CI864
Configuration Manual

Table of Contents

1 Section 1 - Introduction

The CI864 is a communication interface board for the CEX bus of a PM8xx controller. It implements the IEC 60870-5-104 protocol. With a Serial to Ethernet converter it can also be used for the IEC −101 and −103 protocols as well as the SAT 1703 protocol.

The CI864 uses the same hardware as the CI857 (INSUM) board, but a different firmware and libraries and is considered to be a completely different device by the 800xA system.

This document describes how to install and configure a CI864 module. It describes the hardware settings and the function blocks of the supplied libraries. Knowledge of the 800xA system in general and 1131 programming and Control Builder in particular are assumed. Esperience with communication protocols and the particular protocol to be used are recommended, especially if advanced functions like redundancy are to be used.

It is not feasible to document every possible configuration in detail. Upon request, we (ABB Austria) can provide support for implementing specific solutions to meet customer needs.

2 Section 2 - Installation

The installation for version 5 is much simpler than for previous versions, simply import the needed libraries (CI864IEC60870HwLib and IEC60870CommLib) and the optional libraries (IEC60870ExtLib, IEC60870SlaveLib and IEC60870MasterLib) into the project/system.

2.1 Dynamic Linked Version

- Unpack the delivery files into a temporary directory.
- Import the needed libraries into the system, insert them into the project and connect them to the controller and application.
-

2.2 Dynamic Linked Version for PLC ControlBuilder

- Unpack the delivery files into a temporary directory.
- Copy the library files into the project directory, insert them into the project and connect them to the controller and application.

3 Section 3 - Configuration of Hardware Parameters

This section describes the configuration options that are set in the hardware tree.

To use a CI864 communication board, at least two three hardware devices have to be configured:

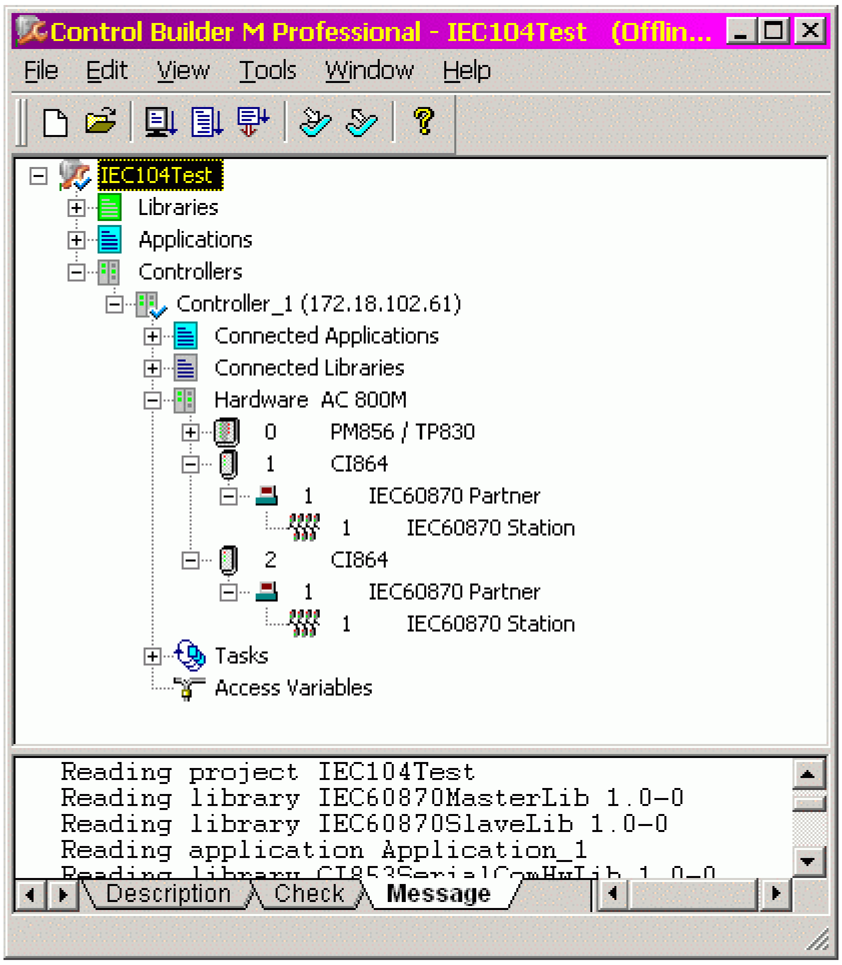
- One device of type **CI864**, which represents the CI864 board itself.
- At least one device of type **IEC60870 Partner**, which represents a connection to a communication partner or a pair of redundant partners.
- At least one device of type **IEC60870 Station** which represents a logical communication partner for each **IEC60870 Partner**.

More than one CI864 communication board can be connected to one AC800M controller. Each CI864 board can handle a maximum of 8 connections.

IEC60870 Partner and **IEC60870 Station** objects must always use consecutive numbers starting from 1. It is not allowed to have a Partner at position 1.1 and one at position 1.3.

Example:

In a configuration with 3 Partners, the second Partner is removed during a revision. In this case the Partner at position 1.3 has to be moved up to position 1.2 and the references to the Partner position in the application have to be updated.


Control Builder M Professional - IEC104Test (Offlin...
File Edit View Tools Window Help
Description Check Message

```
IEC104Test
  Libraries
  Applications
  Controllers
    Controller_1 (172.18.102.61)
      Connected Applications
      Connected Libraries
      Hardware  AC 800M
        0    PM856 / TP830
        1    CI864
          1    IEC60870 Partner
            1    IEC60870 Station
        2    CI864
          1    IEC60870 Partner
            1    IEC60870 Station
      Tasks
      Access Variables
```

```
Reading project IEC104Test
Reading library IEC60870MasterLib 1.0-0
Reading library IEC60870SlaveLib 1.0-0
Reading application Application_1
Reading library CI853SerialComHwLib 1.0-0
```

Figure 1: Sample Configuration

3.1 Description of HW Settings

Some of the parameters are reserved for future expansion. These should be left at the default values. These are indicated as [Not implemented, [value]] in the text and should be set to the given value.

Also, some parameters have setting values that should not be used, these are indicated with [Not implemented].

3.2 CI864 Settings

These parameters apply to the CI864 board as a whole and affect every connection to an IEC partner.

Parameter	Description
Use Redundant CI	Indicates whether this CI864 board is part of a redundant pair. [Not implemented, false]
Use Redundant Network	Enables the second network interface if set to true. The currently used hardware (CI857) does not support this feature. [Not implemented, false]
Primary IP Address	Network address of the primary network interface. Enter a valid IP address.
Secondary IP Address	Network address of the secondary network interface. Enter a valid IP address, or leave empty if the second network is not used. If the parameter ***Use Redundant Network*** is false but a valid address is given here, a second IP Address will be configured for the primary network interface (virtual second network).
IP SubNet Mask	The network mask to be used. This applies to both interfaces. Enter a valid IP network mask.
Default Route	Defines the default gateway (router). Enter the IP address or leave empty if a default gateway is not used.
Use Common Listen Task	Normally only one task can listen on the same port number. If the same port number is to be used by more than IEC task acting as a server, then the Common Listen Task has to be used. If this parameter is set to true, then a separate listen task is used. This task listens on the port given by ***Common Listen Port***, examines the source and destination address of a connect request and hands it over to the correct IEC task for handling (or rejects the request if it does not match a known address pair). In most configurations this parameter has to be true.
Common Listen Port	TCP/IP port used by the Common Listen Task. In most cases this should be left at the default number (2404) as this port is defined by the IEC104 spec.
Time Source	Determines from where the CI864 board takes its time. CONTROLLER Time is taken from the AC 800 Controller. NTP_GET Time is requested from an external NTP Server. NTP_LISTEN [Not implemented]. IEC_PROTOCOL[Not implemented].
Synchronize Controller	If this is set to true, the CI board will set the time of the AC800 Controller. Must be false if the AC800 Controller is set as the ***Time Source***. The time of the AC 800M Controller will only be set if a valid time is received from the external time source. Currently the CI864 can't be directly configured as the time source for the AC 800M Controller. As a workaround the parameter 'CS Protocol Type' must be set to 'No Synch' if the Controller is to be synchronized from the CI864.
IP Address NTP Server	The IP Address of an NTP Server. Only applicable if the ***Time Source*** is set to NTP_GET.
IP Address NTP Server2	IP Address of a secondary NTP Server that is contacted if the primary NTP Sever can not be reached. Only applicable if the ***Time Source*** is set to NTP_GET.
Debug IP Address	IP Address to which debugging messages are sent. Should be left empty. Discussion of this feature is outside the scope of this document.
Debug Port	TCP Port used for debugging, used together with ***Debug IP Address***.

3.3 IEC60870 Partner Settings

These parameters apply to one connection to an IEC partner or to a pair of redundant connections.

Parameter	Description
Protocol Type	Determines which protocol is used by this IEC task. Currently Only IEC104 is implemented. IEC104 — IEC 60870-5-104 Protocol. IEC101Balanced — Implemented, not released for general use. IEC101Master — Implemented, not released for general use. IEC101Slave — Implemented, not released for general use. IEC103Master — Implemented, not released for general use. IEC103Slave — [Not implemented]. Other — Other protocols.
Partner Type	Type of partner. Used to implement specific features for partners that require non-standard behavior. This is a bit mask, setting certain bits will activate special configuration options. See below for details.
Data Sharing ID	Used to share data items with another IEC task. If 0, this task has its own data items (Read and Write function blocks). If not 0, this task shares data items (Read and Write function blocks) with all other tasks that have the same 'Data Sharing ID'. Use the ID variable of the first (lowest number) IEC Partner for all Read and Write function blocks. Written data is sent to all the IEC Partners (except for acknowledge messages to commands, which are sent only to the partner that originally received the command.) The address of received data is changed so that it appears to have come from the first Partner.
Establish Connection Aut At Download	If true, the connection to the IEC partner is established right after the hardware configuration has been downloaded to the CI864 board. [Not implemented, false].
IP Address Primary	IP Address of the IEC partner.
IP Address Secondary	IP Address of the secondary IEC partner in a redundant pair. Use "127.0.0.1" if the secondary address is the same as the primary. Use "127.0.0.2" if the secondary address is the same as the primary and the next higher port number is to be used.
Network interface usage	Determines which parner uses which network interface or the primary /secondary IP address if two addresses are used for the same interface. Currenly only useful if ***Initiating Communication*** is false and this partner uses the Common Listen Task to accept connections. NORMAL FIRST SECOND REVERSED
TCP/IP Port	TCP/IP Port. This applies to both IEC partners. If this is 0 or the same as ***Common Listen Port***, ***Use Common Listen Task*** is true, and ***Initiating Communication*** is false then the Common Listen Task handles this connection.
IP-route primary	If given, configures a specific route for the primary partner IP address rather than the default gateway.
IP-route secondary	If given, configures a specific route for the secondary partner IP address rather than the default gateway.
Initiating Communication	If this parameter is true, then the IEC task initiates the TCP/IP connection (it is the client), if it is false, it accepts the connection (it is the server).
Control Connection	If true, then the IEC task is the controlling station, if false it is the controlled station. (See handling of STARTDT and STOPDT in the IEC104 spec.)

Timeout Seconds Of Connection Est (t0)	Time to wait after a failure to establish a connection to the IEC partner before the next retry. Only used when ***Initiating Communication*** is true. Defined in IEC104.
Timeout Seconds Of Send Or Test APDUs (t1)	Timeout for acknowledge telegrams. Defined in IEC104.
Timeout Seconds Acknowledge No Data Message (t2)	Time after which an acknowledge telegram is sent if no data telegrams have to be sent. Defined in IEC104.
Timeout Seconds Sending Test Frames (t3)	Time after which test telegrams are sent if no data telegrams are sent in any direction. Defined in IEC104.
Max Diff Receive Sequence Number (k)	Maximum number of telegrams that can be sent before an acknowledge telegram is received. Defined in IEC104.
Latest Ackn After Number Of I-Format (w)	Number of received telegrams after which an acknowledge telegram is sent. Defined in IEC104.
Link address size	Size of the Link Address field. Not used with IEC104, used with IEC101 and IEC103. Must be 1 with IEC103. Defined in IEC104.
Start active only	Only activate data transfer on the active connection. Used to implement a redundant master on a shared physical medium (e.g. a RS485 bus). Now also implemented for IEC104. When set and ControlConnection=TRUE, data transfer is started/stopped (StartDT/StopDT) based on the active/passive state of the connection. When this is set, a connection state of 1 (IP connection ok, data transfer not enabled) does not generate a warning on the HW unit.
Listen passive	When set the passive connection will listen in on the traffic and handle received data normally. This setting is only useful together with the ***Start active only***.
Max Data Size	Maximum number of bytes (octets) of user data in one telegram. This limits the number of information elements that can be sent in one telegram based on the size of one information element. One information element can always be sent, even if it exceeds this size. This does not limit the size of received telegrams.
Max Data Elements	Maximum number of information elements in one telegram. The lower value of ***Max Data Elements*** and the number calculated from ***Max Data Size*** is used. Data elements are split up into as many separate telegrams as needed.
No Of Octets In Common Address	Number of bytes (octets) used for the Common Address. Must be 2 for the IEC104 protocol and 1 for the IEC103 protocol. Defined in IEC104.
No Of Octets In Info Obj Addr	Number of bytes (octets) used for the Information Object Address. Must be 3 for the IEC104 protocol and 2 for the IEC103 protocol. Defined in IEC104.
No Of Octets In Cause Of Transmission	Number of bytes (octets) used for the Cause of Transmission. Must be 2 for the IEC104 protocol and 1 for the IEC103 protocol. Defined in IEC104.
Redundancy	Type of redundancy. Determines what types of data messages are handled or silently ignored when the IEC task is not active. The active partner always handles all messages and both partners handle system messages. The table below indicates if Monitoring or Command data types are handled by the passive partner. NONE No redundancy, all messages are handled. STANDBY No messages are handled.

	SHADOW Incoming Commands and outgoing Monitoring are handled. REVERSED Incoming Monitoring and outgoing Commands are handled. DUAL Incoming Monitoring messages are handled. INCOMING Incoming messages are handled. OUTGOING Outgoing messages are handled.
Send double telegrams	If true, certain data types are sent twice, first as a high priority message without a timestamp and then as a low priority message with the timestamp. Currently only implemented for the SAT 1703 protocol.
Write Buffer Queue Size	Maximum number of telegrams in the outgoing queue. This should be high enough to hold all the data sent by all Write function blocks. A typical value is twice the number of IEC60870Write function blocks for monitoring data types.
Read Buffer Queue Size	Maximum number of telegrams in the incoming queue.
Common Address	Common Address of the IEC task. Used mainly for system telegrams. System telegrams (like a General Acquisition request) received from the partner will only be processed if it is addressed to this address or the broadcast address.
Originator Address	Originator Address. This address is put into all outgoing telegrams. The Originator Address of incoming telegrams is ignored.
Use Local Time	Determines if the time stamps in telegrams are in UTC or in local time. Information about the local time has to be set with the function block SetTimeZoneInfo.
Report invalid time	If set to true, the status of the received timestamp is reported in the Status field with previously unused bits. See below for details.
Status Conversion	The IEC task can convert the data point status information from the format used by input modules (OPC status values) to the status format used by IEC104 and back. This status conversion is done for all data types that use the status bits definitions for binary and analog data types. Among the standard data types of the IEC104 protocol, these are data types 1 - 14 and 30 - 36. NONE No status conversion. Conversion has to be done in the controller. OPC_STD Standard conversion for the data types given above. OPC_ALT [Not implemented]. The Encoding / Decoding function blocks in the supplied libraries assume that this parameter is set to OPC_STD.
Data type translation	Normally IEC data points are passed directly through the CI board. In some cases data points of certain data types have to be sent using a different type, e.g. during a General Acquisition a data point of type 30 (Single Point with time) is sent as type 1 (Single point without time). Which data point types are translated is defined in the conversion list. This parameter determines if the translation is performed or not. If the incoming translation is performed, all data points of the configured types are translated. E.g. a data point of type 1 is always changed to type 30. If the outgoing translation is performed, data types are translated during a GA. It is still possible to send data points of the "short" types. NONE No type tranlation is performed. IN Received types are expanded. OUT Sent types are reduced when sent because of a GA. BOTH Both received and sent types are transformed.
Command Max Act Wait	Maximum time in seconds that the IEC task waits for the activation confirmation for a command. If no confirmation is received, the command is cleared and a negative acknowledge is sent. If ***Command Termination*** or ***SetPoint Termination*** (whichever applies) is 0, then this is the time the command will stay active before it is cleared.

