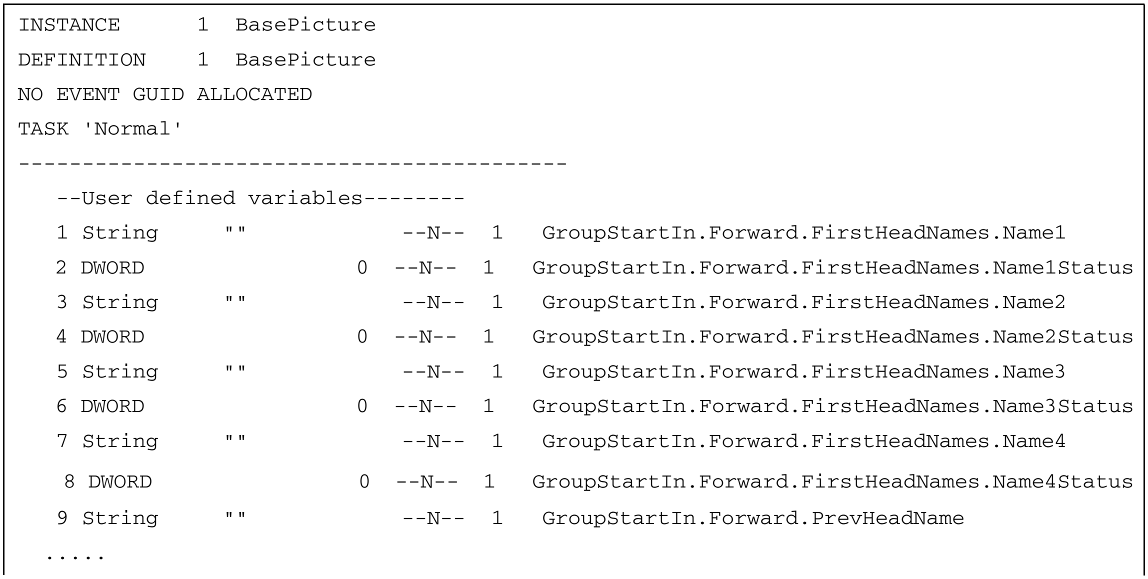
To be able to count modules and instances the application must be compiled. To find out the number of instances:

1. In Project Explorer, go to Test mode or Online mode.
2. Select **Tools > Maintenance > Analysis > WriteVariableMemory**. The Variable Memory dialog is displayed.
3. Select the application and click **OK**. A window will show variable information. This information will also be stored in a file called *varmem.txt*, which is stored

together with other log files at *C:\ABB Industrial IT Data\Engineer IT Data\Compact Control Builder x.y\LogFiles\*.

If the maximum number of instances has been exceeded, it is not possible to compile the application. For instructions on what to do in this case, see page 240.

4. Open the text file, see Figure 102, and browse to the end.



```
INSTANCE      1  BasePicture
DEFINITION    1  BasePicture
NO EVENT GUID ALLOCATED
TASK 'Normal'
-------------------------------------------
   --User defined variables--------
   1 String     ""            --N--  1   GroupStartIn.Forward.FirstHeadNames.Name1
   2 DWORD                  0  --N--  1   GroupStartIn.Forward.FirstHeadNames.Name1Status
   3 String     ""            --N--  1   GroupStartIn.Forward.FirstHeadNames.Name2
   4 DWORD                  0  --N--  1   GroupStartIn.Forward.FirstHeadNames.Name2Status
   5 String     ""            --N--  1   GroupStartIn.Forward.FirstHeadNames.Name3
   6 DWORD                  0  --N--  1   GroupStartIn.Forward.FirstHeadNames.Name3Status
   7 String     ""            --N--  1   GroupStartIn.Forward.FirstHeadNames.Name4
    8 DWORD                 0  --N--  1   GroupStartIn.Forward.FirstHeadNames.Name4Status
   9 String     ""            --N--  1   GroupStartIn.Forward.PrevHeadName
  .....
```

Figure 102. Contents of varmem.txt file (this file can be very long, only part of the information is shown here in the figure).

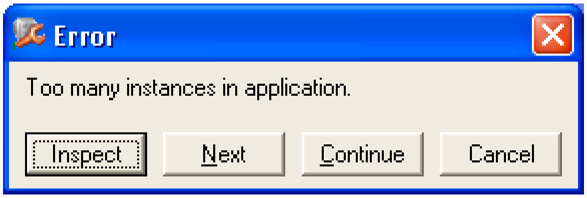
5. Scroll to the end of the file. The number of instances is shown at the bottom row of the file. Find the **last** row that starts with the word “INSTANCE”–the number of instances will follow immediately after this word. This row might look as in the below example:

```
INSTANCE  170  Application_1.MotorUniM2.GroupStartIcon.UniIcon1
```

In the above example, the number of instances is 170.

This is a useful check to perform every now and then. If your application approaches the maximum number of instances (65535), you should consider re-structuring your code.

The maximum number of instances in an application is 65536. When this number is exceeded, the following dialog is shown.



Too many instances in application.

Figure 103. Error message shown at download, when an application has too many instances.

If you have already exceeded the limit, you will not be ably to compile the application. Perform the following steps to find out how many instances you have:

1. Repeat the following steps until compilation is successful:
   a. Remove an item from your system, for example, a program.
   b. Compile the application.
   c. If compilation is still not successful, remove an additional item.
2. Once compilation is successful, check the number of instances, as described in the previous instruction. Write this number down, call it A.
3. Experiment with adding removed items and try to find two items that can be added without compilation errors. For each successful compilation, check the number of instances with that particular item added, and write it down. Call these two numbers B and C.
4. You can now calculate the total number of instances as A+(B-A)+(C-A).
5. You will now have to re-structure your application in such a way that the problem is eliminated. This could be done by splitting the application into two different applications, or by decreasing the number of instances.

System Diagnostics Function

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,
- 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 *Program* folders of the Project Explorer tree, see Figure 104.

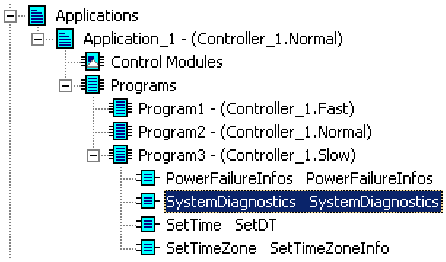

Applications
Application_1 - (Controller_1.Normal)
Control Modules
Programs
Program1 - (Controller_1.Fast)
Program2 - (Controller_1.Normal)
Program3 - (Controller_1.Slow)
PowerFailureInfos PowerFailureInfos
SystemDiagnostics SystemDiagnostics
SetTime SetDT
SetTimeZone SetTimeZoneInfo

Figure 104. 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 24. 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 *TotalSystemLoadPerCent*.
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 25. The Advanced interaction window.

Function	**Description**
Memory size	The allocated heap size, see Figure 105.
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.

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”.

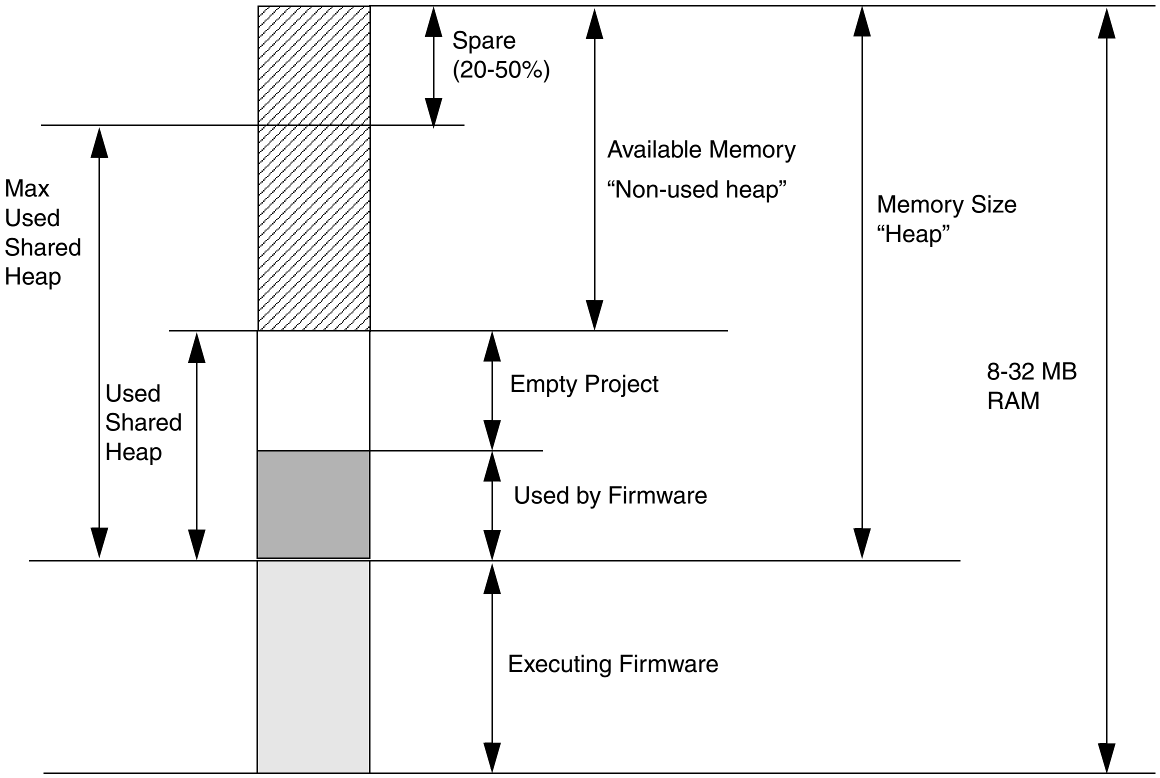


Figure 105. Memory organization

Trouble-Shooting Error Symptoms

Below are some examples of error symptoms and suggested measures.

Table 26. Examples of error symptoms and suggested measures.

Error Symptom	Measure
Control Builder fails, the message in Figure 100 is shown.	1.Click **OK**. 2.Copy the two crash dump files (see Crash Dumps for Analysis and Fault-Localization on page 234), 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 Compact Control Builder>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 *Memory Full* message appears. The Heap Statistics log (Control Builder, SoftController, or OPC Server for AC 800M) states that the heap is full.	Increase the heap size in Control Builder, SoftController, or OPC Server for AC 800M, see Heap Statistics Log on page 230. Open **Help > About** and check the amount of free memory. Free memory should not be lower than 30%.

Table 26. Examples of error symptoms and suggested measures. (Continued)

Error Symptom	Measure
A *Too many instances in application* message appears. The maximum number of about 65535 instances has been reached.	1.Try to reduce your application, see Exceeding the Maximum Number of Instances on page 238.
The MMS Server, OPC Server, or SoftController fails. An error message like the one in Figure 100is shown.	1.Click **OK**. 2.Locate the two crash dump files (see Crash Dumps for Analysis and Fault-Localization on page 234). 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.
The controller fails. The red F LED is lit, and the green R LED is off.	1.Press the **Init** push-button on the controller until the **R**un 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. 2.Fetch the Controller System log and save it, see Remote Systems Information on page 235. 3.Study the log, and find the marked reason for the stop (normally, at the end of the log). 4.If an OPC Server for AC 800M is involved in communication, check the OPC Server function. 5.Make an error report and include the saved log files. 6.Reload the application. 7.If possible, try to reproduce the problem. If the problem is reproducible, backup the project. 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).

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 Log Files on page 222. 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	
Irec	trec1	
Irec1	trec1	
Irec2	trec1	
Irec3	trec1	
Status	dint	
FirstScan	bool	TRUE

Create and initialize an array with 20 array elements of the type *trec1*.

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

SearchRecComponent is a boolean function which searches for a specific part in a record component. The corresponding components in *Exrecord* are scanned to find a part in the component which matches the *SearchComponent*.

```
Variable = SearchRecComponent(ExRecord, ComponentIndex,
SearchCount, SearchRecord, SearchComponent, FoundRecord,
Status)
```

Table 27.

Parameter	Data type	Description
ExRecord	AnyType	in var
ComponentIndex	integer	in/out
SearchCount	integer	in
SearchRecord	AnyType	in var
SearchComponent	AnyType	in var
FoundRecord	AnyType	out
Status	integer	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 *ExRecord*. 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.

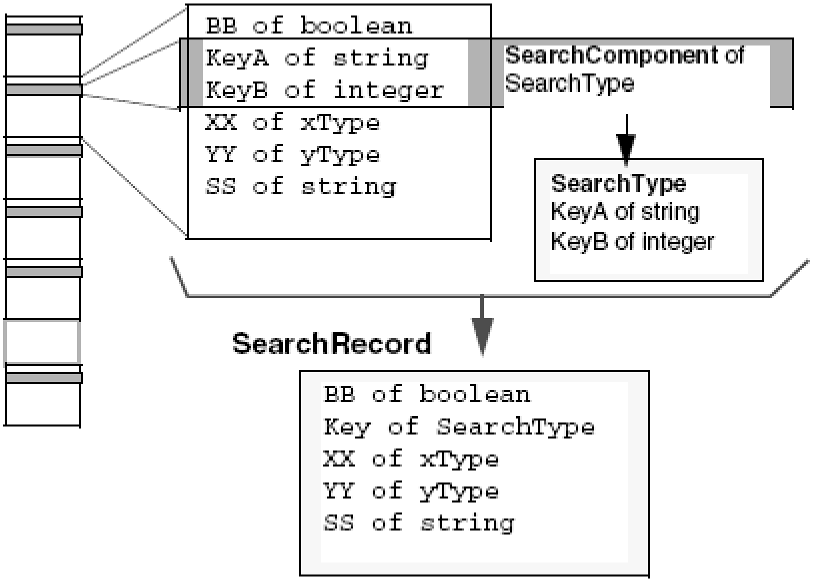


Figure 106. An example of the SearchComponent and a SearchRecord.

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 *FoundRecord* and the function returns true.

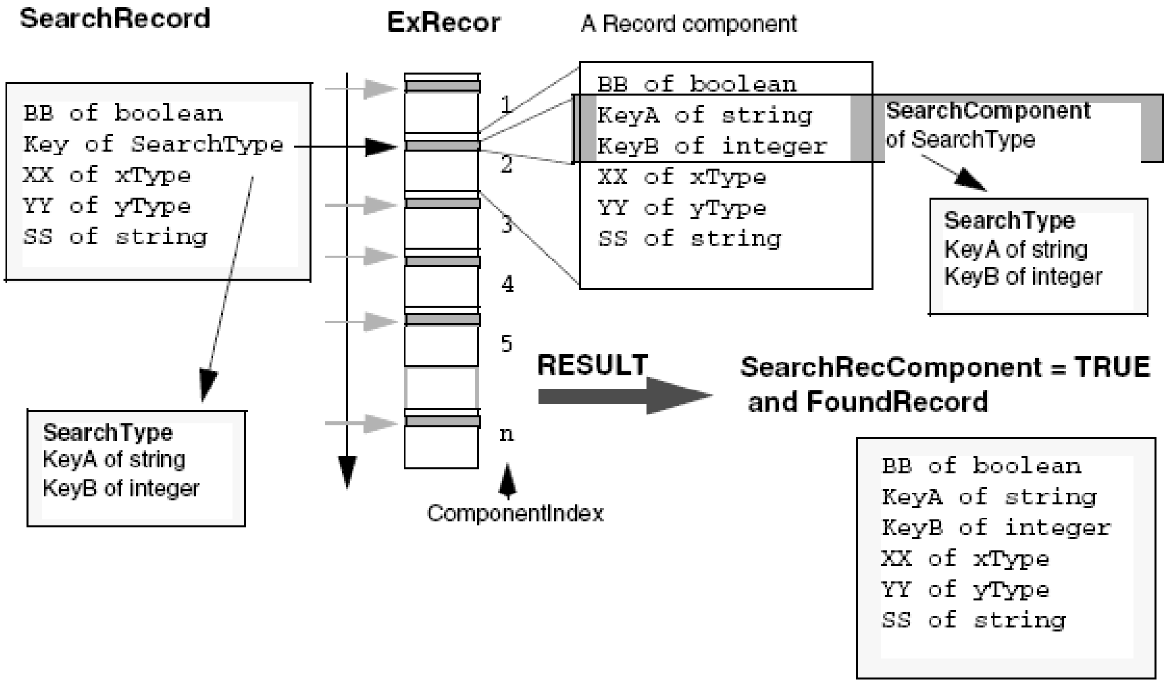


Figure 107. The working principal of the SearchRecComponent.

The search starts in the index *ComponentIndex + 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 *FoundRecord* and the index is returned in *ComponentIndex*.

Note that *ComponentIndex* 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 *ExRecord* will NOT be copied: *QueueObject* 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*FoundRecord*.
  - 2: *SearchComponent* was not a subset of *SearchRecord*
- (- 6 ErrSizeMismatch)
  - 1: *SearchRecord* was not of the same size as the *FoundRecord*.
  - 2: *SearchComponent* size is zero.
- (-30 ErrInvalidPar)
  - 1: *ComponentIndex* was less than 0 or greater than the number of the *ExRecord* 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(ArrayName, Index, ArrElement, 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 *ArrElement* into the record at position Index in the array *ArrayName*. 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 28.

Parameter	**Data type**	**Description**
ArrayName	ArayObject	in var
Index	integer	in
ArrElement	AnyType	in var
Status	integer	out

SearchArray

SearchArray(Arrayname, SearchIndex, SearchCount, SearchElement, SearchComponent, FoundElement, Status)

This boolean function searches the array ArrayName 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 then 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.

```
Variable = SearchArray(Arrayname, SearchIndex, SearchCount,
SearchElement, SearchComponent, FoundElement, Status)
```

Table 29.

Parameter	Data type	Description
Arrayname	ArayObject	in var
SearchIndex	integer	out
SearchCount	integer	in
SearchElement	AnyType	in var
SearchComponent	AnyType	in var
FoundElement	AnyType	out
Status	integer	out

Example

Table 30. Data Type Definitions

Name	Data Type
trec1	RECORD
b	Boolean
i	Integer
s	String
tSearchRec	RECORD
b	Boolean
SSR	tSearchSubRec
tSearchSubRec	RECORD
i	Integer
s	String

Table 31. Variables

Name	Data type	Initial value
Array	ArrayObject	
HitBoolean	Boolean	
HitRec	trec1	
Lrec	trec1	
lrec1	trec1	
lrec2	trec1	
lrec3	trec1	
Status	Integer	
SearchRec	tSearchRec	
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:

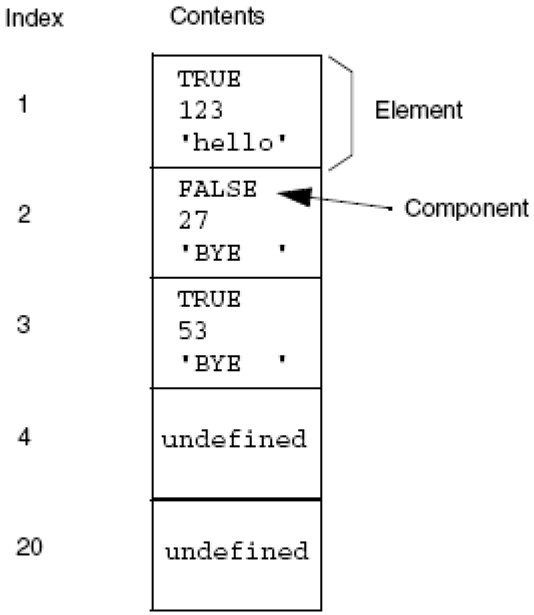


Figure 108. An example of an Array.

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.

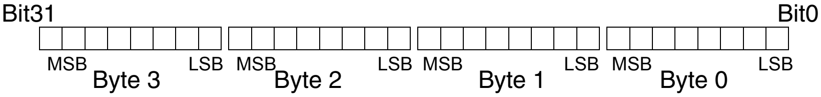


Figure 109. Integers are stored as four bytes in memory.

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 32. 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).

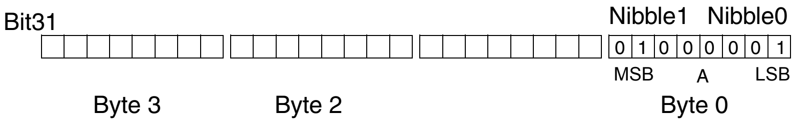


Figure 110. The ASCII code for “A” stored in an integer (packing = 1 character per integer)

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.

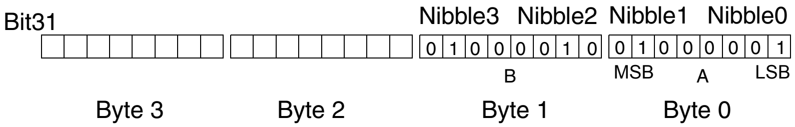


Figure 111. The ASCII codes for “AB” stored in an integer (packing = 2 characters per integer)

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.

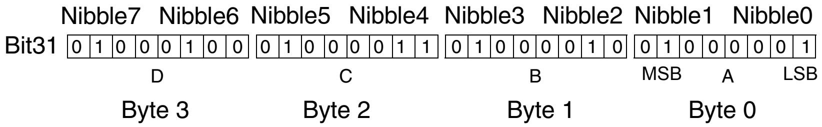


Figure 112. The ASCII codes for “ABCD” stored in an integer (packing = 4 characters per integer)

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.

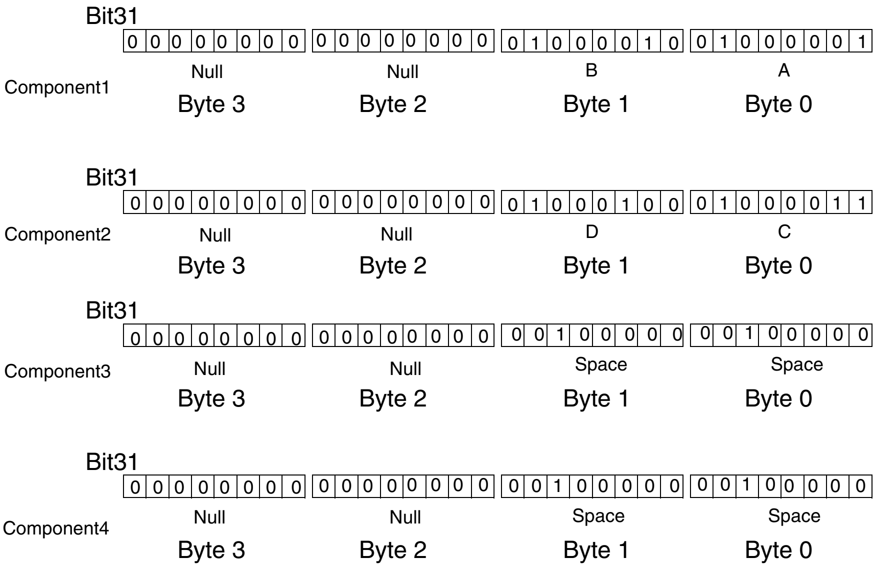
If the input string is “ABCD”, then the components will have the values 65, 66, 67 and 68, respectively. The literal value of 1 for the *NoOfCharsPerDint* determines that there is to be one character code in each component.