Command Max Term Wait	Maximum time in seconds that the IEC task waits for the activation termination after the activation confirmation was received. If no confirmation is received, the command is cleared and a negative acknowledge is sent.
Command Termination	Number of messages that are used to confirm a command. Among the standard data types, this is used for data types 45 - 47, 51 and 58 - 60 and 64. 0 No confirmation. 1 One confirmation message (activation confirmation) 2 Both activation confirmation and activation termination are used.
SetPoint Termination	Same as ***Command Termination***, except that this parameter applies to SetPoint data types. Among the standard data types, this is used for data types 48 - 50 and 61 - 63..
Command clear wait	Time after a command has finished when it is cleared. The command is cleared by sending it to the controller with a COT of 0.
Request GA	Determines if and when the IEC task requests a GA (General Acquisition) from the IEC partner. NO Never. CONNECT After a connection is established. ACTIVATE When the active partner establishes a connection and when a previously passive partner becomes active.
BG Data Send	Determines if and when the IEC task sends its data to the IEC partner with a background scan. This can be used in place of a GA if the IEC partner does not request a GA on its own for some reason. This should also be used in a fully redundant system (2 partners on each side with a total of 4 connections) as it is possible that no GA request will be answered in certain configurations. NO Never. CONNECT After a connection is established. ACTIVATE When the active partner establishes a connection and when a previously passive partner becomes active.
BG Send Cycle	[Not implemented, 0].
BG Tele Send Wait	Time in milliseconds between telegrams sent during a background scan.
Use delay buffer	Since the function blocks in the controller are executed cyclically, it is possible to miss data changes when data messages of the same data point if they happen faster than the cycle time of the receiving application. If this feature is enabled, a delay buffer enforces a minimum delay between data messages for the same data point from the CI to the controller. Which data types are affected is configured in the conversion list.
Msg delay	Minimum delay between messages. If a second message for the same data point is received before this time is elapsed, the second message is delayed so it is sent to the controller after this delay time. This time must be at least as long as the cycle time of the receiving task, it is recommended that it is about two times that, especially if the task has a short cycle time.
Msg max delay	Maximum time that a message is delayed. Mainly used so a message that is sent very often (e.g. it toggles continuously because of a faulty input module) does not "block" the receiving of this message for a long time.
Use trans data buffer	Controls if the Transparent Data buffer is used. This buffer translates transparent data messages to/from data point items. Currently the Transparent Data buffer is only used for Disturbance transmission. If ***Disturbance transmission type*** is NONE, set this to FALSE, otherwise set it to TRUE.
Trans data width	Number of data points that the transmission buffer uses. Currently only with of 16 is implemented.
Trans data delay	Delay between data point changes. Should be at least 2 times the cycle time of the involved application tasks in the controller.

Trans data channels	Number of channels; currently only a setting of 1 or 2 is possible. 2 channels are used for redundant configurations.
Disturbance transmission type	NONE — Disturbance transmission not used. 103M — 103 Master, used with 104S. [Not implemented]. 103MTP — 103 Transparent Master used with 104SAT. 104S — 104 Slave, used with 103M. [Not implemented]. 104SAT — 104 SAT Slave, used with 103MTP.
DisTransMsg timeout	Timeout for answers to transmission requests.
DisTransChan1	Primary transmisson channel, normally 0.
DisTransChan2	Secondary transmission channel, normally 1.
DisTransCmd queue size	Size of queue for transmission messages in command direction.
DisTransMsg queue size	Size of queue for transmission messages in monitoring direction.
DisTransSATCmdIOA	Information Object Address of command messages for disturbance transmission. Used only if ***Disturbance transmission type*** is 104SAT.
DisTransSATMsgIOA	Information Object Address of data messages for disturbance transmission. Used only if ***Disturbance transmission type*** is 104SAT.

3.4 IEC60870 Station Settings

These parameters apply to one station connected through an IEC partner.

Parameter	Description
Link Address	Link Address of the station. Set to 0 if the Link Address is not relevant. Stations can share the same Link Address, e.g when one physical station has two logical stations or if the data of a second station is routed through the first.
Poll Delay	Delay between poll telegrams in milliseconds. This delay is only used when the slave device does not have any data to send. Only relevant for the Master station in a master-slave setup.
Poll Options	Polling options. This is a bit mask, setting certain bits will activate special configuration options. See below for details. Only relevant for a master-slave setup.
Partner Common Address	Common Address of the partner station. This address is used for system telegrams sent to that station. Data messages can be sent and received with arbitrary addresses.
Originator Address	[Not implemented, 0].
Initialisation handling	This parameter determines if and how a received End-Of-Initialisation telegram is handled. NONE — No special handling. GA — A GA is requested. TIMESYNC — A GA is requested. Only TimeSync messages are sent before the EOI telegram is received. DATA — A GA is requested. No data messages are sent before the EOI telegram is received.
Send time sync	Time synchronization interval in seconds. If this is 0, no Time Sync telegrams are sent, otherwise Time Sync telegrams are sent at this frequency. If not 0, a Time Sync telegram is also sent when the connection is established.
Options	Options for this station at the Partner/Message level. This is a bit mask, setting certain bits will activate special configuration options. See below for details.

3.5 Network Configuration

There are three main network configuration options:

- Single Interface.
  One network interface and a single IP address. Only the ***Primary IP Address*** parameter contains a valid IP address.
- Virtual Second Interface.
  One network interface but two seperate IP addresses. The ***Use Redundant Network*** parameter is false and the ***Secondary IP Address*** parameter contains a valid IP address.
- Physical Second Interface.
  Two seperate network interfaces. The ***Use Redundant Network*** parameter is true and the ***Secondary IP Address*** parameter contains a valid IP address. [Not implemented].
  The current hardware (CI857) does not support this configuration.

3.6 Protocol Type

Determines which protocol is used by this IEC task.
Currently IEC104, 101 an 103 are implemented.

Some other protocols are implemented, but not released for general use, mainly because these protocols have not been tested with the full range of configuration options but only with the specific configuration needed for a particular project. The implementation of these protocols may also be missing certain features.

All the protocols can be used, but any protocol other than IEC104 should only be used after specific consultation with ABB Vienna.

Value	Description
IEC104	IEC 60870-5-104 Protocol. This type always uses a balanced point-to-point communication.
IEC101Balanced	IEC 60870-5-101 Protocol in Balanced mode. Implemented, not released for general use.
IEC101Master	IEC 60870-5-101 Protocol as Master in Unbalanced mode. A master can connect to one or several stations. Implemented, not released for general use.
IEC101Slave	IEC 60870-5-101 Protocol as Slave in Unbalanced mode. A slave can share a "party line" with several other slaves; it ignores all telegrams that are not addressed to its Link Address. Implemented, not released for general use.
IEC103Master	IEC 60870-5-103 Protocol as Master. A master can connect to one or several stations, typically protection relays.
IEC103Slave	IEC 60870-5-103 Protocol as Slave. A slave can share a "party line" with several other slaves; it ignores all telegrams that are not addressed to its Link Address. Implemented, not released for general use.
Other	Other protocols. The specific protocol to be used is specified with the ***Partner Type*** setting.

Both IEC101/103-Master and IEC101/103-Slave use half-duplex transfer. That is the slave only sends data as a response to a request from the master. This allows a multi-drop configuration.

3.7 Partner Type

The ***Partner Type*** setting is treated as a bitfield, that is the values of different settings are added together to calculate the value for this parameter. It is used for various settings that do not currently have their own parameter. It is intended that new parameters will be created for these settings at a later date.

Value	Description
0x8000 / 32768	Commands use Originator Address. If this option is set, Commands get/put the Originator Address in the upper word of the COT instead of using the Originator Address setting. With this the Originator Address of received Commands can be reflected back in the confirmation messages.
0x4000 / 16384	CA Offset for RT messages. Use a CA (Common Address) offset for received messages with time (CV_DTT_REDUCE in CV). If Data Type Translation for incoming messages is used, both messages (RT and non-RT e.g. ADSU types 30 and 1, respectively) will be received with the type with the expanded type (the type with a timestamp, e.g. 30).
0x2000 / 8192	Clear Buffer. If this option is set, the any data received directly (2 seconds) after establishing the communication link is discarded. This delays establishing the connection at the higher protocol layers, but prevents cases where the connection cannot be properly established because a device in the connection (comm server) buffers received data. This option is only applicable to serial protocols that use an external Com server (not IEC 60870-5-104). It is recommended to set this option for all serial connections, but it should definitely be set for balanced transmissions and when the CI864 implements a slave.
0x1000 / 4096	High Priority Polling. Decrease idle times in flow control loop. Should be used for serial protocols, only for one Partner and only if the CI864 board does not have to handle many connections. Using this option increases the CPU load of the CI864 board.
Mask: 0x0F00	Protocol Mask.
0x0000 / 0	No special selection. Use this value for all Protocol Types except "Other", in which case this setting is not allowed.
0x0100 / 256	SAT 1703 protocol. Implemented, not released for general use.
0x0400 / 1024	I35 Master. Implemented, not released for general use.
0x0500 / 1280	I35 Slave. [Not implemented].
0x0600 / 1536	Allen-Bradley Master. Implemented, not released for general use.
0x0700 / 1792	Allen-Bradley Slave. [Not implemented].
0x0800 / 2048	Allen-Bradley Balanced (Full Duplex). [Not implemented].
Mask: 0x00C0	Command buffer size Maximum number of concurrent commands per Partner. 0x0000 / 0 4 commands 0x0040 / 64 12 commands 0x0080 / 128 30 commands 0x00C0 / 192 64 commands This setting configures how many commands the Partner can handle at the same time. A command occupies one of these slots until it is finished (termination message sent/received or it times out) and the additional Command clear wait time has elapsed.

0x0020 / 32	Ignore data items with the "Blocked" bit set in the item status. Ignore data items with COT=2 (BG GA).
0x0010 / 16	Use weekday field in timestamp. If this option is set, the weekday field is set according to the current date, otherwise it is 0. The weekday field is ignored in received telegrams.
0x0008 / 8	Always send the GA data on the passive partner. This has priority over bit 1 (send only on active partner). If neither bit is set, the Redundancy setting determines if the GA data is sent on the passive partner. If the passive partner of handles Outgoing Monitoring data, then the GA data is sent.
0x0004 / 4	Use Generic Data services. Used for IEC103 protocol.
0x0002 / 2	When a partner becomes active and requests a GA, send a single GA request to the broadcast address instead of individual GA requests to every station.
0x0001 / 1	The partner only sends data in response to GA requests if it is active, the passive partner confirms the GA requests but does not send any data.
0x0080 / 128	SAT protocol has RT (Real Time) data stream. Used for SAT1703 protocol. With the SAT protocol only the 0x0040 / 64 bit can be used to configure the concurrent commands (4 or 12).

3.8 Poll Options

The PollOptions setting is treated as a bitfield, that is the values of different settings are added together to calculate the value for this parameter. It is used for various settings that do not currently have their own parameter. It is intended that new parameters will be created for these settings at a later date.

Currently this entry is only used by the IEC101 Master and IEC103 Master.

Value	Description

3.9 Options

The ***Options*** setting is treated as a bitfield, that is the values of different settings are added together to calculate the value for this parameter. It is used for various settings that do not currently have their own parameter. It is intended that new parameters will be created for these settings at a later date.

Value	Description
0x0080 / 128	Use Link Address of this Station as the Own Link Address. Only used for IEC101 Balanced.
0x0040 / 64	Use Second Messages for one flank items. Data types and IOA address ranges that can use this feature must be configured through the conversion list. The Delay Buffer must be used for this to work (otherwise the second message will overwrite the first message and it will be lost).
0x0020 / 32	Ignore Station on primary connection.

0x0010 / 16	Ignore Station on secondary connection. This option and the one above only apply to GA Requests and Time Sync telegrams. Setting both is allowed but not particularly useful. Should normally not be set for a station that is used for a broadcast GA (see option 2 below).
0x0002 / 2	Send a GA resquest to this station to to the broadcast address instead of the configured station address. May only be used for the first station.
0x0001 / 1	Send a GA request to this station only in response to an End-of-Initialisation telegram, but not when the connection is established. Usually this option is combined with setting that a broadcast GA is sent and the ***Initialisation Handling*** parameter is GA or higher.

3.10 IEC 60870-5-101 & -103 Specific Settings

When a partner is configured for IEC 101 and IEC 103 protocols, some of the parameters are used differently.

Parameter / Value	Description
Partnertype:	
0x2000 / 8192	Clear Buffer. Applicable here, should usually be used, description above.
Control Connection	Balanced: Physical Transmission Direction. Sets the "DIR" bit for sent telegrams. Master, Slave: Not used.
Timeout Seconds Of Send Or Test APDUs (t1)	Master, Balanced: Time to wait for response before retry. Slave: Time between requests from the master before the connection is set bad. In 100* milliseconds (e.g. 35 give a time of 3.5 seconds).
Timeout Seconds Acknowledge No Data Message (t2)	Balanced: Time between test frames, in seconds. Master, Slave: Not used.
Timeout Seconds Sending Test Frames (t3)	Start delay.
Max Diff Receive Sequence Number (k)	Master: Maximum number of poll telegrams before another station is polled. Must be at least 2. Slave, Balanced: Not used.
Latest Ackn After Number Of I-Format (w)	Master, Balanced: Max. Number of retries. Slave: Not used.
Poll Options	
0x0010 / 16	Change Stations if an "Idle Telegram" is received. Which ASDU types are considered "Idle Telegrams" is defined by the Converion List.
0x0008 / 8	Don't use test telegrams. If this option is set, no test telegrams are sent. Some stations don't answer to this telegram. Link initialisation or Reset FCB telegrams are used to establish connection to a station. Normally test telegrams are only sent to determine the state of the slave station, e.g. when it has stopped data transfer with the DFC bit. Set this option for IEC103.
0x0004 / 4	Use "Reset FCB" telegrams instead of "Init Link" telegrams. Set this option for IEC103.

0x0002 / 2	Test telegram before Init telegram. If this option is set, test telegrams (request link status) are sent to a station after an interruption until a positive response is received and then the communication is initialized with a link initialisation telegram. If this option is not set, link initialisation telegrams are used to determine the connection status and initialize the connection.
0x0001 / 1	Poll High Priority first. If this option is used, the first poll telegram sent to a station after an interruption or after a different station was polled is a request for class 1 data (high priority data), otherwise it is a request for class 2 data. If a slave station sends most or all of its data as class 1 data this can speed up data transfer.

3.11 SAT 1703 Specific Settings

When a partner is configured for SAT 1703 protocol, some of the parameters are used differently.

Parameter	Description
TimeoutSecondsOfSendOrTestAPDUs_t1	Time to wait for response before retry. In 100* milliseconds. (Old default 5 sec or 50)
Timeout Seconds Acknowledge No Data Message (t2)	Time after which a sent telegram is automatically acknowledged. Used for unacknowledged data transfer. Set to 255 if this feature is not used (acknowledged data transfer). In 100* milliseconds. (Old default 255, no auto-ack, when used 2 sec or 20)
Timeout Seconds Sending Test Frames (t3)	Test telegram cycle.
Max Diff Receive Sequence Number (k)	If 1, only one buffer is used, if >1, two buffers are used. Add 16 if Acknowledge telegrams may not be sent as part of data telegrams.
Latest Ackn After Number Of I-Format (w)	Max. Number of retries.
No Of Octets In Common Address	Number of bytes (octets) used for the Common Address. Always 1 for the SAT1703 protocol.
No Of Octets In Info Obj Addr	Number of bytes (octets) used for the Information Object Address. Always 2 for the SAT1703 protocol.

3.12 I35 Specific Settings

When a partner is configured for I35 protocol, some of the parameters are used differently.

Parameter	Description
TimeoutSecondsOfSendOrTestAPDUs_t1	Time to wait for response before retry. In 100* milliseconds.
Timeout Seconds Acknowledge No Data Message (t2)	Not used.
Timeout Seconds Sending Test Frames (t3)	Start delay.
Max Diff Receive Sequence Number (k)	Not used.
Latest Ackn After Number Of I-Format (w)	Max. Number of retries.
No Of Octets In Common Address	Number of bytes (octets) used for the Common Address. Ignored, always 1 for the I35 protocol.
No Of Octets In Info Obj Addr	Number of bytes (octets) used for the Information Object Address. Ignored, always 10 bits for the I35 protocol.