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:



	Bit31			
Component1	0 0 0 0 0 0 0 0 Null Byte 3	0 0 0 0 0 0 0 0 Null Byte 2	0 1 0 0 0 0 1 0 B Byte 1	0 1 0 0 0 0 0 1 A Byte 0
Component2	0 0 0 0 0 0 0 0 Null Byte 3	0 0 0 0 0 0 0 0 Null Byte 2	0 1 0 0 0 1 0 0 D Byte 1	0 1 0 0 0 0 1 1 C Byte 0
Component3	0 0 0 0 0 0 0 0 Null Byte 3	0 0 0 0 0 0 0 0 Null Byte 2	0 0 1 0 0 0 0 0 Space Byte 1	0 0 1 0 0 0 0 0 Space Byte 0
Component4	0 0 0 0 0 0 0 0 Null Byte 3	0 0 0 0 0 0 0 0 Null Byte 2	0 0 1 0 0 0 0 0 Space Byte 1	0 0 1 0 0 0 0 0 Space Byte 0

Figure 113. 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

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 = _SWAlarmEvent

System 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 cowered 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- [1]for 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** s**um alarm**

SrcNameSuffix = _SWFirmware
Condition name = ErrorHandler;
Message = "(1001) ErrorHandler sum alarm created";
SeverityLevel = Medium;

System Alarm **Data transfer failed during FW-upgrade of Alarm&Even**t

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 = Medium;

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 **Configuration load is started from compact flash**

SrcNameSuffix = _SWFirmware;
Message = "(1043) Configuration load 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 **Configuration load 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) Configuration load 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;

SrcNameSuffix = _SW1131Task

System 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;

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 33)
{3} = The Error status is printed in hexadecimal format.
Severity Level = High;

Table 33. 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 34)
{3} = The Error status is printed in hexadecimal format.
Severity Level = High;

Table 34. 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 35)
{3} = The state when the error was detected.
Severity Level = High;

Table 35. 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
RCUDrvErro	Failed when calling the RCU driver

Table 35. 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 (seeTable 36)
Severity Level = High;

Table 36. 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 37)
{3} = Error Status. in hexadecimal format.
Severity Level = High;

Table 37. 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 38)
{3} = Error Status. in hexadecimal format.
Severity Level = High;

Table 38. 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 39)
{3} = The state when the error was detected.
Severity Level = High;

Table 39. 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 39. 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";

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 Backup

System 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 driver

System 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 Integrety 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) Iirq 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 etiher 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 etiher 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 **Comparision 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 40. 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 40. Parameters for Hardware Generated System Alarms and Events

Parameters	Descriptions
Message	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 CB and or in the System status viewer in Operate IT. The error code is stored in two 32 bit words first word is *ErrorsAndWarnings* and the second is *ExtendedStatus*. In each hardware generated system alarm or event message, *ErrorsAndWarnings* and *ExtendedStatus* 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 *ErrorsAndWarnings* are displayed as "Device spcific bit xx" in the message e.g. "Device spcific bit 31" . The same goes for ExtendedStatus. Unit specific bits from *ExtendedStatus* are displayed as "Extended status bit xx" in the message e.g. "Extended status bit 0" . Every unit have a table in this document there the bits are explained. **Example** "Controller_1 (0000) I/O configuration error, Device specific bit 31, Extended status bit 0" If the Unit in this example is a PM865, "Device specific bit 31"= "Battery low" and "Extended status bit 0" = "Backup CPU stopped" In the controller sessionlog *ErrorsAndWarnings* and *ExtendedStatus* are presented as HEX format. **Example**: "E 2004-03-08 10:25:06.677 On Unit= 2 HWError Controller_1 Errorcode=16#80004000 16#00000001 (0000) See HW-tree
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".

Table 40. Parameters for Hardware Generated System Alarms and Events

Parameters	Descriptions
CondName	All hardware generated system alarms have "HWError" 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 41 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 41. General status bit

Bit	StatusType	Indication	Generation	Severity	Description
0	ErrorsAndWarnings	Error	Alarm	High	Connection down
1	ErrorsAndWarnings	Error	Alarm	Medium	I/O error
2	ErrorsAndWarnings	Error	Alarm	High	Module missing
3	ErrorsAndWarnings	Error	Alarm	High	Wrong module type
4	ErrorsAndWarnings	Warning	Alarm	Medium	Channel error
5	ErrorsAndWarnings	Warning	Event	Low	I/O warning
6	ErrorsAndWarnings	Warning	Alarm	Low	Underflow
7	ErrorsAndWarnings	Warning	Alarm	Low	Overflow
8	ErrorsAndWarnings	Warning	Event	Low	Forced
9	ErrorsAndWarnings	Error	Alarm	High	Watchdog timeout
10	ErrorsAndWarnings	Error	Alarm	High	Device failure
11	ErrorsAndWarnings	Error	Alarm	High	Device not found
12	ErrorsAndWarnings	Error	Alarm	High	Wrong device type

Table 41. General status bit (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
13	ErrorsAndWarnings	Error	Alarm	Medium	I/O connection error
14	ErrorsAndWarnings	Error	Alarm	Medium	I/O configuration error
15	ErrorsAndWarnings	Error	Alarm	High	Hardware configuration error
18	ErrorsAndWarnings	Warning	Event	Low	Warning on primary unit
19	ErrorsAndWarnings	Warning	Event	Low	Warning on backup unit
20	ErrorsAndWarnings	Warning	Alarm	Medium	Error on backup unit
21	*Reserved*				
22	*Reserved*				
23	ExtendedStatus	Error	Alarm	High	Version of the Running Primary is incompatible
24	ExtendedStatus	Warning	Alarm	Medium	Version of the Running Backup is incompatible
25	ExtendedStatus	Warning	Alarm	Medium	Version of the Running Primary is not preferred
26	ExtendedStatus	Warning	Alarm	Medium	Version of the Running Backup is not preferred
27	ExtendedStatus	Warning	Alarm	Low	Watchdog timeout on backup
28	ExtendedStatus	Warning	Alarm	Low	Backup device failure

Table 41. General status bit (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
29	ExtendedStatus	Warning	Event	Low	Switchover in progress
30	ExtendedStatus	-	-	-	Redundant mode enabled
31	ExtendedStatus	-	-	-	Unit B acts primary

Unit Specific Alarms and Events

This subsection lists the unit specific alarms and events, sorted in the following categories of units:

- Controller units and communication interfaces (see Controllers Units and Communication Interfaces on page 315).
- Adapters (see Adapters on page 329).
- S800 I/O (see S800 I/O on page 336).
- S900 I/O (see S900 I/O on page 382).
- S100 I/O (see S100 I/O on page 418).
- INSUM devices (see INSUM Devices on page 420).
- FF devices (see FF Devices on page 422).
- MB300 nodes (see MB300 Nodes on page 423).
- ABB Standard drive (see ABB Standard Drive on page 423).

Controllers Units and Communication Interfaces

Table 42. PM851 / TP830

Bit	StatusType	Indication	Generation	Severity	Description
29	ErrorsAndWarnings	Warning	Alarm	Medium	RPA
30	ErrorsAndWarnings	Warning	Alarm	Medium	RPB
31	ErrorsAndWarnings	Warning	Alarm	Medium	Battery Low
2	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the upper CEX bus segment are disabled.
3	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the lower CEX bus segment are disabled.
4	ExtendedStatus	Warning	Alarm	Medium	Hanging or invalid CEX IRQ: A PM has been shut down.
5	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the direct CEX bus segment are disabled.
6	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the indirect CEX bus segment are disabled.

Table 43. PM856 / TP830

Bit	StatusType	Indication	Generation	Severity	Description
29	ErrorsAndWarnings	Warning	Alarm	Medium	RPA
30	ErrorsAndWarnings	Warning	Alarm	Medium	RPB
31	ErrorsAndWarnings	Warning	Alarm	Medium	Battery Low
2	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the upper CEX bus segment are disabled.
3	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the lower CEX bus segment are disabled.
4	ExtendedStatus	Warning	Alarm	Medium	Hanging or invalid CEX IRQ: A PM has been shut down.
5	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the direct CEX bus segment are disabled.
6	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the indirect CEX bus segment are disabled.

Table 44. PM860 / TP830

Bit	StatusType	Indication	Generation	Severity	Description
29	ErrorsAndWarnings	Warning	Alarm	Medium	RPA
30	ErrorsAndWarnings	Warning	Alarm	Medium	RPB
31	ErrorsAndWarnings	Warning	Alarm	Medium	Battery Low
2	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the upper CEX bus segment are disabled.
3	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the lower CEX bus segment are disabled.
4	ExtendedStatus	Warning	Alarm	Medium	Hanging or invalid CEX IRQ: A PM has been shut down.
5	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the direct CEX bus segment are disabled.
6	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the indirect CEX bus segment are disabled.

Table 45. PM861 / TP830

Bit	StatusType	Indication	Generation	Severity	Description
23	ErrorsAndWarnings	Warning	Alarm	Medium	CEX-bus fuse on Backup
24	ErrorsAndWarnings	Warning	Alarm	Medium	RPB on Backup
25	ErrorsAndWarnings	Warning	Alarm	Medium	RPA on Backup
26	ErrorsAndWarnings	Warning	Alarm	Medium	Battery Low on Backup
28	ErrorsAndWarnings	Warning	Alarm	High	CEX-bus fuse
29	ErrorsAndWarnings	Warning	Alarm	Medium	RPA
30	ErrorsAndWarnings	Warning	Alarm	Medium	RPB
31	ErrorsAndWarnings	Warning	Alarm	Medium	Battery Low
0	ExtendedStatus	Warning	Alarm	Medium	Backup CPU stopped
1	ExtendedStatus	Warning	Alarm	Medium	Switchover occurred
2	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the upper CEX bus segment are disabled.
3	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the lower CEX bus segment are disabled.
4	ExtendedStatus	Warning	Alarm	Medium	Hanging or invalid CEX IRQ: A PM has been shut down.

Table 45. PM861 / TP830 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
5	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the direct CEX bus segment are disabled.
6	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the indirect CEX bus segment are disabled.
7	ExtendedStatus	Warning	Alarm	Medium	RCUcable connector is open

Table 46. PM864 / TP830

Bit	StatusType	Indication	Generation	Severity	Description
23	ErrorsAndWarnings	Warning	Alarm	Medium	CEX-bus fuse on Backup
24	ErrorsAndWarnings	Warning	Alarm	Medium	RPB on Backup
25	ErrorsAndWarnings	Warning	Alarm	Medium	RPA on Backup
26	ErrorsAndWarnings	Warning	Alarm	Medium	Battery Low on Backup
28	ErrorsAndWarnings	Warning	Alarm	High	CEX-bus fuse
29	ErrorsAndWarnings	Warning	Alarm	Medium	RPA
30	ErrorsAndWarnings	Warning	Alarm	Medium	RPB
31	ErrorsAndWarnings	Warning	Alarm	Medium	Battery Low
0	ExtendedStatus	Warning	Alarm	Medium	Backup CPU stopped

Table 46. PM864 / TP830 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
1	ExtendedStatus	Warning	Alarm	Medium	Switchover occurred
2	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the upper CEX bus segment are disabled.
3	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the lower CEX bus segment are disabled.
4	ExtendedStatus	Warning	Alarm	Medium	Hanging or invalid CEX IRQ: A PM has been shut down.
5	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the direct CEX bus segment are disabled.
6	ExtendedStatus	Warning	Alarm	High	Hanging or invalid CEX IRQ: All CEMs on the indirect CEX bus segment are disabled.
7	ExtendedStatus	Warning	Alarm	Medium	RCUcable connector is open

Table 47. CF Card

Bit	StatusType	Indication	Generation	Severity	Description
27	ErrorsAndWarnings	Warning	Alarm	Medium	Image is corrupt
28	ErrorsAndWarnings	Warning	Alarm	Medium	Controller version mismatch
29	ErrorsAndWarnings	Warning	Alarm	Medium	Invalid save setting
30	ErrorsAndWarnings	Warning	Alarm	Medium	Application version mismatch
31	ErrorsAndWarnings	Warning	Alarm	Medium	No card present

Table 48. CI852

Bit	StatusType	Indication	Generation	Severity	Description
22	ErrorsAndWarnings	Warning	Alarm	Medium	Ext FF Config missing
25	ErrorsAndWarnings	Error	Alarm	High	CIff DB Compatibility Error
26	ErrorsAndWarnings	Warning	Alarm	Medium	CIff EEPROM error
27	ErrorsAndWarnings	Error	Alarm	High	CIff Power Up Test Fail
28	ErrorsAndWarnings	Error	Alarm	High	Ctrl WD Stall
29	ErrorsAndWarnings	Error	Alarm	High	CIff WD Stall
30	ErrorsAndWarnings	Error	Alarm	High	Syst Mgm Not Op
31	ErrorsAndWarnings	Warning	Event	Medium	H1 Bus Idle

Table 49. CI854

Bit	StatusType	Indication	Generation	Severity	Description
23	ErrorsAndWarnings	Error	Alarm	Medium	Hardware watchdog on CI854(A) expired
24	ErrorsAndWarnings	Error	Alarm	Medium	Error in PROFIBUS master configuration
25	ErrorsAndWarnings	Warning	Alarm	Medium	PROFIBUS com. failure between Primary and Backup
26	ErrorsAndWarnings	Warning	Event	High	Communication memory obtained too long
27	ErrorsAndWarnings	Warning	Alarm	Medium	Duplicate slave address
28	ErrorsAndWarnings	Warning	Alarm	Medium	No activity on PROFIBUS line A
29	ErrorsAndWarnings	Warning	Alarm	Medium	No activity on PROFIBUS line B
30	ErrorsAndWarnings	Error	Alarm	High	Hardware fail of CI854(A)
31	ErrorsAndWarnings	Error	Alarm	Medium	Firmware needs to be reloaded
0	ExtendedStatus	Warning	Event	Low	Timeout on bus, maybe duplicate slave address (TTO)
1	ExtendedStatus	Warning	Event	Low	Bus synchronization failure, check hardware (SYN)

Table 49. CI854 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
2	ExtendedStatus	Warning	Event	Low	Taken out of ring by another master, check system conf.
3	ExtendedStatus	Warning	Event	Low	Fatal medium access error
4	ExtendedStatus	Warning	Event	Low	Fatal hardware error
5	ExtendedStatus	Warning	Alarm	Medium	All slaves failed
6	ExtendedStatus	Warning	Event	Low	Hardware configuration error on backup
7	ExtendedStatus	Warning	Event	Low	Backup device not found
8	ExtendedStatus	Warning	Alarm	Medium	I/O configuration error on backup
9	ExtendedStatus	Warning	Alarm	Medium	I/O connection error on backup
10	ExtendedStatus	Warning	Event	Low	Hardware watchdog on Backup CI854(A) expired
11	ExtendedStatus	Warning	Event	Low	Error in PROFIBUS master configuration of Backup
12	ExtendedStatus	Warning	Alarm	Medium	No activity on PROFIBUS line A of Backup

Table 49. CI854 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
13	ExtendedStatus	Warning	Alarm	Medium	No activity on PROFIBUS line B of Backup
14	ExtendedStatus	Warning	Alarm	Medium	Hardware fail of CI854A Backup
15	ExtendedStatus	Warning	Alarm	Medium	Firmware needs to be reloaded on Backup
16	ExtendedStatus	Warning	Alarm	Medium	CEX-Bus com. failure between Primary and Backup
17	ExtendedStatus	Error	Alarm	High	Fatal error on Primary detected
18	ExtendedStatus	Warning	Alarm	Medium	Fatal error on Backup detected

Table 50. CI855

Bit	StatusType	Indication	Generation	Severity	Description
22	ErrorsAndWarnings	Warning	Alarm	Medium	No communication on port 1
23	ErrorsAndWarnings	Warning	Alarm	Medium	No communication on port 2
31	ErrorsAndWarnings	Warning	Event	Low	MB300 System message received. Check log-file

Table 51. CI856

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Warning	Event	Low	CPU overload
1	ExtendedStatus	Warning	Event	Low	Scan task overload
2	ExtendedStatus	Warning	Event	Low	Lack of scan resources
3	ExtendedStatus	Warning	Event	Low	PTC status queue full
4	ExtendedStatus	Warning	Event	Low	PTC event queue full
5	ExtendedStatus	Warning	Event	Low	SOE status queue full
6	ExtendedStatus	Warning	Event	Low	DI queue full
7	ExtendedStatus	Warning	Event	Low	AI queue full
8	ExtendedStatus	Warning	Alarm	Medium	Unknown I/O module type
9	ExtendedStatus	Warning	Alarm	Medium	Illegal I/O module ID
10	ExtendedStatus	Warning	Alarm	Medium	I/O module ID conflict
11	ExtendedStatus	Warning	Alarm	Medium	Max number of PTC devices exceeded

Table 52. CI857

Bit	StatusType	Indication	Generation	Severity	Description
22	ErrorsAndWarnings	Error	Alarm	High	FW downl mode
23	ErrorsAndWarnings	Error	Alarm	High	Internal Supv
24	ErrorsAndWarnings	Error	Alarm	High	Appl Task Failed

Table 52. CI857 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
25	ErrorsAndWarnings	Error	Alarm	High	Init Failed
26	ErrorsAndWarnings	Error	Alarm	High	Device Not Found
27	ErrorsAndWarnings	Error	Alarm	High	FW Watchdog Error
28	ErrorsAndWarnings	Error	Alarm	High	Ethernet Error
29	ErrorsAndWarnings	Error	Alarm	High	Device Failure
30	ErrorsAndWarnings	Warning	Event	Low	Warning!
31	ErrorsAndWarnings	Error	Alarm	Medium	Error!
0	ExtendedStatus	Error	Alarm	High	No MAC Addr
1	ExtendedStatus	Error	Alarm	High	HW Fail
2	ExtendedStatus	Error	Event	Medium	Error reading CI status reg.
3	ExtendedStatus	Warning	Alarm	Medium	Suspend State
4	ExtendedStatus	Warning	Alarm	Medium	Shutdown State
6	ExtendedStatus	Warning	Event	Medium	Cfg State
7	ExtendedStatus	Warning	Event	Medium	Init State
8	ExtendedStatus	Error	Alarm	High	Incompat driver version
9	ExtendedStatus	Error	Alarm	High	Incompat FW version
10	ExtendedStatus	Error	Alarm	Medium	PH task stalled
12	ExtendedStatus	Error	Alarm	High	Wrong dev type
13	ExtendedStatus	Warning	Event	Low	Data Trans Q Full
14	ExtendedStatus	Warning	Event	Low	Status Trans Q Full
15	ExtendedStatus	Warning	Event	Low	Misc Trans Q Full

Table 52. CI857 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
16	ExtendedStatus	Warning	Event	Low	Dev Trans Q Full
17	ExtendedStatus	Warning	Event	Low	Trans Q Full
18	ExtendedStatus	Warning	Event	Low	Net Q Full
19	ExtendedStatus	Warning	Event	Low	Intern Q Full
20	ExtendedStatus	Error	Alarm	High	FW Corrupt

Table 53. CI865 (Satt I/O Interface)

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Warning	Alarm	Medium	Overload

Table 54. ModuleBus

Bit	StatusType	Indication	Generation	Severity	Description
29	ErrorsAndWarnings	Warning	Alarm	Medium	RPA
30	ErrorsAndWarnings	Warning	Alarm	Medium	RPB
0	ExtendedStatus	Warning	Alarm	High	Backup CPU, ModuleBus cluster modem error
1	ExtendedStatus	Warning	Alarm	High	Backup CPU, ModuleBus cluster modem error
2	ExtendedStatus	Warning	Alarm	High	Backup CPU, ModuleBus cluster modem error
3	ExtendedStatus	Warning	Alarm	High	Backup CPU, ModuleBus cluster modem error

Table 54. ModuleBus (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
4	ExtendedStatus	Warning	Alarm	High	Backup CPU, ModuleBus cluster modem error
5	ExtendedStatus	Warning	Alarm	High	Backup CPU, ModuleBus cluster modem error
6	ExtendedStatus	Warning	Alarm	High	Backup CPU, ModuleBus cluster modem error

Table 55. Ethernet

Bit	StatusType	Indication	Generation	Severity	Description
29	ErrorsAndWarnings	Warning	Alarm	Medium	No communication Backup CPU
30	ErrorsAndWarnings	Warning	Alarm	Medium	No communication

Table 56. MODBUS

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Warning	Event	Medium	Offline

Table 57. PPP

Bit	StatusType	Indication	Generation	Severity	Description
30	ErrorsAndWarnings	Warning	Event	Medium	No communication

Adapters

Table 58. DSBC 173A

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Alarm	Low	Inhibit
27	ErrorsAndWarnings	Warning	Event	Low	Parity error
28	ErrorsAndWarnings	Warning	Alarm	Low	Regulator failure
29	ErrorsAndWarnings	Warning	Alarm	Low	Regulator missing
30	ErrorsAndWarnings	Warning	Alarm	Medium	Fan failure
31	ErrorsAndWarnings	Warning	Alarm	Low	Voltage warning

Table 59. DSBC 174

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Alarm	Low	Inhibit
27	ErrorsAndWarnings	Warning	Event	Low	Parity error
28	ErrorsAndWarnings	Warning	Alarm	Low	Regulator failure
29	ErrorsAndWarnings	Warning	Alarm	Low	Regulator missing
30	ErrorsAndWarnings	Warning	Alarm	Medium	Fan failure
31	ErrorsAndWarnings	Warning	Alarm	Low	Voltage warning