Status Conversion	The I35 protocol always uses the equivalent of the OPC_STD Standard conversion.
Data type translation	For the I35 Master this should be set to IN or BOTH if data items with a time stamp are received, since the data items are always sent without a timestamp during the GA.
Command Max Act Wait	Maximum time in seconds that the IEC task waits for the activation confirmation for a command.
Command Max Term Wait	Not used.
Command Termination	Ignored, the I35 protocol always sends one confirmation message.
SetPoint Termination	Ignored, the I35 protocol always sends one confirmation message.
Command clear wait	Time after a command has finished when it is cleared. The command is cleared by sending it to the controller with a COT of 0.
Request GA	The I35 Master always issues Read-Type-Value telegrams for Group Types 1 (Simple Input) and 5 (Measurands). (Equivalent to CONNECT.)
Options	(Station)
0x0008 / 8	Use Group Type 8 (Double Inputs)

3.13 Allen-Bradley Specific Settings

When a partner is configured for Allen-Bradley protocol, some of the parameters are used differently.

Parameter	Description
Link address size	Type / Length of checksum: 0: None 1: 1 Byte BCC 2: 2 Byte CRC.
TimeoutSecondsOfSendOrTestAPDUs_t1	Time to wait for response / ACK before retry. (TO1) In 100* milliseconds.
Timeout Seconds Acknowledge No Data Message (t2)	Start delay.
Timeout Seconds Sending Test Frames (t3)	Balanced: Time to wait before sending Test telegrams (Diagnostic Loop) after no data was sent in seconds. Slave: Not used.
Max Diff Receive Sequence Number (k)	Master: Maximum number of poll telegrams before another station is polled. Must be at least 2. Slave, Balanced: Not used.
Latest Ackn After Number Of I-Format (w)	Max. Number of retries.
Disturbance transmission type	Nust be set to NONE. Disturbance transmission is not supported, some of the parameters are used for other purposes.
DisTransMsg timeout	Time to wait for response to command (TO23). Master, Slave: Not used.
DisTransChan1	Options (Bitfield):

	1: Raw Data Mode: If set, Data telegrams do not use normal Allen-Bradley Data block structure (DST, SRC, CMD, STS, TNS, FUNC) but all the ingoing / outgoing data is sent as a single data block of x words. 2: Optimize incoming data: If set, incomind data telegrams are checked for changes and only sent to the 1131 application if the data was changed. (Only applicable if Raw Data Mode is set). 4: Slave Mode: DI/AI is outgoing, DO/AO incoming. (Not applicable to Raw Data Mode). 8: Extra Configuration. Non-standard addressing used, this requires a special system telegram to configure the addressing of the different data areas. The HW parameters for these will be ignored (Not applica-ble to Raw Data Mode).
DisTransChan2	
DisTransCmd queue size	
DisTransMsg queue size	
DisTransSATCmdIOA	Raw Data Mode: Size of outgoing Data Area in Words (Address starts with 1) Currently the Size is limited to 32 Words. Not Raw Data Mode: Size of DI and DO Data Area in Words: DI Size + 65536*DO Size. Size 0 is the default size (499).
DisTransSATMsgIOA	Raw Data Mode: Size of incoming Data Area in Words (Address starts with 1) Currently the Size is limited to 32 Words. Not Raw Data Mode: Size of AI and AO Data Area in Words: AI Size + 65536*AO Size. Size 0 is the default size (499).

The Own/Partner Common Addresses are used as follows: The Station Address in the Allen-Brad-ley protocol is only 1 byte, the Common Addresses are built from both:

The Own Common Address should be set to SRC + DST * 256.

The Partner Common Address should be set to DST + SRC * 256.

SRC is the Source Station Address, DST is the Destination Station Address.

In Raw Mode the Data Areas are addressed as follows:

The Common Address is the Common Address of the Slave for Data in both directions.

The Information Object Address is 1 to x (x is the size of the Data Area) for outgoing Data (use ASDU Type 9, RAW_OUT) and 1025 to 1024+y (y is the size of the Data Area) for incoming Data (use ASDU Type 8, RAW_IN).

4 Section 4 - Software Configuration

This section describes how to implement the data transfer in a 1131 application.

4.1 General concepts:

4.1.1 Status Codes

The IEC60870 Read / Write function codes use the following numbers as output on the ***Status*** pin:

Value	Description
<0:	An error has occurred.
0:	Function block is starting up or processing data. The Receive block uses this number to indicate that no data has yet been received.
1:	Function block has processed the last request or delivered valid data.
2:	IEC60870WriteCyc only, new data is requested to be written before the previous request has been satisfied.
9:	The function block is in idle state, that is not Enabled and has finished shutting down the connection to the protocol handler. This result can be used to disable the function block completely when it is not needed to save CPU time. The following logic will usually work: EN := Enable OR (Fb.Status<>9);

4.1.2 Data Sharing

More than one IEC task / IEC partner can share the same data (Read and Write function blocks) if they have the same non-zero value for the HW parameter 'Data Sharing ID'.

If this function is used, several rules have to be obeyed:

All Read and Write function blocks must use the connection Id variable of the Connect block of the first partner (the one with the lowest Partner Position) and this Connect block must be the first to be enabled. Obviously, the connection Id variable must be valid for any Read or Write function block to execute.

Connect function blocks are required for all partners in the group, however.

It is possible to read data if only some IEC tasks are connected, but it will (obviously) only return results from the connected tasks.

Data should only be written after all the IEC tasks are connected, otherwise it will only be written to the tasks that are currently connected. IEC tasks that are connected later will only receive the data if it changes afterwards or the Write function block is disabled and re-enabled.

It is recommended to use the ***Valid*** pin of the first Connect block as the ***Enable*** pin for the next Connect block. The Read and Write function blocks should only be enabled once all the Connect blocks are valid.

These rules also apply to the function blocks from the IEC60870SupLib that wrap Read and Write function blocks.

4.1.3 Structured Data Types

Structured data types are defined for all the implemented ASDU types. While it is possible to use standard data types to transfer data from/to the IEC protocol, in most cases it makes more sense to use the structured data types.

The first item in each type is a dint constant initialized with the ASDU type number. The second data item is used for the most important data field of the ASDU type. This is the only field that can be transferred if a basic data type is used rather than a structured type. For most data types it makes sense to use more than the first data field, for some it is needed. Unless memory usage on the AC800M controller is a real problem, it is recommended to use the structured data types.

In SV 5 many data types have been extended with additional information, making it almost impossible to use anything but the structured data types.

The fields in the structured data type do not have to be in the same order as the data fields in the IEC telegram. The conversion list defines how the structured variable is translated into the IEC type and vice versa.

4.1.4 Status Conversion

If desired, the CI864 module can translate the IEC telegram status bits into the OPC status values used by AC800 I/O modules. If the HW parameter 'Status Conversion' is "NONE" the status is simply copied and the IEC status bits have to be set / decoded by the 1131 application.

If the HW parameter 'Status Conversion' is "OPC_STD" the CI864 translates the status codes used in binary and analog monitoring status types. These are ASDU types 1 - 14 and 30 - 36 in the compatible range.

When receiving telegrams, the following rules are used (the last applicable rule is used):

If no IEC status bit is set, the status is OPC_QUALITY_GOOD (0xC0).

If the SB (simulated) bit is set, the status is OPC_QUALITY_LOCAL_OVERRIDE (0xD8).

If the IV (invalid) bit is set, the status is OPC_QUALITY_BAD (0x00).

If the NT (not topical) bit is set, the status is OPC_QUALITY_ABB_ISP | OPC_QUALITY_DEVICE_FAILURE (0x10000C).

If the OV (overflow) bit is set, the OPC_LIMIT_LOW (0x01) bit is set (in addition to one of the above codes).

When sending telegrams, the following rules are used:

Status quality OPC_QUALITY_BAD (0x00) or OPC_QUALITY_UNCERTAIN (0x40):
If the OPC_QUALITY_ABB_ISP (0x100000) is set, the status bit NT (not topical) is set, if not the IV (invalid) bit is set.

Status quality OPC_QUALITY_GOOD (0xC0):
If the status is OPC_QUALITY_LOCAL_OVERRIDE (0xD8) the SB (simulated) bit is set, if not no IEC status bits are set.

If any Limit status bits are set OPC_LIMIT_LOW (0x01) or OPC_LIMIT_HIGH (0x02), the OV (overflow) bit is set.

When configured, the quality of the timestamp is indicated by the following bits:

OPC_QUALITY_TIME_INVALID (0x01000000): The timestamp was received with the invalid timestamp bit set.

OPC_QUALITY_TIME_SUBST (0x02000000): The timestamp was substituted. Used with data type translation when a telegram without a timestamp is translated into one with a timestamp. When sending data this bit (as well as the previous one) will cause the timestamp to be marked as invalid if set.

OPC_QUALITY_TIME_IGNORE (0x04000000): Used only when sending data. Causes the timestamp on the structured variable to be ignored.

4.1.5 Timestamp Handling

Normally a timestamp in the protocol is directly copied (after translating it) into the structured variable (or vice versa). The timestamp in a structured variable is alway in UTC. Several special cases are handled as described below:

When receiving telegrams, the following rules are used:

If a valid and complete timestamp is received, it is used as is. If the CI864 is configured to use local time (HW parameter 'Use Local Time' is true), then the timestamp is converted to UTC using the rules set through the SetTimeZoneInfo function block.

If a partial timestamp is received (e.g. a timestamp that contains only seconds and minutes but not the hour or date), then the missing information is filled in from the current system time. If this puts the resulting timestamp too far into the future, it is shifted into the past.
E.g. when the timestamp contains only minutes and seconds and the resulting timestamp is more than 5 minutes in the future, one hour is subtracted from the timestamp.

When an invalid timestamp is received, it is ignored and the current system time is substituted. If reporting invalid timestamps is enabled, this is indicated with a bit in the status entry in the structured variable.

When Data Type Translation is used, a current system time is used for telegrams without a timestamp. If reporting invalid timestamps is enabled, the substituted timestamp is indicated with a bit in the status entry in the structured variable.

When sending telegrams the following rules are used:

If the structured variable contains a valid timestamp this is used. If the CI864 is configured to use local time (HW parameter 'Use Local Time' is true), then the timestamp is converted to local time using the rules set through the SetTimeZoneInfo function block.

If not timestamp is used or the timestamp is Zero (all bits 0), then the current system time is used.

If reporting invalid timestamps is enabled and the bits in the status word that indicate an invalid or substituted timestamp are set, the timestamp is marked as invalid.

When reporting invalid timestamps is enabled, another bit indicates that the timestamp should be ignored. If this bit is set, the timestamp is handled as if it was Zero, as above. This bit has priority over the invalid or substituted bits.

Invalid timestamp reporting is only used when the status conversion is OPC_STD and only for monitoring types.

4.1.6 Data Type Translation

Normally each data item is always sent with the same data type. In other configurations a single data item can be sent with different data types depending on the reasons why a telegram is sent. According to a strict interpretation of the IEC spec, data types that include a timestamp are sent as the corresponding type without a timestamp during a GA (General Acquisition). Data Type Translation is used to hide this behavior from the 1131 application.

Data Type Translation only affects data types for which a translation rule has been defined in the Conversion List.

Expand Incoming Data Types:
If this option is used, incoming data types without a timestamp are translated into the corresponding type with a timestamp. The current system time is used for the timestamp.
All received telegrams with the affected data types will be translated. If some data items actually use the data type without the timestamp, they are also translated and the structured type for the data type with the timestamp has to be used in the 1131 application.

Reduce Outgoing Data Types:
If this option is used, data types that normally contain a timestamp are sent as the corresponding type without a timestamp during a GA (General Acquisition). It is possible to send data items of the type without a timestamp; they are not affected by this setting.

4.1.7 Using Dummy Data Items to reduce the number of Function Blocks

The function blocks used to receive and send data (and to a lesser extend the blocks used to en/decode the data have a significant overhead to set up the data connection and such. It takes a lot less resources (memory and CPU load) to use one function block to send three data items than to use three function blocks that send one data item each.

Combining the blocks is only feasible if the addressing of the data items follows a regular pattern, since the Receive and Write blocks use a starting address (***InformationObjectAddress*** input pin) and a offset (***InfoObjAddr_StepSize*** input pin) that is added for each following data item. For example, one common "addressing system" is set up similarly to the way Hardware addresses are used: The first byte of the address (IOA1) is based on the data type (it identifies the type of "input/output module"), the second byte (IOA2) gives the number / position of the "module" while the third byte (IOA3) gives the signal within the "module" (counting up from 0 to 15).

When the addressing does not follow such a scheme closely, it is sometimes still possible to combine function blocks by using dummy data items in between the real data items. This can also be used if the addessing system uses a clear pattern but data items are removed during the implementation process and the resulting gaps in the addressing are not filled so that the addresses of existing items is not changed.

For example, if data items with addresses 1,2, 4 and 5 are sent, one could either use two function blocks with a starting address of 1 and 4 (and a step size of 1) or use one function block with an additional dummy data item for address 3.

Points to consider when combining function blocks:

- Dummy data items connected to a Write block are actually sent to the partner. With most partners this is not a big problem as unconfigured data items will simply be ignored, but that is not always the case. If the number of dummy items is high, this can slow down the data transmission. Since the dummy data items don't change their value, they will typically only be sent once (during the Initialisation).
- If the number of dummy items is high, it may cause more overhead than splitting the function blocks: If data items with addresses 1, 2 and 4 are sent, it makes sense to add one data item with address 3; if data items with addresses 1, 2 and 15 are sent, it probably does not make sense to add 12 data items for the addresses 3 to 14. On the other hand, having the dummy items connected to the function blocks makes it easy to replace them with actual data items if more data items are added later on.
- Depending on the protocol and the settings for it, sending several data items with a single function block can be faster, because changes to two signals that happen during the same task cycle can be sent as one message if the two signals are connected to the same block, while they must be sent as two seperate messages if they are connected to two different blocks. The specifics of how this is handled depend on the protocol and data types.
- All data items connected to a single Receive or Write block must use the same data type. If you are sending binary signals with addresses 1, 2 and 4 and an analog signal at address 3, then you must split the function blocks (use one block for addresses 1 and 2, one for address 3 and one for address 4).
  In addition, dummy data items must not be used for addresses at which a data item with a different type is configured. In the above example, you can't use one function block for the three binary signals with a dummy binary data item at address 3.
  When sending data, this can mess up the data item cross reference in the CI board, possibly crashing it.
  When receiving data, this can cause the controller to crash as the data for one data type is copied into a data structure meant for a different data type.
- With most Encode / Decode function blocks in the libraries dummy data items can be indicated by setting the Status entry of the BoolIO / RealIO (not of the structured variable connected to the Receive / Write block) to 16#FFFFFFFF. This will cause the block to skip copying the data for this item and reduce the CPU load.
- When receiving data items, one should make sure that no data items with the same address as a dummy item are sent by the partner. This is not a problem if the data items have the same type as the dummy item, but it can crash the controller if they have a different type.
  The Protocol Handler version 2.0/16 (HW library 1.1.72) introduces the possibility for the Receive block to perform a type check and only copy data of the correct type.
  The IEC Library 2.0.29 adds the possibility to use Receive Type Separation (RTS; see below). When RTS is used, each data type uses a separate address space. This relaxes the rules of using dummy items in Receive blocks as an analog signal with address 3 will actually have a different address than a binary signal with address 3.

4.1.8 Receive Type Checking and Receive Type Separation

In some cases the IEC Partner sends a data item on a given address with a different type than is used in the Receive block for that address or the IEC Partner sends data items with different types on the same address (this is allowed in some IEC-104 implementations). The Protocol Handler (PH) does not know that there is a mismatch between the received data and the configuration of the Receive block and will copy data into a data structure that can't contain that data. This can crash the controller. The crash can either occur directly after the data is received or when the connection is broken or disabled.

There are two options to work around this problem:

4.1.8.1 Receive Type Checking (RTC)

This option is available in the Protocol Handler version 2.0/16 (CI864HWLib 1.1.72) or higher.

Receive Type Checking (RTC) is activated by specifying the negative of the data type on the pin ***IECDataTypeToUse*** (e.g. use -30 to specify data type 30). If data of a different type is received it is discarded by the Protocol Handler instead of copying it to the Rd variable of the Receive block.

For some Function blocks (especially combined Receive/Decode blocks) this option is activated by setting the Project Constant cIEC101.ReceiveTypeCheck to true.

RTC is can be used for all protocols.

4.1.8.2 Receive Type Separation (RTS)

This option is available in the IECCommLib 2.0-29 or higher and requires a CI Firmware with Build number 597 with a date 22.11.2011 (CI864HWLib x.x.54) or higher. RTS only works together with Data Type Translation (DTT) for incoming signals (IN/BOTH) with a CI Firmware Build number 703 with date 8.7.2015 (CI864HWLib x.x.88) or higher.

Receive Type Separation (RTS) is activated by setting the Project Constant cIEC101.UseTypeOffset from 0 (the default) to 536870912. Exactly the given value must be used. When RTS is activated the data type to be used must be specified on the pin ***IECDataTypeToUse*** (use 30 for data type 30) for the data types for which RTS is defined (currently monitoring and command ASDU types 1 to 63).

When RTS is used each data type uses its own address space (the data type is used as an offset to the data item address). The Receive blocks automatically perform the calculation of the address offset if the RTS is enabled and the correct data type is specified. Function blocks that wrap Receive blocks (Command Send/Receive, combined Encode/Write and combined Receive/Decode blocks set the Data type automatically when RTS is enabled.

RTS is currently only implemented in the Conversion List for the IEC-104 Protocol (and the -101 Protocol which uses the same data types) and only for ASDU types 1-37.

Currently RTS can only be combined with the "CA Offset for RT messages" option with a CI Firmware Build number 703 with date 8.7.2015 (CI864HWLib x.x.88) or higher.

4.1.8.3 Choosing RTC or RTS

Below are given some points to consider when deciding to implement RTC and/or RTS.

- If a given configuration has run for a time without problems, then it is not recommended to implement either RTC or RTS ("Never touch a running system"). If the decision to implement RTC or RTS is made, it is recommended to implement them during a major revision or system upgrade when there is enough time to test the changes before the plant has to return to productive use.
- Neither RTC nor RTS can guarantee that the controller does not crash if bad data is received from the Partner. We have not been able to investigate this problem properly as the crashes occur only occasionally and often only in a setup that we could replicate in the lab.
- RTC and RTS require using structured variables on the Receive blocks. Both require the Receive blocks to be properly configured (the correct data type must be specified on the pin ***IECDataTypeToUse*** or the Receive blocks will not work properly.
- RTC and RTS have different requirements:
  RTS needs 800xA SV 5.1 or higher. It can be used with any data type or protocol.
  RTC can be used with both 800xA SV 5.0 or higher as long as the required version of the CI Firmware and IEC SW Libraries are used. RTS only applies to the data types for which it is configured.
- If both are available RTS is the recommended option unless one of the following points makes it impracticable.
- RTC can be enabled or disabled separately for each function block. For some function blocks that wrap Receive blocks it can be enabled by setting the Project Constant cIEC101.ReceiveTypeCheck.
- RTS is enabled by setting the Project Constant cIEC101.UseTypeOffset from 0 (the default) to 536870912. When RTS is enabled or disabled it is recommended to reset (clear the memory and download the new configuration to the empty controller) all affected controllers as the CI modules may crash when the changed configuration is downloaded online.
- RTS always applies to all controllers in a Project. When RTS is enabled the correct data type must be specified on all Receive blocks on the pin ***IECDataTypeToUse***. If RTS is disabled later on the pin ***IECDataTypeToUse*** must be set back to 0 (or empty as 0 is the default value).
- With a CI Firmware Build number 703 with date 8.7.2015 (CI864HWLib x.x.88) or higher RTS can be combined with the "CA Offset for RT messages" option. In this configuration use cIEC60870.SYS_MSG_CA_OFFSET_ASDU for non-RT Messages,
- cIEC60870.SYS_MSG_CA_OFFSET_RT for RT Messages and
- cIEC60870.SYS_MSG_CA_OFFSET_ASDU_RT for RT Messages as offset for FBs that automatically add the RTS offset based on the ASDU type.

4.2 Software configuration with IEC60870CommLib:

The basic function blocks to configure the communication and data transfer are in the library IEC60870CommLib.

This library contains the following function blocks:

Function Block	Description
IEC60870Connect	
IEC60870Receive	
IEC60870ReceiveSize	Receive (Read) data from an IEC partner.
IEC60870WriteCyc	Send (Write) data to an IEC partner.
IEC60870WriteCont	Send (Write) data to an IEC partner.
IEC60870WriteSize	Send (Write) data to an IEC partner.

It is recommended that the Connect function block is in the same task as the Read and Write function blocks. The behavior can be unpredictable if different function blocks are in different tasks with different priorities and two function blocks are executed at the same time.

IEC60870ReceiveSize and IEC60870WriteSize are designed to be used inside other function blocks with extensible parameters. Since the size (number of Extensible Parameters) must be defined before the size of the calling function block is defined, these blocks can be used by defining the maximum number of Extensible Parameters and then specifying the actual number when executed.

All examples are given in structured text, but function block diagram can be used as well.

4.2.1 IEC60870Connect

This function block establishes the connection to the IEC partner or a pair of redundant IEC partners. There must be one connect function block per IEC60870 Partner configured in the hardware configuration.

When the ***En_C*** pin is set to true, this function block sends information about the connection to the CI864 board (such as the conversion list) and enables the connection. The ***CIPos*** and ***IEC60870PartnerPos*** pins identify the IEC partner. If this is successful, the ***Valid*** pin is set to true. If not, the ***Error*** pin will be set to true for one cycle and the ***Status*** pin will indicate the reason for the error.

To try again, the ***En_C*** pin has to be cleared and set again. If the error code is -7103, the ***En_C*** pin should not be toggled off, this error is handled inside the Protocol Handler and the Connect block will succeed once the error condition has be cleared.

The variable connected to the ***Id*** pin identifies this connection and must be connected to all IEC60870Receive, IEC60870WriteCyc and IEC60870WriteCyc function blocks belonging to this partner.

4.2.2 IEC60870Receive

This function block receives data sent by an IEC partner. It does not actually request data from the IEC partner; it only returns data that the IEC partner sent on its own.

The ***Enable*** pin must be set to true and the ***Id*** pin must be connected to the Id variable from a valid IEC60870Connect function block for this function block to work.

The ***CommonAddress pin*** gives the common address for all data elements.

The ***InformationObjectAddress*** gives the information object address for the first data element. The information object address is increased by ***InfoObjAddr_StepSize*** for each following data element. If ***InfoObjAddr_StepSize*** is 0, the function block will fail, even if only one data element is read.

The ***IECDataTypeToUse*** pin specifies the IEC data type or how it is found.

If ***IECDataTypeToUse*** is 0, then structured variables have to be used and the actual data type is found in the first data element of the structure. All the data elements must be of the same type. Structured types for many IEC data types are defined in this library.

If ***IECDataTypeToUse*** is greater than 0, then the value specifies the IEC data type and simple variables have to be used. Only the most important piece of information is returned (usually the value).
If ***IECDataTypeToUse*** is greater than 0 and Receive Type Separation is used, structured types are used and this pin specifies the actual data type (only for data types that use RTS). This is only available with IECCommLib 2.0-29 and higher.
If ***IECDataTypeToUse*** is less than 0, then structured variables have to be used (similar to using the value 0). The type of the received data is checked and the data is only copied if the type of the received data is the negative of the value given. e.g. if ***IECDataTypeToUse*** is -30, then the received data is only copied if it is of type 30. This option is only available if the PH version 2.0/16 or higher is used (HW library 1.x.72 or higher on SV5.1 or higher).

The ***Valid*** pin indicates if the data elements contain valid data (or at least some of them).

The ***NewData*** pin becomes non-zero whenever the Receive block has new data. This can be used to optimize following logic so that is only executed when new data arrives.

The ***Status***, ***Error*** and ***Warning*** pin contain information about possible errors.

The variables that receive the data are connected to the ***Rd[x]*** pins.

4.2.3 IEC60870ReceiveSize

This function block receives data sent by an IEC partner. It is very similar to IEC60870Receive but adds a pin for the actual number of used extensible parameters.

The ***ActualSize*** pin defines how many of the defined extensible parameters are actually used. It must be less or equal to the defined number of parameters (maximum 32).

4.2.4 IEC60870WriteCyc

This function block sends data sent to an IEC partner.

The ***Enable*** pin must be set to true and the ***Id*** pin must be connected to the Id variable from a valid IEC60870Connect function block for this function block to work.

The ***CycleTime*** pin defines how often the data is sent to the CI864 board. It is only sent if it has actually changed. It is always sent once when the ***Enable*** pin becomes true.
In SV 5 the behavior of the ***CycleTime*** pin has changed: The default value of 0 seconds means that the data is sent every cycle.

The ***RetryTime*** pin specifies how long the function block waits after an error before it retries the operation.

The ***CommonAddress*** pin gives the common address for all data elements.

The ***InformationObjectAddress*** gives the information object address for the first data element. The information object address is increased by ***InfoObjAddr_StepSize*** for each following data element. If ***InfoObjAddr_StepSize*** is 0, the function block will fail, even if only one data element is written.

The ***IECDataTypeToUse*** pin specifies the IEC data type or how it is found.

If ***IECDataTypeToUse*** is 0, then structured variables have to be used and the actual data type is found in the first data element of the structure. All the data elements must be of the same type. Structured types for many IEC data types are defined in this library.

If ***IECDataTypeToUse*** is greater than 0, then the value specifies the IEC data type and simple variables have to be used. Only the most important piece of information can be specified (usually the value).

If ***IECDataTypeToUse*** has the value TYPEID_IEC101_USE_DEFAULT_STRUCT, then a structured variable has to be connected to the first pin. This variable defines the actual data type and default values for the all data fields except the second field, but is not an actual data element. The data elements are defined by simple variable connected to the following pins, same as when this pin has a value > 0.

The ***ChangedValues*** pin is used to indicate if any values changed. If this pin is 0, the write block will not send the data to the CI864 board. The data is always sent once after the ***Enable*** pin becomes true. If this pin is non-zero, the write function block will send the data after checking it for changes. The main use of this pin is to reduce the controller load if the check for changed data is already performed by a preprocessing function block (See IEC60870SlaveLib).

Setting the ***ChangedValues*** pin to nonzero will not force the data to be sent to the IEC parner, it will only ensure that the data is sent to the protocol handler. The protocol handler will only send the data to the IEC partner if it has actually changed. If the data must be sent to the IEC partner (e.g. Counter values that have to be sent at regular intervals even if they have not changed since the last interval), resending of the data can be forced by disabling the function block and enabling it again.

The ***Valid*** pin indicates if the data elements contain valid data (or at least some of them).

The ***Status***, ***Error*** and ***Warning*** pin contain information about possible errors.

The variables with the data to be sent are connected to the ***Sd[x]*** pins.

4.2.5 IEC60870WriteCont

This function block sends data sent to an IEC partner.

This block is a simplified version of the IEC60870WriteCyc block. It behaves exactly the same as this block when the ***CycleTime*** pin is set to 0, but uses a bit less memory and execution time. Use this block if the data should be (potentially) sent every cycle, especially when used together with the Encode blocks that already check for data changes and use the ***ChangedValues*** pin.

4.2.6 IEC60870WriteSize

This function block sends data to an IEC partner. It is very similar to IEC60870WriteCont but adds a pin for the actual number of used extensible parameters.

The ***ActualSize*** pin defines how many of the defined extensible parameters are actually used. It must be less or equal to the defined number of parameters (maximum 32).

4.3 Example

ASDU type 30, M_SP_TB_01

Name	Data Type	Attributes	Initial Value
TypeIdent	dint	constant	cIEC60870.TYPEID_IEC101_030_M_SP_TB (=30)
COT	dword	retain	
Value	bool	retain	
Status	dword	retain	
TimeStamp	date_and_time	retain	

The fields are used as follows:

TypeIdent identifies the ASDU type and is constant.

COT is the Cause Of Transmission. This is normally 0, in which case the the CI864 fills in the normally used values.

Value contains the current value (binary state in this case).

Status contains the state of the measurement (valid, disturbed, overflow). In the usual case the CI864 translates the status from the OPC values used inside the AC800M to the IEC representa-tion.

TimeStamp contains the time of the latest change. When sending data ***TimeStamp*** can be left empty; in this case the CI864 uses the current time.

Other measurement ASDU types look very similar.

ASDU type 50, C_SE_NC_01

Name	Data Type	Attributes	Initial Value
TypeIdent	dint	constant	cIEC60870.TYPEID_IEC101_050_C_SE_NC (=50)
COT	dword	retain	
Qualifyer	word	retain	
Value	real	retain	

The fields are used as follows:

TypeIdent identifies the ASDU type and is constant.

COT (Cause Of Transmission) contains the reason the telegram is sent (command activation, acknowledge, negative acknowledge, command termination).

Qualifyer contains additional information for the command (select or execute, pulse duration).

Value contains the setpoint value or the command direction (on or off, higher or lower).

Other command ASDU types look very similar.

4.3.1.1 Variables

Name	Data Type	Attributes	Initial Value
Enable_Comm	bool	retain	true
Id	Comm_Channel	retain	
Connect1Valid	bool	retain	
Connect1Status	dint	retain	
Connect1Error	bool	retain	
EnableData	bool	retain	
M_001_M_SP_NA	MSG_IEC101_001_M_SP_NA_Type	retain	
M_003_M_DP_NA	MSG_IEC101_003_M_DP_NA_Type	retain	
M_013_M_ME_NC	MSG_IEC101_013_M_ME_NC_Type	retain	
M_058_C_SC_TA	MSG_IEC101_058_C_SC_TA_Type	retain	
M_058_C_SC_TA_C	MSG_IEC101_058_C_SC_TA_Type	retain	
M_058_C_SC_TA_V	bool	retain	false
M_058_C_SC_TA_E	bool	retain	false
M_058_C_SC_TA_N	dint	retain	0
BoolIO1	BoolIO		
BoolIOOn	BoolIO		
BoolIOOff	BoolIO		
RealIO1	RealIO		

4.3.1.2 Function blocks

Name	Function Block Type	Task Connec-tion	Description
Connect1	IEC60870Connect		
Write001	IEC60870WriteCyc[2]		
Write003	IEC60870WriteCyc[1]		
Write013	IEC60870WriteCyc[1]		
Read058	IEC60870Receive[1]		
Write058	IEC60870WriteCyc[1]		

4.3.1.3 Code

```
(* Connect to IEC task / partner *)

Connect1( En_C := Enable_Comm,
          CIPos := 1,
          IEC60870PartnerPos := 1,
          Valid => Connect1Valid,
          Error => Connect1Error,
          Status => Connect1Status,
          Id := Id );

(* Copy data from IO variables to IEC variables *)

M_001_M_SP_NA.Value := BoolIO1.Value;
M_001_M_SP_NA.Status := BoolIO1.Status;

M_003_M_DP_NA.Value := 0;
IF (BoolIOOn.Value) THEN M_003_M_DP_NA.Value := M_003_M_DP_NA.Value + 2; END_IF;
IF (BoolIOOff.Value) THEN M_003_M_DP_NA.Value := M_003_M_DP_NA.Value + 1; END_IF;
M_003_M_DP_NA.Status := BoolIOOn.Status;

M_013_M_ME_NC.Value := RealIO1.Value;
M_013_M_ME_NC.Status := RealIO1.Status;

(* Write data to IEC partner *)

Write001( Enable := EnableData,
          Id := Id,
          CycleTime := CycleTime,
          CommonAddress := 1,
          InformationObjectAddr := 1,
          InfoObjAddr_StepSize := 256,
          Sd[1] := M_001_M_SP_NA,
          Sd[2] := M_001_M_SP_NA2 );

Write003( Enable := EnableData,
          Id := Id,
          CycleTime := CycleTime,
          CommonAddress := 1,
          InformationObjectAddr := 3,
          Sd[1] := M_003_M_DP_NA );

Write013( Enable := EnableData,
          Id := Id,
          CycleTime := CycleTime,
          CommonAddress := 1,
          InformationObjectAddr := 13,
          Sd[1] := M_013_M_ME_NC );

(* Read a command, prepare and send a reply *)

Read058( Enable := EnableData,
         Id := Id,
         CommonAddress := 1,
         InformationObjectAddr := 58,
         Valid => M_058_C_SC_TA_V,
         Rd[1] := M_058_C_SC_TA );

IF ((M_058_C_SC_TA_V) AND (M_058_C_SC_TA.COT=6)) THEN
   M_058_C_SC_TA_C.Qualifier := M_058_C_SC_TA.Qualifier;
   M_058_C_SC_TA_C.Value := M_058_C_SC_TA.Value;
   IF (NOT M_058_C_SC_TA_E) THEN
      M_058_C_SC_TA_N := 0;
   END_IF;
   M_058_C_SC_TA_N := M_058_C_SC_TA_N + 1;
   IF (M_058_C_SC_TA_N<10) THEN
      M_058_C_SC_TA_C.COT := 7;
   ELSE
      M_058_C_SC_TA_C.COT := 10;
   END_IF;
   M_058_C_SC_TA_E := TRUE;
ELSE
   M_058_C_SC_TA_E := FALSE;
END_IF;

Write058( Enable := M_058_C_SC_TA_E,
          Id := Id,
          CycleTime := CycleTime,
          CommonAddress := 1,
          InformationObjectAddr := 58,
          Sd[1] := M_058_C_SC_TA_C );
```

4.4 Software configuration with the support libraries:

In SV 4 only a single support library was used, IEC60870SupLib, but in SV 5 this library has been split into 3 seperate libraries, IEC60870ExtLib, IEC60870MasterLib and IEC60870SlaveLib. Most of the function blocks are similar, but some of them have been changed.