Table 60. DSBC 176

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Alarm	Low	Inhibit
27	ErrorsAndWarnings	Warning	Event	Low	Parity error
28	ErrorsAndWarnings	Warning	Alarm	Low	Regulator failure
29	ErrorsAndWarnings	Warning	Alarm	Low	Regulator missing
30	ErrorsAndWarnings	Warning	Alarm	Medium	Fan failure
31	ErrorsAndWarnings	Warning	Alarm	Low	Voltage warning

Table 61. CI801

Bit	StatusType	Indication	Generation	Severity	Description
25	ErrorsAndWarnings	Warning	Event	High	Station warning
26	ErrorsAndWarnings	Warning	Alarm	Low	Power B error
27	ErrorsAndWarnings	Warning	Alarm	Low	Power A error
0	ExtendedStatus	Error	Alarm	Medium	Slave does not exist
1	ExtendedStatus	Error	Alarm	Medium	Configuration data fault
2	ExtendedStatus	Error	Alarm	Medium	Parameter data fault
3	ExtendedStatus	Warning	Event	Low	Static diagnostic
5	ExtendedStatus	Warning	Event	Medium	Diagnostic configuration data fault
6	ExtendedStatus	Warning	Event	Medium	Report diagnostics fault
10	ExtendedStatus	Warning	Alarm	Low	Station address warning

Table 62. CI830

Bit	StatusType	Indication	Generation	Severity	Description
28	ErrorsAndWarnings	Warning	Alarm	Low	Power B error
29	ErrorsAndWarnings	Warning	Alarm	Low	Power A error
30	ErrorsAndWarnings	Warning	Event	High	Peripheral HW error
31	ErrorsAndWarnings	Error	Alarm	Medium	Error
0	ExtendedStatus	Error	Alarm	Medium	Slave does not exist

Table 62. CI830

Bit	StatusType	Indication	Generation	Severity	Description
1	ExtendedStatus	Error	Alarm	Medium	Configuration data fault
2	ExtendedStatus	Error	Alarm	Medium	Parameter data fault
3	ExtendedStatus	Warning	Event	Low	Static diagnostic
4	ExtendedStatus	Warning	Alarm	Low	Redundant slave does not exist
5	ExtendedStatus	Warning	Event	Medium	Diagnostic configuration data fault
6	ExtendedStatus	Warning	Event	Medium	Report diagnostics fault

Table 63. CI840

Bit	StatusType	Indication	Generation	Severity	Description
25	ErrorsAndWarnings	Warning	Event	High	Station warning
26	ErrorsAndWarnings	Warning	Alarm	Low	Power B error
27	ErrorsAndWarnings	Warning	Alarm	Low	Power A error
28	ErrorsAndWarnings	Warning	Alarm	Low	Cable B error
29	ErrorsAndWarnings	Warning	Alarm	Low	Cable A error
30	ErrorsAndWarnings	Warning	Alarm	Low	Unit A error
31	ErrorsAndWarnings	Warning	Alarm	Low	Unit B error
0	ExtendedStatus	Error	Alarm	Medium	Slave does not exist
1	ExtendedStatus	Error	Alarm	Medium	Configuration data fault
2	ExtendedStatus	Error	Alarm	Medium	Parameter data fault

Table 63. CI840 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
3	ExtendedStatus	Warning	Event	Low	Static diagnostic
4	ExtendedStatus	Warning	Alarm	Low	Redundant slave does not exist
5	ExtendedStatus	Warning	Event	Medium	Diagnostic configuration data fault
6	ExtendedStatus	Warning	Event	Medium	Report diagnostics fault
10	ExtendedStatus	Warning	Alarm	Low	Station address warning

Table 64. S900

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	-	-	Slave does not exist
1	ExtendedStatus	Error	-	-	Configuration data fault
2	ExtendedStatus	Error	-	-	Parameter data fault
3	ExtendedStatus	Warning	-	-	Static diagnostic
4	ExtendedStatus	Warning	-	-	Redundant slave does not exist

Table 65. CI920* (CIPB)

Bit	StatusType	Indication	Generation	Severity	Description
29	ErrorsAndWarnings	Warning	Alarm	Low	Red. CIPB missing
31	ErrorsAndWarnings	Warning	Event	Low	Red. CIPB error
0	ExtendedStatus	Error	Alarm	Medium	ROM error
1	ExtendedStatus	Error	Alarm	Medium	RAM error

Table 65. CI920 (CIPB) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
2	ExtendedStatus	Error	Alarm	Medium	EEPROM error
3	ExtendedStatus	Warning	Event	Low	Cold start
4	ExtendedStatus	Warning	Event	Low	Error 20
5	ExtendedStatus	Warning	Event	Low	Error 21
6	ExtendedStatus	Error	Event	Medium	Internal bus fault
7	ExtendedStatus	Warning	Event	Low	Internal bus fault (passive)
8	ExtendedStatus	Warning	Event	Low	Power supply 1 error
9	ExtendedStatus	Warning	Event	Low	Power supply 2 error
10	ExtendedStatus	Warning	Event	Low	Reset after watchdog
11	ExtendedStatus	Warning	Event	Low	Redundancy switchover
12	ExtendedStatus	Warning	Event	Low	Red. CIPB missing
13	ExtendedStatus	Warning	Event	Low	Red. CIPB not ready
14	ExtendedStatus	Warning	Event	Low	Red. CIPB error
15	ExtendedStatus	Warning	Event	Low	Red. CIPB no DP comm.

Table 66. RPBA-01 (PROFIBUS DP adapter module)

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Slave does not exist
1	ExtendedStatus	Error	Alarm	High	Configuration data fault
2	ExtendedStatus	Error	Alarm	High	Parameter data fault
3	ExtendedStatus	Warning	Event	Low	Static diagnostic
4	ExtendedStatus	Warning	Event	Medium	Redundant slave does not exist
5	ExtendedStatus	Warning	Event	Medium	Diagnostic configuration data fault
6	ExtendedStatus	Warning	Event	Medium	Report Diagnostics fault
10	ExtendedStatus	Warning	Event	Low	Communication temporary lost
11	ExtendedStatus	Warning	Event	Low	Communication permanently lost

Table 67. NPBA-12 (PROFIBUS DP adapter module)

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Slave does not exist
1	ExtendedStatus	Error	Alarm	High	Configuration data fault
2	ExtendedStatus	Error	Alarm	High	Parameter data fault
3	ExtendedStatus	Warning	Event	Low	Static diagnostic

Table 67. NPBA-12 (PROFIBUS DP adapter module)

Bit	StatusType	Indication	Generation	Severity	Description
4	ExtendedStatus	Warning	Event	Medium	Redundant slave does not exist
5	ExtendedStatus	Warning	Event	Medium	Diagnostic configuration data fault
6	ExtendedStatus	Warning	Event	Medium	Report Diagnostics fault

S800 I/O

Table 68. AI801

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error

Table 68. AI801 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 69. AI810

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError

Table 69. AI810 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 70. AI820 and AI825

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured

Table 70. AI820 and AI825 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 71. AI830

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP

Table 71. AI830 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 72. AI835

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP

Table 72. AI835 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 73. AI843

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning

Table 73. AI843 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 74. AI845

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing

Table 74. AI845 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedWarning	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 75. AI890

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 76. AI893 RTD

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 77. AI893 TC

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 78. AI895

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 79. AO801

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 80. AO820

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 81. AO810

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 82. AO845

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 83. AO890

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 84. AO895

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 85. DI801

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 86. DI802

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 87. DI803

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 88. DI810

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 89. DI811

Bit	StatusType	Indication	Generation	Severity	Description
30	ErrorsAndWarnings	Warning	Event	Low	Warning
0	ExtendedStatus	Warning	Event	Low	OSP
4	ExtendedStatus	Warning	Event	Low	Not configured

Table 90. DI814

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing

Table 90. DI814

Bit	StatusType	Indication	Generation	Severity	Description
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 91. DI820

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error

Table 91. DI820 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 92. DI821

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing

Table 92. DI821 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 93. DI830

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured

Table 93. DI830 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 94. DI825

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error

Table 94. DI825 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 95. DI831

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing

Table 95. DI831 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 96. DI840

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 96. DI840 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error

Table 97. DI885

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError

Table 97. DI885 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 98. DI890

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured

Table 98. DI890 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 99. DO801

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP

Table 99. DO801 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 100. DO802

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP

Table 100. DO802 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 101. DO810

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type

Table 101. DO810 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 102. DO814

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 103. DO815

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 104. DO820

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 105. DO821

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 106. DO840

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 107. DO890

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 108. DP820

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

Table 109. DP840

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Warning	Event	Low	Backup Warning
27	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Error
28	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Module Missing
29	ErrorsAndWarnings	Warning	Alarm	Medium	Backup Wrong module type
30	ErrorsAndWarnings	Warning	Event	Low	Warning
31	ErrorsAndWarnings	Error	Alarm	High	Error
0	ExtendedStatus	Warning	Event	Low	OSP
1	ExtendedStatus	Warning	Event	Low	Backup Not configured
2	ExtendedStatus	Warning	Event	Low	Backup OSP
3	ExtendedStatus	Warning	Event	Low	Backup Process power missing
4	ExtendedStatus	Warning	Event	Low	Not configured
5	ExtendedStatus	Warning	Event	Low	Backup Internal channel error
6	ExtendedStatus	Warning	Event	Low	Backup PulseSyncError
7	ExtendedStatus	Warning	Event	Low	Process power missing
8	ExtendedStatus	Warning	Event	Low	Internal channel error
9	ExtendedStatus	Warning	Event	Low	Channel short circuit
10	ExtendedStatus	Warning	Event	Low	Channel open wire
11	ExtendedStatus	Warning	Event	Low	Channel sensor power sup. error

S900 I/O

Table 110. AI910* (AI4)

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 1
5	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 2
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 3
7	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 4
8	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 1

Table 110. AI910 (AI4) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
9	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 2
10	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 3
11	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 4

Table 111. AI920 (AI4I)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 1
5	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 2

Table 111. AI920 (AI4I) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 3
7	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 4
8	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 1
9	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 2
10	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 3
11	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 4

Table 112. AI921 (AI4I U)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3

Table 112. AI921 (AI4I U) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 1
5	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 2
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 3
7	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 4
8	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 1
9	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 2
10	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 3
11	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 4

Table 113. AI930 (AI4H A)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict

Table 113. AI930 (AI4H A) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 1
5	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 2
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 3
7	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 4
8	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 1
9	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 2
10	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 3
11	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 4
12	ExtendedStatus	Warning	Event	Low	HART status available ch. 1
13	ExtendedStatus	Warning	Event	Low	HART status available ch. 2
14	ExtendedStatus	Warning	Event	Low	HART status available ch. 3
15	ExtendedStatus	Warning	Event	Low	HART status available ch. 4

Table 113. AI930 (AI4H A) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
16	ExtendedStatus	Warning	Event	Low	HART communication error ch. 1
17	ExtendedStatus	Warning	Event	Low	HART communication error ch. 2
18	ExtendedStatus	Warning	Event	Low	HART communication error ch. 3
19	ExtendedStatus	Warning	Event	Low	HART communication error ch. 4

Table 114. AI930 (AI4H A 1H)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	-	-	Line fault ch. 1
1	ExtendedStatus	Warning	-	-	Line fault ch. 2
2	ExtendedStatus	Warning	-	-	Line fault ch. 3
3	ExtendedStatus	Warning	-	-	Line fault ch. 4