IEC60870ExtLib contains general functions, IEC60870MasterLib contains functions for implementing a master or controlling station, and IEC60870SlaveLib contains functions for implementing a slave or controlled station. The two master and slave functions are separated in two libraries, but there is no restriction in the IEC code so that both of them can be used together to implement both master and slave functions for a single IEC partner.

Usually IEC60870ExtLib is used together with one of the other two libraries.

4.5 Software configuration with IEC60870ExtLib:

This library contains function blocks that build on the function blocks from IEC60870CommLib to offer generally useful functionality. All the following function blocks could be implemented in the application by hand, but it usually makes sense to use the provided ones.

This library contains the following function blocks:

Function Block	Description
ConnectIEC60870	Connect to an IEC partner.
ConnectionIEC60870	Connect to an IEC partner.
ConnectionIEC60870_DS	Connect to an IEC partner in a Data Sharing group.
DataDelay	Set 4 different Enable variable one after the other.
DataDelayMulti	Set several different Enable variables one after the other.
PartnerStatus	Report the partner status. Report if the connection to the IEC partner is valid.
HoldStatus	Freezes the connection status during application download.
SetPartnerActive	Set an IEC task to active or standby.
Redundancy	Redundancy switchover for 2 CI boards with single connections.
RedundancyDual	Redundancy switchover for 2 CI boards with double connections.
RedundancyDual3	Redundancy switchover for 2 CI boards with 2+1 connections.
RedundancyLineShare	Redundancy switchover for 2 CI boards as master on a shared line.
StationStatus	Reports the status of a 4 stations.
RcvClockSync	Receive and process Time Synchronization telegrams.
	Address conversion functions:
Conv_IEC_CA	Calculate the common address from its components.
Conv_IEC_IOA	Calculate the information object address from its components.
Conv_IEC_IOA_Multi	Calculate the multiple information object addresses from their components.
Conv_SAT_to_IEC_CA	Calculate the common address from its components. (SAT terminology)
Conv_SAT_to_IEC_IOA	Calculate the information object address from its components. (SAT terminology)
Decode_DP2	Decode an IEC DP value into two bool variables.
Decode_DP3	Decode an IEC DP value into three bool variables.
BitCountIEC	Counts the number of set bits, used internally.
ChangeLimiter	Limit the number of status changes per task cycle.
SetDebugOut	Sets the Debug output bits.
	Transparent Data channel functions:
TransData16Rcv	Transfer transparent data (receiving part).
TransData16Write	Transfer transparent data (sending part).

4.5.1 ConnectIEC60870

This function block establishes the connection to the IEC partner or a pair of redundant IEC partners. There must be one connect function block per IEC60870 Partner configured in the hardware configuration.

This function block encapsules an **IEC60870Connect** function block and behaves very much the same.

The main difference is that it tries to reconnect the IEC task if the Connect block cannot connect to it (e.g. because the CI864 module has crashed or is currently removed) as long as the Enable pin is set to true.

If the connection is not valid after the time given in ***ConnectWait***, it disables the Enable pin of the embedded IEC60870Connect block for ***Restart_Delay*** before it enables it again.

This allows the Connect block to handle hot-swapping and severe error situations that crash the CI864 board automatically.

4.5.2 ConnectionIEC60870

This function block establishes the connection to the IEC partner or a pair of redundant IEC partners. There must be one connect function block per IEC60870 Partner configured in the hardware configuration.

This function block contains most of the function blocks handling the connections to one IEC partner through one CI864 board: One **ConnectIEC60870** block, one **PartnerStatus** block, one **SetPartnerActive** block and the logic to tie them together. If this function block is used, then the other blocks listed above should not be used.

Most of the input/output pins are connected directly to the wrapped function blocks; see the description of those function blocks.

Enable enables the connection; this is connected to the ***En_C*** pin of the **IEC60870Connect** function block.

The ***Redundant*** pin determines if the IEC partner is single or redundant (both primary and secondary IP address used). When this function block is used, redundancy switchover between the two connections is always handled automatically; this block cannot be used if the switchover has to be controlled by logic.

The ***CIRedundancy*** pin determines if one CI864 board is used or if two CI864 boards are used as a redundant pair. If this pin is TRUE, then there needs to be a second block of this type to handle the redundant connection plus logic to handle the switchover (typically a Redundancy or RedundancyDual function block). Also the connection (or both connections) is set active based on the ***Primary*** pin.

The ***V2_DecodeUsed*** pin determines if "V2" style Decode blocks are used or not. See the discussion of Data Consistency in the section on Redundancy.

The ***ColdStartDelay*** pin is used to delay activating the connection after a controller start. Internally the ***Enable*** pin will be considered FALSE until this time has elapsed. This can be used to reduce the peak load imposed on the controller after startup by delaying the activation of the connection.

The ***ActivateDelay*** pin is used to delay the ***Primary*** signal, so the partners of both CI modules of a redundant connection cannot be active at the same time.

The ***CommandDelay*** pin is used to delay setting the ***CmdEnable*** pin to TRUE.

The ***Primary*** pin pin determines if this connection is the primary connection or not; only relevant if the ***CIRedundancy*** pin is TRUE.

The ***CI_Valid*** pin indicates if the CI864 board functions properly; this is connected to the ***Valid*** pin of the **IEC60870Connect** function block.

The ***CmdEnable*** pin has the same function as the corresponding pins of the **Redundancy** or **RedundancyDual** block. When this function block is used, this pin should be used and the pins of the **Redundancy** block should be ignored.

The ***InEnable*** pin indicates if the **Receive** blocks for this connection can be enabled. If ***CIRedundancy*** is FALSE or ***V2_DecodeUsed*** is TRUE, then this can be used directly, otherwise additional logic is used; See the discussion of Data Consistency in the section on Redundancy.

The ***DecodeConnValid*** pin indicates that the connection is considered valid. This is connected to the ***ConnValid*** pin of a **Decode_xxx_v2** function block. If redundant CI864 boards are used, the outputs of the two function blocks are or-ed together.

4.5.3 ConnectionIEC60870_DS

This function block establishes the connection to the IEC partner or a pair of redundant IEC partners. There must be one connect function block per IEC60870 Partner configured in the hardware configuration.

This function block is similar to the **ConnectionIEC60870** block, but it is used for Data Sharing groups. (The _DS stands for Data Sharing)

The first/primary connection uses a **ConnectionIEC60870** block, the further connections use the **ConnectionIEC60870_DS** block.

Most of the input/output pins are the same as for the **ConnectionIEC60870** block, see above.

The ***Id_DSM*** pin (DSM stands for Data Sharing Master) is connected to the Id variable of the first/primary connection in the Data Sharing group.

4.5.4 DataDelay

This function block is used to set four different 'enable' variables one after another. For each 'enable' variable there is another variable that becomes true for one ***DataInitTime***.

If the ***Enable*** pin is false, then all output pins are false as well. ***DataDelay*** time after the ***Enable*** pin becomes true, ***DataEnable1*** is set to true and ***DataEnable1New*** is set to true for ***DataInitTime***. ***DataDelay*** time after that ***DataEnable2*** is set to true as well, and so on.

This function block is useful if a lot of Write function blocks are used to split them into groups and to avoid causing a very high CPU load by enabling them all at the same time. For new projects it is suggested to use the ***DataDelayMulti*** (see below).

4.5.5 DataDelayMulti

This function block is used to set a number of different 'enable' variables one after another. This function block is similar to **DataDelay**, but it has a varying number of outputs and is more efficient.

If the ***Enable*** pin is false, then all output pins are false as well.

DataDelayMS milliseconds after the ***Enable*** pin becomes true, ***EnableOut[1]*** is set to true and ***UpdateOut[1]*** is set to true for ***DataInitMS*** milliseconds. ***DataDelayMS*** milliseconds after that the next group of variables is set.

On a rising edge of ***Update[x]***, ***UpdateOut[x]*** is set to true for one cycle, provided that ***EnableOut[x]*** is also TRUE. This can be used to force a "Data Refresh".

This function block is useful if a lot of Write function blocks are used to split them into groups and to avoid causing a very high CPU load by enabling them all at the same time.

4.5.6 PartnerStatus

This function block returns the status of the IEC partner; that is if the CI864 board has a valid connection to the IEC partner.

The ***Enable*** pin must be set to true and the ***Id*** pin must be connected to the Id variable from a valid **IEC60870Connect** function block for this function block to work. The Id variable must be the same one as the one from the Connect block with the same ***IEC60870PartnerPos*** or the Id variable of the first Connect block in a data sharing group.

If the ***Redundant*** pin is true, then this function block returns information about both IEC partners (primary and secondary IP Address is configured for this IEC partner), if not it only returns information about the first partner.

Status1 and ***Status2*** indicate if a valid connection exists to the primary / secondary IEC partner.

StatusValue1 and ***StatusValue2*** give a little more detailed information: 0 if there is no network connection, 1 if the TCP/IP connection has been established and 2 if the connection is fully enabled. If an End-Of-Initialisation telegram is used, the value becomes 3 after the EOI telegram has been received.

ReadValid1 and ***ReadValid2*** are true when the corresponding status has actually been received from the CI board.

4.5.7 HoldStatus

This function block is used to preserve the connection status during an application download, because the Connect block becomes invalid for a few task cycles until it re-establishes the connection to the protocol handler. This short interruption can be problematic if it causes a line switchover or connection error messages in the event log. This function block freezes the status for a given time or until the **PartnerStatus** block reports that it is reporting reliable information.

The ***ConnValid*** pin should be connected to the ***Valid*** pin of the Connect block.

The ***Status1In***, ***Status2In***, ***StatusValue1In***, ***StatusValue2In***, ***ReadValid1In***, and ***ReadValid2In*** pins should be connected to the corresponding pins of the PartnerStatus block.

The ***HoldTime*** pin determines how long the status is frozen.

The output pins ***Status1Out***, ***Status2Out***, ***StatusValue1Out***, and ***StatusValue2Out*** are then used just like the corresponding pins from the **PartnerStatus** block.

4.5.8 SetPartnerActive

In simple cases the (possibly) two connections an IEC Partner can have will perform redundancy switchover, but in more complicated redundancy situations, this function block is used to force each connection to be either active or standby.

The ***Enable*** pin must be set to true (usually the output of the ***Valid*** pin is used) and the ***Id*** pin must be connected to the Id variable from a valid **IEC60870Connect** function block for this function block to work. The Id variable must be the same one as the one from the Connect block with the same ***IEC60870PartnerPos*** or the Id variable of the first Connect block in a data sharing group.

If the ***Redundant*** pin is true, then this function block sets both IEC partners (primary and secondary IP Address is configured for this IEC partner), if not it only sets the first partner.

If the ***Automatic*** pin is true, then the IEC partner connections determine active / standby automatically as if this function block was not used.

The ***Partner1Active*** and ***Partner2Active*** pins are used to set the two connections of that partner to active (true) or standby (false).

4.5.9 Redundancy

This function block implements logic to perform redundancy switchover between two connections, usually two connections on two different CI boards. This function block implements one possible algorithm for determining which connection is active, others are certainly possible.

The basic logic is as follows: The first connection that becomes valid will be the active one. A switchover to the standby connection occurs if the currently active connection becomes invalid and the standby connection is valid.

The pins ***ConnA1Ok*** and ***ConnB1Ok*** indicate if the two connections are working properly. Typically they are directly connected to the ***Status*X** pins of a **PartnerStatus** block.

A rising flank on the ***LineChange*** pin forces a switchover.

The pin ***SwitchoverDelay*** gives the time before switchover occurs. This is useful to avoid switchover if a short connection break occurs (e.g. during application download).

After switchover the currently active connection is set passive. After an extra delay of ***ActivateDelay***, the other connection is set active. ***CommandDelay*** is an extra delay on top of that before the ***ConnXCmdOut*** pins become active.

The pins ***ConnAPrimary*** and ***ConnBPrimary*** indicate which connection is active. During switchover both pins can be false for a short while (***ActivateDelay***) to avoid situations that both connections are treated as active at the same time.

The pins ***ConnACmdOut*** and ***ConnBCmdOut*** can be used for enabling commands only after a short delay after switchover has occurred.

4.5.10 RedundancyDual

This function block is similar to **Redundancy**, except that it is used for two pairs of redundant connections.

The switchover logic is similar, except that switchover happens if the passive pair of connections has more valid connections than the active pair. E.g. if one connection of the active pair fails while both passive connections are working, then switchover will occur.

The pins ***ConnA1Ok*** and ***ConnA2Ok*** indicate if the two connections of the first pair are working, ***ConnB1Ok*** and ***ConnB2Ok*** indicate the status of the second pair.

4.5.11 RedundancyDual3

This function block is similar to **RedundancyDual**, except that it is used for two pairs of three connections (usually one partner with two connections and one with a single connection).

The switchover logic is similar, except that switchover happens if the passive pair of connections has more valid connections than the active pair. E.g. if one connection of the active pair fails while both passive connections are working, then switchover will occur.

The pins ***ConnA1Ok***, ***ConnA2Ok*** and ***ConnA3Ok*** indicate if the two connections of the first triple are working, ***ConnB1Ok***, ***ConnB2Ok*** and ***ConnB3Ok*** indicate the status of the second triple

4.5.12 StationStatus

This function block reports the status of 4 stations. This block is normally used only in the master station in a party-line configuration (when using the -101 or -103 Master). In such a configuration the PartnerStatus block will return a valid status as long as at least one station is responding, while this block reports the status of each station.

The ***Enable*** pin must be set to true and the ***Id*** pin must be connected to the Id variable from a valid **IEC60870Connect** function block for this function block to work. The Id variable must be the same one as the one from the Connect block with the same ***IEC60870PartnerPos*** or the Id variable of the first Connect block in a data sharing group.

If the ***Redundant*** pin is true, then this function block returns information about the second Partner, if not it returns information about the first partner.

The ***FirstStation*** pin gives an offset to the station number. If this pin is 0, then the function block reports the status of stations 1 to 4, if it is 4, then the block reports the status of stations 5 to 8 and so on.

Status01, ***Status02***, ***Status03*** and ***Status04*** indicate if a valid connection exists to the stations.

StatusValue01, ***StatusValue02***, ***StatusValue03*** and ***StatusValue4*** give a little more detailed information: 0 if there is no network connection, 1 if the TCP/IP connection has been established and 2 if the connection is fully enabled. If an End-Of-Initialization telegram is used, the value becomes 3 after the EOI telegram has been received.

If less than 4 stations are connected, simply ignore the extraneous outputs.

4.5.13 RcvClockSync

Receive and process Time Synchronization telegrams. Use this function block in a substation that should have its time synchronized using the IEC Time Set telegrams. Normally other time synchronization methods (such as SNTP) are preferable as the accuracy of this method is relatively low. Under optimal conditions the accuracy is about 50 milliseconds, but in the worst case the time can be off by several seconds as there is no possibility to detect (or compensate for) transmission delays or retransmissions on the TCP/IP connection.

The ***Enable*** pin must be set to true and the ***Id*** pin must be connected to the Id variable from a valid **IEC60870Connect** function block for this function block to work. The Id variable must be the same one as the one from the Connect block with the same ***IEC60870PartnerPos*** or the Id variable of the first Connect block in a data sharing group.

The ***CommonAddress pin*** gives the common address to which the Time Set telegram is sent, in this case usually the common address of the substation.

If ***EnableTimeSet*** is true, then this function block will actually change the controller time. If this is false, then received Time Set telegrams are still processed and the time difference reported, but the controller time is not modified.

EnableTimeSetDelay gives the number of task cycles after enabling the function block during which Time Set telegrams are not processed. This is used to ignore "old" Time Set telegrams that were received before the function block was enabled and buffered by the protocol handler or the CI board.

Transmission Delay gives the estimated average time the Time Set telegrams spend in "transit". This time is compensated for when calculating the time difference between the sender and the receiver.

SetTimeDiffMS gives the minimum adjustment to the controller time in milliseconds. This is used to prevent the controller time to be set too often. Since the random delays in transmitting and processing the telegram cannot be detected or compensated for, using a value that is too low will cause the controller time to be adjusted back and forth very often.

SinceLastTele gives the time since the last Time Set telegram was received.

TimeReceived is set to true for one task cycle when a new Time Set telegram is received.

LastTimeDiff gives the time difference between the time from the latest Time Set telegram and the controller time.

TimeChanged is set to true for one task cycle when the controller time is actually adjusted.

LastTimeChange gives the amount of time that the controller clock was adjusted.

4.5.14 Address conversion functions

All these functions have input pins for the two (Common Address) or three (Information Object Address) address bytes and an output pin that outputs the calculated address. While these functions add execution time to the IEC program, they make the used addresses much more readable if the IEC partner is configured using a structured address.

The two functions **Conv_IEC_CA** and **Conv_SAT_to_IEC_CA** perform exactly the same function, only their input pins are named differently. They convert the two bytes of a structured Common Address into the single value of an unstructured Common Address. The function block **Conv_IEC_CA** has input pins called ***CA1*** and ***CA2*** while the function block **Conv_SAT_to_IEC_CA** has ***QRGN*** and ***QKOMP***. The output pin is called ***IEC_Addr***.

The function block **Conv_IEC_IOA** has the input pins ***IOA1***, ***IOA2*** and ***IOA3*** and the output pin ***IEC_Addr***.

The function block **Conv_SAT_to_IEC_IOA** is similar to **Conv_IEC_IOA**, but the input pins are called ***QBG***, ***QNEW*** and ***QSUB***. It uses a bool external variable called ***IEC60870_SAT_Addressing***. If this variable is true, the parameters are used as ***IOA1***, ***IOA2*** and ***IOA3*** in the given order, if it is false, ***QBG*** and ***QNEW*** are swapped.

The function block **Conv_IEC_IOA_Multi** works the same as the one for a single address, except that it calculates several addresses at once.

4.5.15 Decode_DP2

Decode an IEC DP value into two bool variables.

The outputs are set as follows:

Value	OnValue	OffValue
0 (Indeterminate state)	FALSE	FALSE
1 (Off state)	FALSE	TRUE
2 (On state)	TRUE	FALSE
3 (Indeterminate state; error)	TRUE	TRUE

4.5.16 Decode_DP3

Decode an IEC DP value into three bool variables.

The outputs are set as follows:

Value	OnValue	OffValue	ErrorValue
0 (Indeterminate state)	FALSE	FALSE	FALSE
1 (Off state)	FALSE	TRUE	FALSE
2 (On state)	TRUE	FALSE	FALSE
3 (Indeterminate state; error)	FALSE	FALSE	TRUE

4.5.17 BitCountIEC

Counts the number of set bits; used internally.

4.5.18 ChangeLimiter

Limits the number of status changes per task cycle. This function block can be used spread to extra load of enabling / disabling Receive, Decode or Write function blocks and the associated functionality (e.g. logging of the change of signal quality) by delaying the status change so only a limited number of status changes is acted upon each task cycle. This is especially useful when one controller handles many IEC connections. When the total number of signals is very high (especially received signals) this can also be used to split the signals into groups which are enabled one after the other. For sent signals the **DataDelay** function block is usually used for a similar purpose.

Connect the ***In*** pin to the signal to be delayed (e.g. the output of the PartnerStatus block).

The ***Valid*** pin of the **IEC60870Connect** function block becomes FALSE for a short while after a changed application is downloaded; the same happens to function block that wrap or depend on this block. The ***AppStartFreeze*** pin defines a "grace period" during which status changes of these pins a processed directly, but only if the ***In*** pin was TRUE before the application download.

The ***MaxChanges*** pin gives the maximum number of changes per period (usually a task cycle).

The ***ChangeCounter*** pin must be connected to a (global) variable. All the **ChangeLimiter** function blocks that share the same variable work together as a group. This variable needs to be set to 0 by outside logic; setting it to 0 starts a new period.

4.5.19 SetDebugOut

This function block sets the debug control bits. Debugging must be enabled for this to have any effect (see HW Parameters "Debug IP Address" and "Debug Port") and a tool to log the messages must be running (e.g. UDPDebug). Debugging can add significantly to the CPU load of the CI864 and slow data transfer, so it should only be enabled when needed. Set the debug bits to 0 to minimize the overhead of debugging. There are separate debugging bits for the CI864 as a whole, each IEC Partner (Logical Partner) and each Connection (Partner). Each set bit enabled debugging for a certain function.

Usage of this function requires detailed knowledge about the internal structure of the CI864 and/or the used communication protocol (e.g. IEC 60870-5-104). Therefore it is usually only used by (or at the behest of) a developer from ABB Austria.

The ***Enable*** pin must be set to true (usually the output of the ***Valid*** pin is used) and the ***Id*** pin must be connected to the Id variable from a valid **IEC60870Connect** function block for this function block to work. The Id variable must be the same one as the one from the Connect block with the same ***IEC60870PartnerPos*** or the Id variable of the first Connect block in a data sharing group.

IEC60870PartnerPos and ***Partner*** identify which debug bits are set: If both are 0, the debug bits of the CI864 as a whole are set; if ***IEC60870PartnerPos*** is greater 0 and ***Partner*** is 0 then the bits of the IEC Partner (Logical Partner) are set; if ***IEC60870PartnerPos*** is greater 0 and if ***Partner*** is 1 or 2 the the bits of that Connection (Primary and Secondary respectively) are set.

DebugBits gives the value to which to set the debug bits.

Currently debug bits are only used for the Connection.

CI864: (***IEC60870PartnerPos***=0 Partner=0)

Bitmask	Option Name	Description

Logical Partner: (***IEC60870PartnerPos***>0 Partner=0)

Bitmask	Option Name	Description

CI864: (***IEC60870PartnerPos***>0 Partner=1 or 2)

The bits as given here apply to the IEC60870-5-104 Protocol. The exact usage of these bits depends on the used protocol.

Bitmask	Option Name	Description
0x000000FF	DB_PARTNER_MASK	Bits allocated to the Partner layer.
0x00000001	DB_PARTNER_READ	Telegrams read from partner (without flow-level data).
0x00000002	DB_PARTNER_WRITE	Telegrams written to partner (without flow-level data).
0x00000004	DB_PARTNER_STATUS	Currently not used.
0x00000008	DB_PARTNER_CMD	Details about command handling (the CI864 uses extra code and data structures to associate command activation messages with the response messages).
0x00000080	DB_PARTNER_POLL	Detailed information about polling telegrams generated at the partner level. Currently used only by I35 protocol.
0x0000FF00	DB_MSG_MASK	Bits allocated to the Message layer.
0x00000100	DB_MSG_READ	Messages received from the partner.
0x00000200	DB_MSG_WRITE	Currently not used.
0x00000400	DB_MSG_STATUS	Currently not used.
0x00000800	DB_MSG_ANALYZE	Detailed information about message processing.
0x00001000	DB_MSG_DEC_HDR	Detailed information about message header decoding. Currently used only by the Allen-Bradley protocol.
0x00002000	DB_MSG_DEC_INFOBJ	Detailed information about message item decoding. Currently used only by the Allen-Bradley protocol.
0x00008000	DB_MSG_POLL	Detailed information about data item polling messages. Currently used only by the I35 protocol.
0x00F00000	DB_FLOW_MASK	Bits allocated to the Flow layer.
0x00100000	DB_FLOW_READ	Complete telegrams (without the telegram frame data) read from the partner.
0x00200000	DB_FLOW_WRITE	Complete telegrams (without the telegram frame data) written to the partner.
0x00400000	DB_FLOW_STATUS	Currently not used.
0x00800000	DB_FLOW_POLL	Detailed information about the telegam flow control (e.g. polling telegrams, ...)
0x0F000000	DB_FRAME_MASK	Bits allocated to the Frame layer.
0x01000000	DB_FRAME_READ	Complete telegrams (including telegram frame data) read from the partner. This is after frame-level integrity checks on the received data were performed, so partial or corrupted data is not shown.
0x02000000	DB_FRAME_WRITE	Complete telegrams (including telegram frame data) written to the partner.
0x04000000	DB_FRAME_STATUS	Currently not used.
0x08000000	DB_FRAME_EN_DE_CODE	Detailed information about the telegram frame encoding / decoding. Currently used only by Allen-Bradley X328 framing.
0xF0000000	DB_TRANS_MASK	Bits allocated to the Transport layer. This layer is generally the same for all protocols.
0x10000000	DB_TRANS_READ	Raw data read. At this layer a telegram from the partner is usually read as several chunks of bytes.
0x20000000	DB_TRANS_WRITE	Raw data written.

0x40000000	DB_TRANS_STATUS	Status changes of Transport layer (TCP/IP connect / disconnect, ...)

4.5.20 Transparent Data channel functions:

A Transparent Data channel is a virtual serial data connection between two IEC Partners. The two IEC Partners are usually on two different CI864 boards connected to the same controller, but it is also possible to use CI864 boards connected to different controllers and to send the transparent data from one controller to the other with MMS or other means.

Each Transparent Data channel needs 4 of the following function blocks, one Rcv block and one Write block to send data from the first IEC Partner to the second and a second such pair to send data in the opposite direction. The output of the Rcv block is connected to the corresponding Write block.

In a fully redundant configuration there are 4 channels with a total of 8 Rcv and 8 Write blocks.

At the moment Transparent Data is used only for SAT style disturbance transmission, but if needed it can be adapted to other uses, such as IEC 60870 style file transfer.

4.5.20.1 TransData16Rcv

This function block reads the data of a Transparent Data channel of width 16 from an IEC Partner and writes it to a structured variable of type TransData16Chan.

The ***Ena*** pin must be set to true and the ***Id*** pin must be connected to the Id variable from a valid **IEC60870Connect** function block for this function block to work. The Id variable must be the same one as the one from the Connect block with the same ***IEC60870PartnerPos*** or the Id variable of the first Connect block in a data sharing group.

Channel gives the number of the data channel to be used. Normally 0 is used for the first channel and 1 for the second channel (in a redundant configuration). This channel number must match the one given in the HW setting ***DisTransChan1/2***.

ExecCycle determines how often (every how many task cycles) this function block is actually executed. The TransData16Rcv and the corresponding Write block are usually placed in different 1131 programs. In some configurations these programs are executed in different tasks with significantly different execution cycles. It does not make much sense to execute the two blocks at significantly different intervals, since the slower task determines the maximum speed of the channel. Executing one of the two blocks more often than needed does not cause any problems as such, but it can increase the CPU load unnessesarily.

ExecOffs gives an offset within the cycle, when exactly the block is executed. This can be used to spread the CPU load over different task cycles. This parameter must be less than ***ExecCycle***.

The structured variable of type TransData16Chan that receives the data read from the IEC Partner is connected to the ***TCData*** pin.

4.5.20.2 TransData16Write

This function block takes the data from a structured variable of type TransData16Chan and writes it to a Transparent Data channel of width 16 of an IEC Partner.

The connections are the same as for the Rcv block above, except that the ***TCData*** pin is (logically) an input rather than an output pin.

4.6 Software configuration with IEC60870SlaveLib:

This library contains function blocks that build on the function blocks from IEC60870CommLib and IEC60870ExtLib to implement data handling functions for a slave station; that is a station that sends monitoring messages and receives commands and setpoints. All the following function blocks could be implemented in the application by hand, but it usually makes sense to use the provided ones.

This library contains the following function blocks:

Function Block	Description
MakeBoolIO	Convert simple values to a BoolIO structure.
MakeRealIO	Convert simple values to a RealIO structure.
	Functions to convert I/O types to IEC types:
Encode_030_SP_TB	Convert BoolIO to single point (IEC ASDU type 30).
Encode_031_DP_TB	Convert two BoolIO to double point (IEC ASDU type 31).
Encode_036_ME_TF	Convert RealIO to measured value (IEC ASDU type 36).
	Simplified functions to convert I/O types to IEC types:
Encode_015_IT_NA_Basic	Convert dint to integrated total (IEC ASDU type 15).
Encode_030_SP_TB_Basic	Convert BoolIO to single point (IEC ASDU type 30).
Encode_031_DP_TB_Basic	Convert two BoolIO to double point (IEC ASDU type 31).
Encode_036_ME_TF_Basic	Convert RealIO to measured value (IEC ASDU type 36).
Encode_037_IT_TB_Basic	Convert dint to integrated totals (IEC ASDU type 37).
	Simplified functions to convert simple types to IEC types:
Encode_009_ME_NA_NBIO	Convert real to measured value (IEC ASDU type 9).
Encode_011_ME_NB_NBIO	Convert real to measured value (IEC ASDU type 11).
Encode_030_SP_TB_NBIO	Convert bool to single point (IEC ASDU type 30).
Encode_031_DP_TB_NBIO	Convert two bool to double point (IEC ASDU type 31).
Encode_032_ST_TB_NBIO	Convert dint to step position (IEC ASDU type 32).
Encode_033_BO_TB_NBIO	Convert dword to binary output (IEC ASDU type 33).
Encode_034_ME_TD_NBIO	Convert real to measured value (IEC ASDU type 34).
Encode_036_ME_TF_NBIO	Convert real to measured value (IEC ASDU type 36).
	Combined Output functions for Monitoring data types (inputs are I/O types):
EncWrite_001_030_SP_NA_TB	Encode and send single point values (IEC ASDU type 1 or 30).
EncWrite_001_030_SP_NA_TB_32	Encode and send single point values (IEC ASDU type 1 or 30).
EncWrite_003_031_DP_NA_TB	Encode and send double point values (IEC ASDU type 3 or 31).
EncWrite_003_031_DP_NA_TB_SI	Encode and send double point values (IEC ASDU type 3 or 31).
EncWrite_005_032_ST_NA_TB	Encode and send step positions (IEC ASDU type 5 or 32).
EncWrite_009_034_ME_NA_TD	Encode and send measured values (IEC ASDU type 9, 11, 34 or
EncWrite_013_036_ME_NC_TF	Encode and send measured values (IEC ASDU type 13 or 36).
EncWrite_015_037_IT_NA_TB	Encode and integrated total values (IEC ASDU type 15 or 37).
EncWrite_015_037_IT_NA_TB_RI	Encode and integrated total values (IEC ASDU type 15 or 37).
	Simplified Combined Output functions for Monitoring data types (inputs are simple types):

EncWrite_xxx_xxx_xx_Nx_Tx_NIO	Same name as the corresponding function for IO types, name ends with "_NIO"
	Functions for command handling; these functions receive a command, decode it and sent the proper responses:
Rcv_045_SC_NA	Receive a single command (IEC ASDU type 45).
Rcv_046_DC_NA	Receive a double command (IEC ASDU type 46).
Rcv_048_DC_NA	Receive a set point command (IEC ASDU type 48).
Rcv_050_SE_NC	Receive a set point command (IEC ASDU type 50).
Rcv_051_BO_NA	Receive a bitstring (IEC ASDU type 51).
Rcv_058_SC_TA	Receive a single command (IEC ASDU type 58).
Rcv_059_DC_TA	Receive a double command (IEC ASDU type 59).
Rcv_063_SE_TC	Receive a set point command (IEC ASDU type 63).
Rcv_064_BO_TA	Receive a bitstring (IEC ASDU type 64).
	Functions for handling select and execute commands:
Rcv_045_SC_NA_SE	Receive a single command (IEC ASDU type 45).
Rcv_046_DC_NA_SE	Receive a double command (IEC ASDU type 46).
Cmd_Or	"Or" function block for commands.
Cmd_Or_SE	"Or" function block for setpoints. Stores the latest value of the setpoint.
	Combined functions to receive commands (on one or two connections):
CmdRcv_045_058_SC_NA_TA	Receive a single command (IEC ASDU type 45 or 58).
CmdRcv_046_059_DC_NA_TA	Receive a double command (IEC ASDU type 46 or 59).
CmdRcv_047_060_RC_NA_TA	Receive a set point command (IEC ASDU type 47 or 60).
CmdRcv_048_061_SE_NA_TA	Receive a set point command (IEC ASDU type 48 or 61).
CmdRcv_050_063_SE_NC_TC	Receive a set point command (IEC ASDU type 50 or 63).
CmdRcv_051_064_BO_NA_TA	Receive a bitstring (IEC ASDU type 51 or 64).
	Functions for handling counter values:
Rcv_101_CI_NA	Receive a counter interrogation (IEC ASDU type 101), decode it and send a response.
Decode_Cnt_Request	Performs further decoding of a counter request.
Decode_Cnt_Request_Ex	Performs further decoding of a counter request (extended).