Table 114. AI930 (AI4H A 1H) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
4	ExtendedStatus	Warning	-	-	Upper limit exceeded ch. 1
5	ExtendedStatus	Warning	-	-	Upper limit exceeded ch. 2
6	ExtendedStatus	Warning	-	-	Upper limit exceeded ch. 3
7	ExtendedStatus	Warning	-	-	Upper limit exceeded ch. 4
8	ExtendedStatus	Warning	-	-	Lower limit underrun ch. 1
9	ExtendedStatus	Warning	-	-	Lower limit underrun ch. 2
10	ExtendedStatus	Warning	-	-	Lower limit underrun ch. 3
11	ExtendedStatus	Warning	-	-	Lower limit underrun ch. 4
12	ExtendedStatus	Warning	-	-	HART status available ch. 1
13	ExtendedStatus	Warning	-	-	HART status available ch. 2
14	ExtendedStatus	Warning	-	-	HART status available ch. 3
15	ExtendedStatus	Warning	-	-	HART status available ch. 4
16	ExtendedStatus	Warning	-	-	HART communication error ch. 1

Table 114. AI930 (AI4H A 1H) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
17	ExtendedStatus	Warning	-	-	HART communication error ch. 2
18	ExtendedStatus	Warning	-	-	HART communication error ch. 3
19	ExtendedStatus	Warning	-	-	HART communication error ch. 4

Table 115. AI930 (AI4H A 4H)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 1

Table 115. AI930 (AI4H A 4H) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
5	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 2
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 3
7	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 4
8	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 1
9	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 2
10	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 3
11	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 4
12	ExtendedStatus	Warning	Event	Low	HART status available ch. 1
13	ExtendedStatus	Warning	Event	Low	HART status available ch. 2
14	ExtendedStatus	Warning	Event	Low	HART status available ch. 3
15	ExtendedStatus	Warning	Event	Low	HART status available ch. 4
16	ExtendedStatus	Warning	Event	Low	HART communication error ch. 1

Table 115. AI930* (AI4H A 4H) (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
17	ExtendedStatus	Warning	Event	Low	HART communication error ch. 2
18	ExtendedStatus	Warning	Event	Low	HART communication error ch. 3
19	ExtendedStatus	Warning	Event	Low	HART communication error ch. 4

Table 116. AI930* (AI4H A 8H)

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 1

Table 116. AI930* (AI4H A 8H) (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
5	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 2
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 3
7	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 4
8	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 1
9	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 2
10	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 3
11	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 4
12	ExtendedStatus	Warning	Event	Low	HART status available ch. 1
13	ExtendedStatus	Warning	Event	Low	HART status available ch. 2
14	ExtendedStatus	Warning	Event	Low	HART status available ch. 3
15	ExtendedStatus	Warning	Event	Low	HART status available ch. 4
16	ExtendedStatus	Warning	Event	Low	HART communication error ch. 1

Table 116. AI930 (AI4H A 8H) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
17	ExtendedStatus	Warning	Event	Low	HART communication error ch. 2
18	ExtendedStatus	Warning	Event	Low	HART communication error ch. 3
19	ExtendedStatus	Warning	Event	Low	HART communication error ch. 4

Table 117. AI931 (AI4H P)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 1

Table 117. AI931 (AI4H P) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
5	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 2
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 3
7	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 4
8	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 1
9	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 2
10	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 3
11	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 4
12	ExtendedStatus	Warning	Event	Low	HART status available ch. 1
13	ExtendedStatus	Warning	Event	Low	HART status available ch. 2
14	ExtendedStatus	Warning	Event	Low	HART status available ch. 3
15	ExtendedStatus	Warning	Event	Low	HART status available ch. 4
16	ExtendedStatus	Warning	Event	Low	HART communication error ch. 1

Table 117. AI931 (AI4H P) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
17	ExtendedStatus	Warning	Event	Low	HART communication error ch. 2
18	ExtendedStatus	Warning	Event	Low	HART communication error ch. 3
19	ExtendedStatus	Warning	Event	Low	HART communication error ch. 4

Table 118. AI931 (AI4H P 1H)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	-	-	Line fault ch. 1
1	ExtendedStatus	Warning	-	-	Line fault ch. 2
2	ExtendedStatus	Warning	-	-	Line fault ch. 3
3	ExtendedStatus	Warning	-	-	Line fault ch. 4
4	ExtendedStatus	Warning	-	-	Upper limit exceeded ch. 1

Table 118. AI931 (AI4H P 1H) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
5	ExtendedStatus	Warning	-	-	Upper limit exceeded ch. 2
6	ExtendedStatus	Warning	-	-	Upper limit exceeded ch. 3
7	ExtendedStatus	Warning	-	-	Upper limit exceeded ch. 4
8	ExtendedStatus	Warning	-	-	Lower limit underrun ch. 1
9	ExtendedStatus	Warning	-	-	Lower limit underrun ch. 2
10	ExtendedStatus	Warning	-	-	Lower limit underrun ch. 3
11	ExtendedStatus	Warning	-	-	Lower limit underrun ch. 4
12	ExtendedStatus	Warning	-	-	HART status available ch. 1
13	ExtendedStatus	Warning	-	-	HART status available ch. 2
14	ExtendedStatus	Warning	-	-	HART status available ch. 3
15	ExtendedStatus	Warning	-	-	HART status available ch. 4
16	ExtendedStatus	Warning	-	-	HART communication error ch. 1

Table 118. AI931 (AI4H P 1H) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
17	ExtendedStatus	Warning	-	-	HART communication error ch. 2
18	ExtendedStatus	Warning	-	-	HART communication error ch. 3
19	ExtendedStatus	Warning	-	-	HART communication error ch. 4

Table 119. AI931 (AI4H P 4H)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 1

Table 119. AI931* (AI4H P 4H) (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
5	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 2
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 3
7	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 4
8	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 1
9	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 2
10	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 3
11	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 4
12	ExtendedStatus	Warning	Event	Low	HART status available ch. 1
13	ExtendedStatus	Warning	Event	Low	HART status available ch. 2
14	ExtendedStatus	Warning	Event	Low	HART status available ch. 3
15	ExtendedStatus	Warning	Event	Low	HART status available ch. 4
16	ExtendedStatus	Warning	Event	Low	HART communication error ch. 1

Table 119. AI931* (AI4H P 4H) (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
17	ExtendedStatus	Warning	Event	Low	HART communication error ch. 2
18	ExtendedStatus	Warning	Event	Low	HART communication error ch. 3
19	ExtendedStatus	Warning	Event	Low	HART communication error ch. 4

Table 120. AI931* (AI4H P 8H)

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 1

Table 120. AI931* (AI4H P 8H) (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
5	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 2
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 3
7	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 4
8	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 1
9	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 2
10	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 3
11	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 4
12	ExtendedStatus	Warning	Event	Low	HART status available ch. 1
13	ExtendedStatus	Warning	Event	Low	HART status available ch. 2
14	ExtendedStatus	Warning	Event	Low	HART status available ch. 3
15	ExtendedStatus	Warning	Event	Low	HART status available ch. 4
16	ExtendedStatus	Warning	Event	Low	HART communication error ch. 1

Table 120. AI931 (AI4H P 8H) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
17	ExtendedStatus	Warning	Event	Low	HART communication error ch. 2
18	ExtendedStatus	Warning	Event	Low	HART communication error ch. 3
19	ExtendedStatus	Warning	Event	Low	HART communication error ch. 4

Table 121. AI950 (TI4 R)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 1

Table 121. AI950 (TI4 R) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
5	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 2
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 3
7	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 4
8	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 1
9	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 2
10	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 3
11	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 4

Table 122. AI950 (TI4 T)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1

Table 122. AI950 (TI4 T) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 1
5	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 2
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 3
7	ExtendedStatus	Warning	Event	Low	Upper limit exceeded ch. 4
8	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 1
9	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 2
10	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 3
11	ExtendedStatus	Warning	Event	Low	Lower limit underrun ch. 4

Table 123. AO910 (AO4)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected

Table 123. AO910 (AO4) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4

Table 124. AO920 (AO4I)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4

Table 125. AO930 (AO4H)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
12	ExtendedStatus	Warning	Event	Low	HART status available ch. 1
13	ExtendedStatus	Warning	Event	Low	HART status available ch. 2
14	ExtendedStatus	Warning	Event	Low	HART status available ch. 3
15	ExtendedStatus	Warning	Event	Low	HART status available ch. 4
16	ExtendedStatus	Warning	Event	Low	HART communication error ch. 1

Table 125. AO930 (AO4H) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
17	ExtendedStatus	Warning	Event	Low	HART communication error ch. 2
18	ExtendedStatus	Warning	Event	Low	HART communication error ch. 3
19	ExtendedStatus	Warning	Event	Low	HART communication error ch. 4

Table 126. AO930 (AO4H 1H)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
12	ExtendedStatus	Warning	Event	Low	HART status available ch. 1

Table 126. AO930* (AO4H 1H) (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
13	ExtendedStatus	Warning	Event	Low	HART status available ch. 2
14	ExtendedStatus	Warning	Event	Low	HART status available ch. 3
15	ExtendedStatus	Warning	Event	Low	HART status available ch. 4
16	ExtendedStatus	Warning	Event	Low	HART communication error ch. 1
17	ExtendedStatus	Warning	Event	Low	HART communication error ch. 2
18	ExtendedStatus	Warning	Event	Low	HART communication error ch. 3
19	ExtendedStatus	Warning	Event	Low	HART communication error ch. 4

Table 127. AO930* (AO4H 4H)

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict

Table 127. AO930* (AO4H 4H) (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
12	ExtendedStatus	Warning	Event	Low	HART status available ch. 1
13	ExtendedStatus	Warning	Event	Low	HART status available ch. 2
14	ExtendedStatus	Warning	Event	Low	HART status available ch. 3
15	ExtendedStatus	Warning	Event	Low	HART status available ch. 4
16	ExtendedStatus	Warning	Event	Low	HART communication error ch. 1
17	ExtendedStatus	Warning	Event	Low	HART communication error ch. 2
18	ExtendedStatus	Warning	Event	Low	HART communication error ch. 3
19	ExtendedStatus	Warning	Event	Low	HART communication error ch. 4

Table 128. AO930 (AO4H 8H)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
12	ExtendedStatus	Warning	Event	Low	HART status available ch. 1
13	ExtendedStatus	Warning	Event	Low	HART status available ch. 2
14	ExtendedStatus	Warning	Event	Low	HART status available ch. 3
15	ExtendedStatus	Warning	Event	Low	HART status available ch. 4
16	ExtendedStatus	Warning	Event	Low	HART communication error ch. 1

Table 128. AO930* (AO4H 8H) (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
17	ExtendedStatus	Warning	Event	Low	HART communication error ch. 2
18	ExtendedStatus	Warning	Event	Low	HART communication error ch. 3
19	ExtendedStatus	Warning	Event	Low	HART communication error ch. 4

Table 129. DO910* (DO4)

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4

Table 130. DO930 (RO6)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error

Table 131. DO940 (TO8)*

Bit	StatusType	Indication	Generation	Severity	Description
25	ErrorsAndWarnings	Warning	Event	Medium	External power supply missing
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2