4.6.1 Functions to make structured variables

The output conversion functions in the next group need structured variables of type BoolIO or RealIO (so they can be directly connected to I/O variables) as input. For variables that have a simple type (bool or real) because they are created by logic, these two function blocks simplify filling in the structured variables. The two function blocks work the same, except for the different types. Connect the process value to the ***Value*** pin. The ***Status*** pin can be used to indicate the variable status, if it is left unconnected, the default is the "OK" status code (16#C0). If the ***Error*** pin is set to true, the the status code is ignored and a "Bad" status code (16#0C) is used instead, the default is false, so if this function is not needed, the pin can be left unconnected. A possible use for the ***Error*** pin is for signals that are created on a different controller and sent via MMS to set them to "Bad" if the MMS communication is interrupted.

All that data is written to the structured output variable that is connected to the ***Out*** pin.

4.6.2 Output conversion functions for Monitoring data types

These function blocks take individual variables (the fields of a BoolIO variable or a variable calculated in a program) and copy them into a structured variable suitable for the selected ASDU type. They check for changes and set an output variable to indicate if the data has changed and should be sent to the IEC partner.

The functions blocks have a similar function but different input pins, depending on the specific data type. The output is always a structured variable appropriate to the ASDU type. The function blocks are extensible and can accept from 1 to 32 data items.

Compared to the function blocks from the IEC60870SupLib, the ***Enable*** pin has been removed (it does not provide any advantage of using the optional ***EN*** pin) and the function blocks now use BoolIO or RealIO structured types instead of separate pins for the value and status.

While the ***UpdateOutput*** pin is true, the data is always copied to the output variables ignoring Hysteresis processing. The ***ChangedValues*** pin is 0 if the output variables are not changed. Is is 16#FFFFFFFF if they have changed, the ***UpdateOutput*** pin is true or on the first cycle after the Enable pin becomes true.

For each data item there is one input pin (or more) for the value (the type depends on the data block), one or more pins for the status and an output pin that outputs a structured variable. These function blocks assume that the status conversion is handled by the CI864 board and accept an OPC status value.

The pins ***InTime*** and ***InCOT*** are optional parameters. The default is that the timestamp is generated by the PH and the COT (Cause of Transmission) is set to Spontaneous Transmission, which is the wanted behavior in most cases. ***InTime*** sets the timestamp that is sent with the telegram. Generally it should be changed together with the value or status and is used when the 1131 application generates the timestamp. ***InCOT*** is used only in special cases, such as when a data point that is received from one IEC 104 connection is sent to another one and the cause of transmission has to be preserved.

These function blocks assume that the Status Conversion is set to "OPC_STD".

The simplified versions of theses function blocks (..._Basic) ommit the ***InTime*** and ***InCOT*** pins and the associate functionality, which is not needed in many cases. This saves controller resources and makes engineering easier. They behave the same as the above group if these two pins are not connected.

The third group with (...BNIO) is similar to the _Basic blocks, except it uses simple types (bool, real) instead of structured types. BNIO stands for Bool, No IO. Each block has a common ***Status*** pin that applies to all data items.

This library only contains function blocks for the most commonly used data types. If additional data types are required, one of the existing function blocks can be copied and modified (the library is not locked).

4.6.2.1 Encode_030_SP_TB

Conversion of single point values. It has the following extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	BoolIO		in_out (IN)	
InTime	date_and_time	retain	in	
InCOT	dword	retain	in	
OutVal	MSG_IEC101_030_M_SP_TB_Type		in_out (OUT)	

4.6.2.2 Encode_031_DP_TB

Converts two or three bool input values to a dual point value. It has the following extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InValOn	BoolIO		in_out (IN)	
InValOff	BoolIO		in_out (IN)	
ErrValue	bool	retain	in	false
InTime	date_and_time	retain	in	

InCOT	dword	retain	in	
OutVal	MSG_IEC101_031_M_DP_TB_Type		in_out (OUT)	

4.6.2.3 Encode_036_ME_TF

Conversion of a real value with hysteresis handling. It has the following extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	RealIO		in_out (IN)	
InTime	date_and_time	retain	in	
InCOT	dword	retain	in	
Hys	real	retain	in	0.0
OutVal	MSG_IEC101_036_M_ME_TF_Type		in_out (OUT)	
HysDiff	real	hidden	out	0.0

The ***Hys*** input pin determines the hysteresis handling:

If ***Hys*** is 0, there is no hysteresis processing; the Value is copied to the output whenever it is changed.

If ***Hys*** is > 0, it is an absolute hysteresis; the Value is copied to the output when the absolute value of the difference to the previous output is greater than ***Hys***.

If ***Hys*** is < 0, it is an "integrated value change" hysteresis. Each cycle the difference between the Value and the output is added up. When the absolute value of this integrated difference (which is stored in ***HysDiff***) is greater than the absolute value of ***Hys***, the Value is copied to the output and ***HysDiff*** is set to 0.

The ***HysDiff*** pin is normally not used outside the function block.

4.6.2.4 Encode_037_IT_TB_Basic

Conversion of integrated total (counter) values. This function block is substantially different from the others and is described separately. It needs special handling, detailed in an example below.

While the ***UpdateOutput*** pin is true, the data is always copied to the output variables ignoring Hysteresis processing. The ***ChangedValues*** pin is 0 if the output variables are not changed. Is is 16#FFFFFFFF if they have changed, the ***UpdateOutput*** pin is true or on the first cycle after the Enable pin becomes true.

If the ***Trigger*** pin is true, then the ***UpdateOutput*** becomes edge-sensitive, that is data is only copied on the rising edge. In addition, if ***Trigger*** is true, the item data is not checked for changes, it is only updated on the rising edge of ***UpdateOutput***.

Set the ***Trigger*** pin to true to use this function block to freeze integrated totals to process them in the manner suggested in the IEC specification.

Set the ***Trigger*** pin to false to process counters similar to measurements, that is to send them when they change.

The ***COT*** pin gives the Cause of Transmission for all counters and the ***Status*** pin the status (actually the sequence counter and overflow bits).

If the ***SetTimeStamp*** pin is true, then the current time will be set to all counters. This ensures that all counters are sent with the time, when the ***UpdateOutput*** pin is / becomes true.

It has the following extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	BoolIO		in_out (IN)	
OutVal	MSG_IEC101_037_M_IT_TB_Type		in_out (OUT)	

4.6.2.5 Encode_009_ME_NA_BNIO

Conversion of a real value to a normalized integer with hysteresis handling. It has the following extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	Real	retain	in	0.0
Status	dword	retain	in	16#C0

Hys	real	retain	in	0.0
OutVal	MSG_IEC101_036_M_ME_TF_Type		in_out (OUT)	
HysDiff	dint	hidden	out	0

The value (InVal) must be between -1.0 and +1.0. 1.0 is mapped to 32767 internally. Hysteresis processing is done after the value is converted.

The ***Hys*** input pin determines the hysteresis handling:

If ***Hys*** is 0, there is no hysteresis processing; the Value is copied to the output whenever it is changed.

If ***Hys*** is > 0, it is an absolute hysteresis; the Value is copied to the output when the absolute value of the difference to the previous output is greater than ***Hys***.

If ***Hys*** is < 0, it is an "integrated value change" hysteresis. Each cycle the difference between the Value and the output is added up. When the absolute value of this integrated difference (which is stored in ***HysDiff***) is greater than the absolute value of ***Hys***, the Value is copied to the output and ***HysDiff*** is set to 0.

The ***HysDiff*** pin is normally not used outside the function block.

4.6.3 Combined Output functions for Monitoring data types

These functions combine the functionality of a Conv_IEC_IOA function block, an Encode_xxx function block and two IEC60870WriteCont function blocks. These blocks are defined to output data on a fully redundant connection (two CI864 boards), but can also be used for a single connection (in which case they incur a small penalty over using the separate function blocks, mainly in the amount of used memory.)

Generally, it is recommended to use these function blocks, as they are simpler to configure and require significantly less engineering time.

The function blocks with the "_NIO" suffix are designed to be connected to simple data types and have an Input pin for a separate status for each data item while the blocks without the suffix are designed for structured I/O types (BoolIO, RealIO or DintIO).

The functions blocks have a similar function but different input pins, depending on the specific data type. The function blocks are extensible and while they can be extended to 32 data items, only up to 16 data items will work, except for the blocks with the "_32" suffix. Since some of these blocks have several input pins per data item, they may not fit on a page in CBM when used with a large number of data items.

The following pins are the same for all the function blocks:

Name	Data Type	Attributes	Direction	Initial Value
Enable1	bool	retain	in	
Id1	Comm_Channel		in_out	
Enable2	bool	retain	in	
Id2	Comm_Channel		in_out	
CommonAddress	dint	retain	in	
IOA1	dint	retain hidden	in	0
IOA2	dint	retain hidden	in	0
IOA3	dint	retain hidden	in	0
InfoObjAddr_StepSize	dint	retain	in	1
UseTimeStamp	bool	retain	in	true
UpdateOutput	bool	retain	in	false
Valid1	bool	retain	out	
Status1	dint	retain	out	1
Valid2	bool	retain	out	
Status2	dint	retain	out	1
Extensible:				
InStatus	dword		in	

TmpVal	Depending on type		out	
HysDiff	real	hidden	out	0.0

´The ***Id1*** pin must be connected to the Id variable from a valid **IEC60870Connect** function block for this function block to work. The Id variable must be the same one as the one from the Connect block with the same ***IEC60870PartnerPos*** or the Id variable of the first Connect block in a data sharing group. The ***Enable1*** pin must be set to true to enable data transfer on the primary comnection.

The pins ***Enable2*** and ***Id2*** are used for the secondary connection. If the secondary connection is not used, ***Enable2*** should be set to FALSE and ***Id2*** should be connected to a dummy variable. ***Id2*** should not be connected to the same Id variable as ***Id1***, as this will incur additional processing overhead.

The ***CommonAddress*** pin gives the common address for all data elements.

The ***IOA1, IOA2 and IOA3*** give the information object address for the first data element (if an unstructured IOA is used, the the whole IOA can be connected to ***IOA1*** while ***IOA2*** and ***IOA3*** are set to 0). The information object address is increased by ***InfoObjAddr_StepSize*** for each following data element. The Common Address and the Step Size are given with single input pins, since these are typically set to the same number on many such function blocks and so should be calculated once at the beginning of the IEC program and stored in variables.

The ***UseTimeStamp*** pin indicates if the IEC data type with a Time Stamp (30 to 37) is used or if the IEC data type without a Time Stamp (1, 3, 5, ...) is used. The name of the function block indicates which ASDU types are used: e.g. **EncWrite_003_031_DP_NA_TB_NIO** uses either type 3 (if ***UseTimeStamp*** is FALSE or type 31 if it is TRUE). The value of this pin should not be changed online, if it is changed, the corresponding Connect block(s) should be disabled and then enabled again.

While the ***UpdateOutput*** pin is true, the data is always copied to the IEC variables ignoring Hysteresis processing.

The ***Valid1*** and ***Valid2*** pins indicate if the data elements contain valid data (or at least some of them).

The ***Status1*** and ***Status2*** pins contain information about possible errors.

The ***InStatus*** pin can be used to indicate the variable status, if it is left unconnected, the default is the "OK" status code (16#C0). This pin is only present on function blocks for simple types ("_NIO").

The structured I/O variables have a Status member that indicates the item status. The structured I/O variables use the Direction "in_out", but functionally they are actually direction "in".

The ***TmpVar*** output pin is actually an internal variable and should normally not be connected. It can be connected for debugging purposes, to see if the encoding part of the function block works properly.

This library only contains function blocks for the most commonly used data types. If additional data types are required, one of the existing function blocks can be copied and modified (the library is not locked).

4.6.3.1 EncWrite_001_030_SP_NA_TB and EncWrite_001_030_SP_NA_TB_32

Output of single point values. It has the following additional extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	BoolIO		in_out (IN)	

4.6.3.2 EncWrite_001_030_SP_NA_TB_NIO

Output of single point values. It has the following additional extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	bool		in	
InStatus	dword		in	

4.6.3.3 EncWrite_003_031_DP_NA_TB

Output of a double point data item composed of two or three bool input values. It has the following additional extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InValOn	BoolIO		in_out (IN)	
InValOff	BoolIO		in_out (IN)	
ErrValue	bool		in	

4.6.3.4 EncWrite_003_031_DP_NA_TB_SI

Output of a double point data item composed of two or three bool input values. It has the following additional extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InValOn	BoolIO		in_out (IN)	
InValOff	BoolIO		in_out (IN)	
ErrValue	BoolIO		in_out (IN)	

4.6.3.5 EncWrite_003_031_DP_NA_TB_NIO

Output of a double point data item composed of two or three bool input values. It has the following additional extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InValOn	bool		in	
InValOff	bool		in	
InValError	bool	retain	in	false
InStatus	dword		in	

4.6.3.6 EncWrite_005_032_ST_NA_TB

Output of a trafo step position. It has the following additional extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	RealIO		in_out (IN)	

4.6.3.7 EncWrite_005_032_ST_NA_TB_NIO

Output of a trafo step position. It has the following additional extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	real		in	
InTrans	bool		in	false
InStatus	dword		in	

The value (given on ***InVal***) is given as a real and converted to an integer (-64 to 63), ***InTrans*** gives if the equipment is in transient state (if this information is used).

4.6.3.8 EncWrite_007_033_BO_NA_TB_NIO

Output of a bitstring of 32 bit. It has the following additional extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	dword		in	

4.6.3.9 EncWrite_009_034_ME_NA_TD_NIO

Output of a real (floating point) value as an integer value with hysteresis handling. It has the following additional extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
Scale	real	retain	in	32767.0
UseASDU_11_35	bool	retain	in	false

InVal	RealIO		in_out	
Offset	real		in	0.0
Mult	real		in	1.0
HysA	real		in	0.0
HysI	real		in	0.0
HysDiff	dint	retain	out	0

The value (given on ***InVal***) is converted to an integer value between -32768 and 32767, using the following formula: value := (***InVal*** - ***Offset***) / ***Mult*** * ***Scale***. EncWrite_009_034_ME_NA_TD_NIO

Output of a real (floating point) value as an integer value with hysteresis handling. It has the following additional extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
UseASDU_11_35	bool	retain	in	false
InVal	real		in	
InStatus	dword		in	
Val_X0	real		in	0.0
Val_X1	real		in	1.0
HysA	real		in	0.0
HysI	real		in	0.0
HysDiff	dint	retain	out	0

The ***UseASDU_11_35*** input pin determines which data type is used. If it is TRUE, types 11 or 35 are used, if FALSE, types 9 or 34 are used.

The value (given on ***InVal***) is converted to an integer value between -32768 and 32767, using the following formula: value := (***InVal*** - ***Val_X0***) / ***Val_X1*** * 32767.

The ***HysA*** and ***HysI*** input pin determines the hysteresis handling, see the function block **EncWrite_013_036_ME_NC_TF_NIO** below. The hysteresis values are converted simiarly to the actual value, except that only ***Val_X1*** is used.

The ***HysDiff*** output pin is actually an internal variable and should normally not be connected. This pin is used to store the current value of the integrated hysteresis

4.6.3.10 EncWrite_013_036_ME_NC_TF

Conversion of a real value with hysteresis handling. It has the following additional extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	RealIO		in_out (IN)	
HysA	real		in	0.0
HysI	real		in	0.0
HysDiff	real	retain	out	0.0

4.6.3.11 EncWrite_013_036_ME_NC_TF_Sc

Conversion of a real value with hysteresis handling. It has the following additional extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	RealIO		in_out (IN)	
Offset	real		in	0.0
Mult	real		in	1.0
HysA	real		in	0.0
HysI	real		in	0.0
HysDiff	real	retain	out	0.0

The parameters ***Offset*** and ***Mult*** allow the value to be rescaled according to the following formula: value := (***InVal - Offset***) / ***Mult***.

4.6.3.12 EncWrite_013_036_ME_NC_TF_NIO

Conversion of a real value with hysteresis handling. It has the following additional extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	real		in	
InStatus	dword		in	
HysA	real		in	0.0
HysI	real		in	0.0
HysDiff	real	retain	out	0.0

The ***HysA*** and ***HysI*** input pin determines the hysteresis handling:

HysA gives an absolute hysteresis. If ***HysA*** is 0 or less than 0, there is no hysteresis processing; the Value is copied to the output whenever it is changed.

If ***HysA*** is > 0, it is an absolute hysteresis; the Value is copied to the output when the absolute value of the difference to the previous output is greater than ***HysA***.