Table 131. DO940* (TO8) (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Line fault ch. 5
5	ExtendedStatus	Warning	Event	Low	Line fault ch. 6
6	ExtendedStatus	Warning	Event	Low	Line fault ch. 7
7	ExtendedStatus	Warning	Event	Low	Line fault ch. 8

Table 132. DO980* (TO16)

Bit	StatusType	Indication	Generation	Severity	Description
25	ErrorsAndWarnings	Warning	Event	Medium	External power supply missing
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Line fault ch. 5

Table 132. DO980 (TO16) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
5	ExtendedStatus	Warning	Event	Low	Line fault ch. 6
6	ExtendedStatus	Warning	Event	Low	Line fault ch. 7
7	ExtendedStatus	Warning	Event	Low	Line fault ch. 8
8	ExtendedStatus	Warning	Event	Low	Line fault ch. 9
9	ExtendedStatus	Warning	Event	Low	Line fault ch. 10
10	ExtendedStatus	Warning	Event	Low	Line fault ch. 11
11	ExtendedStatus	Warning	Event	Low	Line fault ch. 12
12	ExtendedStatus	Warning	Event	Low	Line fault ch. 13
13	ExtendedStatus	Warning	Event	Low	Line fault ch. 14
14	ExtendedStatus	Warning	Event	Low	Line fault ch. 15
15	ExtendedStatus	Warning	Event	Low	Line fault ch. 16

Table 133. DP910 (FI2 P)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2

Table 133. DP910 (FI2 P) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Line fault ch. 5
5	ExtendedStatus	Warning	Event	Low	Line fault ch. 6
6	ExtendedStatus	Warning	Event	Low	Line fault ch. 7
7	ExtendedStatus	Warning	Event	Low	Line fault ch. 8

Table 134. DP910 (FI2 F)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Line fault ch. 5

Table 134. DP910* (FI2 F) (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
5	ExtendedStatus	Warning	Event	Low	Line fault ch. 6
6	ExtendedStatus	Warning	Event	Low	Line fault ch. 7
7	ExtendedStatus	Warning	Event	Low	Line fault ch. 8

Table 135. DX910* (DIO8)

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Line fault ch. 5
5	ExtendedStatus	Warning	Event	Low	Line fault ch. 6
6	ExtendedStatus	Warning	Event	Low	Line fault ch. 7
7	ExtendedStatus	Warning	Event	Low	Line fault ch. 8

Table 136. DX910* (DIO8 S)

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Line fault ch. 5
5	ExtendedStatus	Warning	Event	Low	Line fault ch. 6
6	ExtendedStatus	Warning	Event	Low	Line fault ch. 7
7	ExtendedStatus	Warning	Event	Low	Line fault ch. 8

Table 137. DX910* (DIO8 8I)

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected

Table 137. DX910 (DIO8 8I) (Continued)*

Bit	StatusType	Indication	Generation	Severity	Description
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Line fault ch. 5
5	ExtendedStatus	Warning	Event	Low	Line fault ch. 6
6	ExtendedStatus	Warning	Event	Low	Line fault ch. 7
7	ExtendedStatus	Warning	Event	Low	Line fault ch. 8

Table 138. DX910 (DIO8 8I S)*

Bit	StatusType	Indication	Generation	Severity	Description
26	ErrorsAndWarnings	Error	Alarm	Medium	Error
27	ErrorsAndWarnings	Error	Alarm	Medium	Parameter inconsistent
28	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module detected
29	ErrorsAndWarnings	Error	Alarm	Medium	Unknown module configured
30	ErrorsAndWarnings	Error	Alarm	Medium	Internal address conflict
31	ErrorsAndWarnings	Error	Alarm	Medium	Module error
0	ExtendedStatus	Warning	Event	Low	Line fault ch. 1

Table 138. DX910* (DIO8 8I S) (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
1	ExtendedStatus	Warning	Event	Low	Line fault ch. 2
2	ExtendedStatus	Warning	Event	Low	Line fault ch. 3
3	ExtendedStatus	Warning	Event	Low	Line fault ch. 4
4	ExtendedStatus	Warning	Event	Low	Line fault ch. 5
5	ExtendedStatus	Warning	Event	Low	Line fault ch. 6
6	ExtendedStatus	Warning	Event	Low	Line fault ch. 7
7	ExtendedStatus	Warning	Event	Low	Line fault ch. 8

S100 I/O

Table 139. DSAI 130/130A (S100 I/O)

Bit	StatusType	Indication	Generation	Severity	Description
28	ErrorsAndWarnings	Warning	Event	Low	Conversion overflow
29	ErrorsAndWarnings	Warning	Event	Low	Conversion time-out
30	ErrorsAndWarnings	Warning	Event	Low	Max ref. level error
31	ErrorsAndWarnings	Warning	Event	Low	Zero ref. level error

Table 140. DSAI 130D

Bit	StatusType	Indication	Generation	Severity	Description
28	ErrorsAndWarnings	Warning	Event	Low	Conversion overflow
29	ErrorsAndWarnings	Warning	Event	Low	Conversion time-out
30	ErrorsAndWarnings	Warning	Event	Low	Max ref. level error
31	ErrorsAndWarnings	Warning	Event	Low	Zero ref. level error

Table 141. DSAI 133/133A

Bit	StatusType	Indication	Generation	Severity	Description
27	ErrorsAndWarnings	Warning	Event	Low	Semaphore time-out
28	ErrorsAndWarnings	Warning	Event	Low	Conversion overflow
29	ErrorsAndWarnings	Warning	Event	Low	Conversion time-out
30	ErrorsAndWarnings	Warning	Event	Low	Max ref. level error
31	ErrorsAndWarnings	Warning	Event	Low	Zero ref. level error

Table 142. DSAX 110

Bit	StatusType	Indication	Generation	Severity	Description
30	ErrorsAndWarnings	Warning	Event	Low	Semaphore time-out
31	ErrorsAndWarnings	Warning	Event	Low	Reference level error

Table 143. DSAX 110A

Bit	StatusType	Indication	Generation	Severity	Description
30	ErrorsAndWarnings	Warning	Event	Low	Semaphore time-out
31	ErrorsAndWarnings	Warning	Event	Low	Reference level error

INSUM Devices

Table 144. INSUM Device

Bit	StatusType	Indication	Generation	Severity	Description
23	ErrorsAndWarnings	Error	Alarm	Medium	GW Connection error
28	ErrorsAndWarnings	Error	Alarm	Medium	Wrong INSUM device type
29	ErrorsAndWarnings	Error	Alarm	Medium	INSUM Device not found
30	ErrorsAndWarnings	Warning	Event	Low	Warning!
31	ErrorsAndWarnings	Error	Alarm	Medium	Error!

Table 145. INSUM Gateway

Bit	StatusType	Indication	Generation	Severity	Description
22	ErrorsAndWarnings	Error	Alarm	Medium	GW Disconnected
23	ErrorsAndWarnings	Error	Alarm	Medium	CI857 Connection error
25	ErrorsAndWarnings	Warning	Event	Low	HA Offline
26	ErrorsAndWarnings	Warning	Event	Low	GW paused
27	ErrorsAndWarnings	Warning	Alarm	Medium	GW shutdown
28	ErrorsAndWarnings	Warning	Event	Low	Status unknown
30	ErrorsAndWarnings	Warning	Event	Low	Warning!
31	ErrorsAndWarnings	Error	Alarm	Medium	Error!

Table 145. INSUM Gateway (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
3	ExtendedStatus	-	-	-	Gateway sending lifelist
4	ExtendedStatus	Warning	Alarm	Medium	Consistency check failed
5	ExtendedStatus	Warning	Event	Low	Switched Offline via LNT

Table 146. Circuit Breaker (INSUM)

Bit	StatusType	Indication	Generation	Severity	Description
23	ErrorsAndWarnings	Error	Alarm	Medium	GW Connection error
28	ErrorsAndWarnings	Error	Alarm	Medium	Wrong INSUM device type
29	ErrorsAndWarnings	Error	Alarm	Medium	INSUM Device not found
30	ErrorsAndWarnings	Warning	Event	Low	Warning!
31	ErrorsAndWarnings	Error	Alarm	Medium	Error!
7	ExtendedStatus	Warning	Event	Low	LocalOpMode
14	ExtendedStatus	Warning	Event	Low	Tripped
15	ExtendedStatus	Warning	Event	Low	Warning

Table 147. MCU (INSUM)

Bit	StatusType	Indication	Generation	Severity	Description
23	ErrorsAndWarnings	Error	Alarm	Medium	GW Connection error
28	ErrorsAndWarnings	Error	Alarm	Medium	Wrong INSUM device type
29	ErrorsAndWarnings	Error	Alarm	Medium	INSUM Device not found
30	ErrorsAndWarnings	Warning	Event	Low	Warning!
31	ErrorsAndWarnings	Error	Alarm	Medium	Error!
3	ExtendedStatus	Warning	Event	Low	Tripped
4	ExtendedStatus	Warning	Event	Low	Alarm
8	ExtendedStatus	Warning	Event	Low	Failsafe
9	ExtendedStatus	Warning	Event	Low	TOLBypass
10	ExtendedStatus	Warning	Event	Low	TestPos
14	ExtendedStatus	Warning	Event	Low	No remote reset
15	ExtendedStatus	Warning	Event	Low	LocalOpMode

FF Devices

Table 148. FF Device

Bit	StatusType	Indication	Generation	Severity	Description
27	ErrorsAndWarnings	Error	Alarm	High	FF CIff Power Up Test Fail
28	ErrorsAndWarnings	Error	Alarm	High	FF Ctrl WD Stall
29	ErrorsAndWarnings	Error	Alarm	High	FF CIff WD Stall
30	ErrorsAndWarnings	Error	Alarm	High	FF Resources Low
31	ErrorsAndWarnings	Warning	Event	Medium	FF H1 Bus Idle

MB300 Nodes

Table 149. MB300 Node

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Warning	Alarm	Medium	Node unreachable

ABB Standard Drive

Table 150. ABB Standard Drive

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Wrong drive type
2	ExtendedStatus	Error	Alarm	High	Wrong application ID
3	ExtendedStatus	Warning	Event	Low	Undefined error
4	ExtendedStatus	Warning	Event	Medium	Undefined error
5	ExtendedStatus	Warning	Event	Medium	Undefined error
6	ExtendedStatus	Warning	Event	Medium	Undefined error

Table 151. ABB Engineering Drive

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Wrong drive type
2	ExtendedStatus	Error	Alarm	High	Wrong application ID
3	ExtendedStatus	Warning	Event	Low	Undefined error
4	ExtendedStatus	Warning	Event	Medium	Undefined error
5	ExtendedStatus	Warning	Event	Medium	Undefined error
6	ExtendedStatus	Warning	Event	Medium	Undefined error

Table 152. ABB Drive Template (basic)

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Wrong drive type
2	ExtendedStatus	Error	Alarm	High	Wrong application ID
3	ExtendedStatus	Warning	Event	Low	Undefined error
4	ExtendedStatus	Warning	Event	Medium	Undefined error
5	ExtendedStatus	Warning	Event	Medium	Undefined error
6	ExtendedStatus	Warning	Event	Medium	Undefined error

Table 153. ABB Drive Template (extension)

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Wrong drive type
2	ExtendedStatus	Error	Alarm	High	Wrong application ID
3	ExtendedStatus	Warning	Event	Low	Undefined error
4	ExtendedStatus	Warning	Event	Medium	Undefined error
5	ExtendedStatus	Warning	Event	Medium	Undefined error
6	ExtendedStatus	Warning	Event	Medium	Undefined error

Process Panel

Table 154. ABB Process Panel

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Slave does not exist
1	ExtendedStatus	Error	Alarm	High	Configuration data fault
2	ExtendedStatus	Error	Alarm	High	Parameter data fault
3	ExtendedStatus	Warning	Event	Low	Static diagnostic
4	ExtendedStatus	Warning	Event	Medium	Redundant slave does not exist
5	ExtendedStatus	Warning	Event	Medium	Diagnostic configuration data fault
6	ExtendedStatus	Warning	Event	Medium	Report Diagnostics fault

ITS

Table 155. ITS

Bit	StatusType	Indication	Generation	Severity	Description
23	ErrorsAndWarnings	Error	-	-	GW Connection error
28	ErrorsAndWarnings	Error	Alarm	Medium	Wrong INSUM device type
29	ErrorsAndWarnings	Error	Alarm	Medium	INSUM Device not found
30	ErrorsAndWarnings	Warning	-	-	Warning!
31	ErrorsAndWarnings	Error	-	-	Error!
0	ExtendedStatus	Warning	-	-	Fuse Ph1 blown
1	ExtendedStatus	Warning	-	-	Fuse Ph2 blown
2	ExtendedStatus	Warning	-	-	Fuse Ph3 blown
3	ExtendedStatus	Warning	-	-	Tripped
4	ExtendedStatus	Warning	-	-	Warning
10	ExtendedStatus	Warning	-	-	Overcurr Ph1
11	ExtendedStatus	Warning	-	-	Overcurr Ph2
12	ExtendedStatus	Warning	-	-	Overcurr Ph3
13	ExtendedStatus	Warning	-	-	Overtemp Ph1
14	ExtendedStatus	Warning	-	-	Overtemp Ph2
15	ExtendedStatus	Warning	-	-	Overtemp Ph3

NAIO ff

Table 156. NAIO

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Communication broken
2	ExtendedStatus	Error	Alarm	High	Communication broken
3	ExtendedStatus	Warning	Event	Low	Communication broken
4	ExtendedStatus	Warning	Event	Medium	Communication broken
5	ExtendedStatus	Warning	Event	Medium	Communication broken
6	ExtendedStatus	Warning	Event	Medium	Communication broken

Table 157. NBIO-21

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Communication broken
2	ExtendedStatus	Error	Alarm	High	Communication broken
3	ExtendedStatus	Warning	Event	Low	Communication broken

Table 157. NBIO-21 (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
4	ExtendedStatus	Warning	Event	Medium	Communication broken
5	ExtendedStatus	Warning	Event	Medium	Communication broken
6	ExtendedStatus	Warning	Event	Medium	Communication broken

Table 158. NBIO-31

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Communication broken
2	ExtendedStatus	Error	Alarm	High	Communication broken
3	ExtendedStatus	Warning	Event	Low	Communication broken
4	ExtendedStatus	Warning	Event	Medium	Communication broken
5	ExtendedStatus	Warning	Event	Medium	Communication broken
6	ExtendedStatus	Warning	Event	Medium	Communication broken

Table 159. NCTI

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Communication broken
2	ExtendedStatus	Error	Alarm	High	Communication broken
3	ExtendedStatus	Warning	Event	Low	Communication broken
4	ExtendedStatus	Warning	Event	Medium	Communication broken
5	ExtendedStatus	Warning	Event	Medium	Communication broken
6	ExtendedStatus	Warning	Event	Medium	Communication broken

Table 160. NDIO

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Communication broken
2	ExtendedStatus	Error	Alarm	High	Communication broken
3	ExtendedStatus	Warning	Event	Low	Communication broken

Table 160. NDIO

Bit	StatusType	Indication	Generation	Severity	Description
4	ExtendedStatus	Warning	Event	Medium	Communication broken
5	ExtendedStatus	Warning	Event	Medium	Communication broken
6	ExtendedStatus	Warning	Event	Medium	Communication broken

Table 161. NDSC

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Communication broken
2	ExtendedStatus	Error	Alarm	High	Communication broken
3	ExtendedStatus	Warning	Event	Low	Communication broken
4	ExtendedStatus	Warning	Event	Medium	Communication broken
5	ExtendedStatus	Warning	Event	Medium	Communication broken
6	ExtendedStatus	Warning	Event	Medium	Communication broken

Table 162. NPCT

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Communication broken
2	ExtendedStatus	Error	Alarm	High	Communication broken
3	ExtendedStatus	Warning	Event	Low	Communication broken
4	ExtendedStatus	Warning	Event	Medium	Communication broken
5	ExtendedStatus	Warning	Event	Medium	Communication broken
6	ExtendedStatus	Warning	Event	Medium	Communication broken

Table 163. NTAC

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Communication broken
2	ExtendedStatus	Error	Alarm	High	Communication broken
3	ExtendedStatus	Warning	Event	Low	Communication broken

Table 163. NTAC (Continued)

Bit	StatusType	Indication	Generation	Severity	Description
4	ExtendedStatus	Warning	Event	Medium	Communication broken
5	ExtendedStatus	Warning	Event	Medium	Communication broken
6	ExtendedStatus	Warning	Event	Medium	Communication broken

Table 164. NWIO

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Communication broken
2	ExtendedStatus	Error	Alarm	High	Communication broken
3	ExtendedStatus	Warning	Event	Low	Communication broken
4	ExtendedStatus	Warning	Event	Medium	Communication broken
5	ExtendedStatus	Warning	Event	Medium	Communication broken
6	ExtendedStatus	Warning	Event	Medium	Communication broken

PPO

Table 165. PPO Type1

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Error	Alarm	Medium	Error
0	ExtendedStatus	Warning	Event	Low	Short circuit
1	ExtendedStatus	Warning	Event	Low	Under-voltage
2	ExtendedStatus	Warning	Event	Low	Over-voltage
3	ExtendedStatus	Warning	Event	Low	Overload
4	ExtendedStatus	Warning	Event	Low	Over-temperature
5	ExtendedStatus	Warning	Event	Low	Wirebreak
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded
7	ExtendedStatus	Warning	Event	Low	Value below lower limit

Table 166. PPO Type2 no data consistency

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Error	Alarm	Medium	Error
0	ExtendedStatus	Warning	Event	Low	Short circuit
1	ExtendedStatus	Warning	Event	Low	Under-voltage
2	ExtendedStatus	Warning	Event	Low	Over-voltage
3	ExtendedStatus	Warning	Event	Low	Overload
4	ExtendedStatus	Warning	Event	Low	Over-temperature
5	ExtendedStatus	Warning	Event	Low	Wirebreak
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded
7	ExtendedStatus	Warning	Event	Low	Value below lower limit

Table 167. PPO Type 2

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Error	Alarm	Medium	Error
0	ExtendedStatus	Warning	Event	Low	Short circuit
1	ExtendedStatus	Warning	Event	Low	Under-voltage
2	ExtendedStatus	Warning	Event	Low	Over-voltage
3	ExtendedStatus	Warning	Event	Low	Overload
4	ExtendedStatus	Warning	Event	Low	Over-temperature
5	ExtendedStatus	Warning	Event	Low	Wirebreak
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded
7	ExtendedStatus	Warning	Event	Low	Value below lower limit

Table 168. PPO Type 3

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Error	Alarm	Medium	Error
0	ExtendedStatus	Warning	Event	Low	Short circuit
1	ExtendedStatus	Warning	Event	Low	Under-voltage
2	ExtendedStatus	Warning	Event	Low	Over-voltage
3	ExtendedStatus	Warning	Event	Low	Overload
4	ExtendedStatus	Warning	Event	Low	Over-temperature
5	ExtendedStatus	Warning	Event	Low	Wirebreak
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded
7	ExtendedStatus	Warning	Event	Low	Value below lower limit

Table 169. PPO Type 4 no data consistency

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Error	Alarm	Medium	Error
0	ExtendedStatus	Warning	Event	Low	Short circuit
1	ExtendedStatus	Warning	Event	Low	Under-voltage
2	ExtendedStatus	Warning	Event	Low	Over-voltage
3	ExtendedStatus	Warning	Event	Low	Overload
4	ExtendedStatus	Warning	Event	Low	Over-temperature
5	ExtendedStatus	Warning	Event	Low	Wirebreak
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded
7	ExtendedStatus	Warning	Event	Low	Value below lower limit

Table 170. PPO Type 4

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Error	Alarm	Medium	Error
0	ExtendedStatus	Warning	Event	Low	Short circuit
1	ExtendedStatus	Warning	Event	Low	Under-voltage
2	ExtendedStatus	Warning	Event	Low	Over-voltage
3	ExtendedStatus	Warning	Event	Low	Overload
4	ExtendedStatus	Warning	Event	Low	Over-temperature
5	ExtendedStatus	Warning	Event	Low	Wirebreak
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded
7	ExtendedStatus	Warning	Event	Low	Value below lower limit

Table 171. PPO Type 5 no data consistency

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Error	Alarm	Medium	Error
0	ExtendedStatus	Warning	Event	Low	Short circuit
1	ExtendedStatus	Warning	Event	Low	Under-voltage
2	ExtendedStatus	Warning	Event	Low	Over-voltage
3	ExtendedStatus	Warning	Event	Low	Overload
4	ExtendedStatus	Warning	Event	Low	Over-temperature
5	ExtendedStatus	Warning	Event	Low	Wirebreak
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded
7	ExtendedStatus	Warning	Event	Low	Value below lower limit

Table 172. PPO Type 5

Bit	StatusType	Indication	Generation	Severity	Description
31	ErrorsAndWarnings	Error	Alarm	Medium	Error
0	ExtendedStatus	Warning	Event	Low	Short circuit
1	ExtendedStatus	Warning	Event	Low	Under-voltage
2	ExtendedStatus	Warning	Event	Low	Over-voltage
3	ExtendedStatus	Warning	Event	Low	Overload
4	ExtendedStatus	Warning	Event	Low	Over-temperature
5	ExtendedStatus	Warning	Event	Low	Wirebreak
6	ExtendedStatus	Warning	Event	Low	Upper limit exceeded
7	ExtendedStatus	Warning	Event	Low	Value below lower limit

Special IO Template

Table 173. Special IO template

Bit	StatusType	Indication	Generation	Severity	Description
0	ExtendedStatus	Error	Alarm	Medium	Communication broken
1	ExtendedStatus	Error	Alarm	High	Communication broken
2	ExtendedStatus	Error	Alarm	High	Communication broken
3	ExtendedStatus	Warning	Event	Low	Communication broken
4	ExtendedStatus	Warning	Event	Medium	Communication broken
5	ExtendedStatus	Warning	Event	Medium	Communication broken
6	ExtendedStatus	Warning	Event	Medium	Communication broken

INDEX

3BSE040935R201 Rev A. Printed in Sweden   June 2006

http://www.abb.com

Automation Technology Products
Wickliffe, Ohio, USA
www.abb.com/controlsystems

Automation Technology Products
Västerås, Sweden
www.abb.com/controlsystems

Automation Technology Products
Mannheim, Germany
www.abb.de/controlsystems