HysI gives an "integrated value change" hysteresis. If ***HysI*** is 0, then the integrated hysteresis is not used. Each cycle the difference between the Value and the output is added up. When the absolute value of this integrated difference (which is stored in ***HysDiff***) is greater than the absolute value of ***HysI***, the Value is copied to the output and ***HysDiff*** is set to 0.

The ***HysDiff*** output pin is actually an internal variable and should normally not be connected. This pin is used to store the current value of the integrated hysteresis

4.6.3.13 EncWrite_015_037_IT_NA_TB_RI

Conversion of integrated total (counter) values. It has the following additional extensible paramters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	RealIO		in_out (IN)	
InStatus	dword		in	0
InCF	real		in	1.0

The parameter ***InCF*** allows the value to be rescaled according to the following formula: value := ***InVal*** * ***InCF***.

4.6.3.14 EncWrite_015_037_IT_NA_TB_NIO

Conversion of integrated total (counter) values. It has the following additional extensible paramters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	dint		in	
InStatus	dword		in	0

This function block is substantially different from the others and is described separately. It needs special handling, detailed in an example below.

While the ***UpdateOutput*** pin is true, the data is always copied to the output variables

If the ***Trigger*** pin is true, then the ***UpdateOutput*** becomes edge-sensitive, that is data is only copied on the rising edge. In addition, if ***Trigger*** is true, the item data is not checked for changes, it is only updated on the rising edge of ***UpdateOutput***.

Set the ***Trigger*** pin to true to use this function block to freeze integrated totals to process them in the manner suggested in the IEC specification.

Set the ***Trigger*** pin to false to process counters similar to measurements, that is to send them when they change.

The ***COT*** pin gives the Cause of Transmission for all counters.

The ***InStatus*** pin the status of this block is not used for the I/O status but for the sequence counter and overflow bits:

Sequence Number (0-31) OR (bitwise OR)

16#20 if an overflow occurred OR

16#40 if the counter was adjusted OR

16#80 if the counter is invalid.

The ***TimeStamp*** gives the Time Stamp to be used for the counters, leave at the default value to transmit the current time at the time the counters are sent.

To force all the counters to be sent at a specific time (even if the counter value has not changed), either toggle the ***EnableX*** pins or give a changed time on the ***TimeStamp*** pin (only when using the type with Time Stamp).

4.6.4 Input handling for Command types

These function blocks handle receiving a command, formulating a response message and sending it back to the partner. They are very similar, except that the output pins depend on the type of the command.

These function blocks use several external variables that must be defined at the application level:

Name	Data Type	Attrib-utes	Description
IEC60870_Cmd_BlockedIsError	bool		If true an error code will be reported to the IEC partner if the command is blocked (The pin Blocked is true), otherwise the command will be acknowledged to the IEC partner normally but is silently ignored.
IEC60870_Cmd_NAck_Blocked	bool		If true the execution of the command will be reported with a negative acknowledge if the command is blocked. If IEC60870_Cmd_BlockedIsError is true, then this parameter is ignored.
IEC60870_Cmd_AutoAckTime	time		Time for which the command is active if CmdAutoAck is true. Also used when the command is blocked. 2 seconds (time#2s) is typically used here.
IEC60870_Cmd_Command_Term	dint		Number of acknowledgement messages to send for commands. Must be the same as the hardware parameter Command termi-nation and must use the same value as the master does for this parameter.
IEC60870_Cmd_SetPoint_Term	dint		Number of acknowledgement messages to send for setpoints. Must be the same as the hardware parameter SetPoint termina-tion and must use the same value as the master does for this parameter.

The following pins are the same for all the function blocks:

Name	Data Type	Attributes	Direction	Initial Value
Enable	bool	retain	in	
Id	Comm_Channel		in	
CommonAddress	dint	retain	in	
InformationOb-jectAddress	dint	retain	in	
Confirm	bool	retain	in	true
Blocked	bool	retain	in	true
CmdAutoAck	bool	retain	in	false
CmdOk	bool	retain	in	false
CmdError	bool	retain	in	false

Valid	bool	retain	out	false

The function block only operates if the ***Enable*** pin is true. The pins ***Id***, ***CommonAddress*** and ***InformationObjectAddress*** are connected directly to the Read and Write function blocks inside.

If ***Blocked*** is true, then the command is blocked and will not set the output pins. How this is reported to the IEC partner is determined by the ***IEC60870_Cmd_BlockedIsError*** and ***IEC60870_Cmd_NAck_Blocked*** variables.

If ***CmdAutoAck*** is true, then the command is automatically reported as 'successfully executed' to the IEC partner. Otherwise ***CmdOk*** or ***CmdError*** have to be used to report the successful (or not successful) execution of the command.

If ***Confirm*** is true, confirmation messages are sent, otherwise they are suppressed. This should only be set to false for the passive connection in some redundancy schemes.

The ***Valid*** pin is true while the command is running.

The most common case is to set ***Blocked*** to false (or to connect it to the switch that controls if this station is remotely controlled) and to set ***CmdAutoAck*** to true.

4.6.4.1 Rcv_045_SC_NA and Rcv_058_SC_TA

Handling of a single command.

Name	Data Type	Attributes	Direction	Initial Value
CmdOn	bool	retain	out	false
CmdOff	bool	retain	out	false

If the 'Single Command State' is 1 (on) then ***CmdOn*** is true, otherwise ***CmdOff*** is true. In any case, they are only true while the command is running. Usually only ***CmdOn*** is used for a single command.

4.6.4.2 Rcv_046_DC_NA and Rcv_059_DC_TA

Handling of a double command.

Name	Data Type	Attributes	Direction	Initial Value
CmdOn	bool	retain	Out	false
CmdOff	bool	retain	Out	false

If the 'Double Command State' is 1 (on) then ***CmdOn*** is true, if it is 2, ***CmdOff*** is true. In any case, they are only true while the command is running.

4.6.4.3 Rcv_048_SE_NC

Handling of a setpoint.

Name	Data Type	Attributes	Direction	Initial Value
SPValue	real	retain	Out	0.0

The value of the setpoint command is output on ***SPValue***. In most cases it makes sense to store this value in another variable. The ***SPValue*** is rescaled so it is between -1.0 and 1.0. The ***Valid*** indicates when a new value has been received.

4.6.4.4 Rcv_050_SE_NC and Rcv_063_SE_TC

Handling of a setpoint.

Name	Data Type	Attributes	Direction	Initial Value
SPValue	real	retain	Out	0.0

The value of the setpoint command is output on ***SPValue***. In most cases it makes sense to store this value in another variable. The ***Valid*** indicates when a new value has been received.

4.6.4.5 Rcv_051_BO_NA and Rcv_064_BO_TA

Handling of a bitstring.

Name	Data Type	Attributes	Direction	Initial Value
BOValue	dword	retain	Out	0

The value of the bitsting command is output on ***BOValue***. In most cases it makes sense to store this value in another variable. The ***Valid*** indicates when a new value has been received.

4.6.5 Input handling for Select / Execute Command types

These function blocks handle receiving a command, formulating a response message and sending it back to the partner for Select/Execute commands.

The function blocks are named with an additional "_SE" in the function block name and they behave the same as the regular ones for direct execute commands, except with the following additions:

These function blocks use the following additional external variables that must be defined at the application level:

Name	Data Type	Attributes	Description
IEC60870_Cmd_SelectTime	time		Maximum time a command stays selected before it is aborted.

The function blocks have the following extra pins:

Name	Data Type	Attributes	Direction	Initial Value
Selected	bool	retain	out	false

The ***Selected*** pin becomes true when a select message has been received for this command.

In a redundant configuration this should be used to block all other Select/Execute commands since the firmware of the CI board can only block other S/E commands that are received on the same connection.

4.6.6 Combined functions to receive commands

These functions combine the functionality of a Conv_IEC_IOA function block, two Rcv_xxx function block and one Cmd_Or (or Cmd_Or_SE) function blocks. These blocks are defined to receive commands on a fully redundant connection (two CI864 boards), but can also be used for a single connection (in which case they incur a small penalty over using the seperate function blocks, mainly in the amount of used memory.)

Generally, it is recommended to use these function blocks, as they are simpler to configure and require significantly less engineering.

These function blocks use the same external variables as the normal command handling blocks.

The functions blocks have a similar function but different input pins, depending on the specific data type. The following pins are the same for all the function blocks:

Name	Data Type	Attributes	Direction	Initial Value
Enable1	bool	retain	in	
Id1	Comm_Channel		in_out	
Enable2	bool	retain	in	
Id2	Comm_Channel		in_out	
CommonAddress	dint	retain	in	
IOA1	dint	retain hidden	in	0
IOA2	dint	retain hidden	in	0
IOA3	dint	retain hidden	in	0
UseTimeStamp	bool	retain	in	true
Blocked1	bool	retain	in	true

Confirm1	bool	retain	in	true
Blocked2	bool	retain	in	true
Confirm2	bool	retain	in	true
CmdAutoAck	bool	retain	in	false
CmdOk	bool	retain	in	false
CmdError	bool	retain	in	false
Valid	bool	retain	out	false
UseSelectExecute	bool	retain	in	false
Selected	bool	retain	out	false
PulseValid3	bool	retain	in	false
Valid3	bool	retain	in	false
SPValue3	real	retain	in	0.0
SPValue	real	retain	out	0.0

´The ***Id1*** pin must be connected to the Id variable from a valid **IEC60870Connect** function block for this function block to work. The Id variable must be the same one as the one from the Connect block with the same ***IEC60870PartnerPos*** or the Id variable of the first Connect block in a data sharing group. The ***Enable1*** pin must be set to true to enable data transfer on the primary comnection.

The pins ***Enable2*** and ***Id2*** are used for the secondary connection. If the secondary connection is not used, ***Enable2*** should be set to FALSE and ***Id2*** should be connected to a dummy variable. ***Id2*** should not be connected to the same Id variable as ***Id1***, as this will incur additional processing overhead.

The ***CommonAddress*** pin gives the common address for all data elements.

The ***IOA1, IOA2 and IOA3*** give the information object address for the first data element (if an unstructured IOA is used, the the whole IOA can be connected to ***IOA1*** while ***IOA2*** and ***IOA3*** are set to 0). The Common Address is given with a single input pin, since it is typically set to the same number on many such function blocks and so should be calculated once at the beginning of the IEC program and stored in variables.

The ***UseTimeStamp*** pin indicates if the IEC data type with a Time Stamp (58 to 64) is used or if the IEC data type without a Time Stamp (45 to 51) is used. The name of the function block indicates which ASDU types are used: e.g. **CmdRcv_045_058_SC_NA_TA** uses either type 58 (if ***UseTimeStamp*** is FALSE or type 45 if it is TRUE).

If ***Blocked1*** (or ***Blocked2*** for the secondary connection) is true, then the command is blocked and will not set the output pins. How this is reported to the IEC partner is determined by the ***IEC60870_Cmd_BlockedIsError*** and ***IEC60870_Cmd_NAck_Blocked*** variables.

If ***CmdAutoAck*** is true, then the command is automatically reported as 'successfully executed' to the IEC partner. Otherwise ***CmdOk*** or ***CmdError*** have to be used to report the successful (or not successful) execution of the command.

If ***Confirm1*** (or ***Confirm2*** for the secondary connection) is true, confirmation messages are sent, otherwise they are suppressed. This should only be set to false for the passive connection in some redundancy schemes.

The ***Valid*** pin is true while the command is running.

The most common case is to set ***Blocked*** to false (or to connect it to the switch that controls if this station is remotely controlled) and to set ***CmdAutoAck*** to true.

If ***UseSelectExecute*** is true, then the blocks use the logic for Select/Execute Commands, otherwise they use normal commands with Execute only. The output pin ***Selected*** becomes true when the select message has been received for this command. This functionality is only supported for Single and Double commands (Types 45, 46, 58 and 59).

SetPoint types (48/49, 50, 51, 61/62, 63 and 64) use the pins ***PulseValid3***, ***Valid3*** and ***SPValue3*** to set the SP value from logic and output the value on ***SPValue***. See the function block **Cmd_Or_SE** below for details. For bitstring commands the value pins are called ***BOValue*** instead of ***SPValue***.

4.6.6.1 CmdRcv_045_058_SC_NA_TA

Handling of a single command.

Name	Data Type	Attributes	Direction	Initial Value
CmdOn	bool	retain	out	false
CmdOff	bool	retain	out	false

If the 'Single Command State' is 1 (on) then ***CmdOn*** is true, otherwise ***CmdOff*** is true. In any case, they are only true while the command is running. Usually only ***CmdOn*** is used for a single command.

4.6.6.2 CmdRcv_046_059_DC_NA_TA

Handling of a double command.

Name	Data Type	Attributes	Direction	Initial Value
CmdOn	bool	retain	out	false
CmdOff	bool	retain	out	false

If the 'Double Command State' is 1 (on) then ***CmdOn*** is true, if it is 2, ***CmdOff*** is true. In any case, they are only true while the command is running.

4.6.6.3 CmdRcv_047_060_RC_NA_TA

Handling of a double command.

Name	Data Type	Attributes	Direction	Initial Value
CmdOn	bool	retain	out	false
CmdOff	bool	retain	out	false

If the 'Regulating Command State' is 1 (higher) then ***CmdOn*** is true, if it is 2 (lower), ***CmdOff*** is true. In any case, they are only true while the command is running.

4.6.6.4 CmdRcv_048_061_SE_NA_TA

Handling of a setpoint.

Name	Data Type	Attributes	Direction	Initial Value
UseASDU_49_62	bool	retain	in	false
Val_X0	real	retain	in	0.0
Val_X1	real	retain	in	1.0
SPValue	real	retain	out	0.0

The ***UseASDU_49_62*** input pin determines which data type is used. If TRUE, types 49 or 62 are used, if FALSE, types 48 or 61 are used.

The value of the setpoint command is output on ***SPValue***. The received setpoint value is between -1.0 and 1.0 before it is rescaled. SPValue := ***Val_X0*** + spvalue * ***Val_X1***.

The ***Valid*** indicates when a new value has been received.

4.6.6.5 CmdRcv_050_063_SE_NC_TC

Handling of a setpoint.

Name	Data Type	Attributes	Direction	Initial Value
SPValue	real	retain	Out	0.0

The value of the setpoint command is output on ***SPValue***. The ***Valid*** indicates when a new value has been received.

4.6.6.6 CmdRcv_051_064_BO_NA_TA

Handling of a bitstring.

Name	Data Type	Attributes	Direction	Initial Value
BOValue	dword	retain	Out	0

The value of the bitsting command is output on ***BOValue***. The ***Valid*** indicates when a new value has been received.

4.6.6.7 Cmd_Or

This function block combines the received commands from two different Rcv function blocks. It is used in a redundant configuration with two CI boards as each board needs a seperate Rcv block. This block does not do a whole lot more than a normal "or", but since it is a function block and not a function, it can be created from a bulk data sheet.

If ***Blocked*** is true, the outputs are always false.

CmdOn is true if either ***CmdOn1*** or ***CmdOn2*** is true (or both).

CmdOff is true if either ***CmdOff1*** or ***CmdOff2*** is true (or both).

4.6.6.8 Cmd_Or_SE

This function block combines the received setpoints from two different Rcv function blocks. It is used in a redundant configuration with two CI boards as each board needs a seperate Rcv block.

If ***Blocked*** is true, the setpoint value is not copied to the output.

If ***Valid1*** is true, the value of ***SPValue1*** is copied to the output ***SPValue*** and ***Valid*** is set to true.

If ***Valid2*** is true, the value of ***SPValue2*** is copied to the output ***SPValue*** and ***Valid*** is set to true.

If ***Valid3*** is true, the value of ***SPValue3*** is copied to the output ***SPValue. Valid*** is only set to true if ***PulseValid3*** is also true, otherwise the value is simply copied. ***Valid3*** has priority over the others. The third input is to be able to set the setpoint value from logic, for example to track the local set-point value while the device is in local control.

4.6.7 Functions for handling counter values:

4.6.7.1 Rcv_101_CI_NA

This function block handles receiving of a counter request, formulating a response message and sending it back to the partner.

Pins of this function block:

Name	Data Type	Attributes	Direction	Initial Value
Enable	bool	retain	in	
Id	Comm_Channel		in	
CommonAddress	dint	retain	in	
InformationObjectAddress	dint	retain	in	0
TermTime	time	retain	in	time#3s
Done	bool	retain	in	false
Valid	bool	retain	out	false
Cnt_Request	dint	retain	out	0
Cnt_Read	bool	retain	out	false
Cnt_Freeze	bool	retain	out	false
Cnt_Reset	bool	retain	out	false

The function block only operates if the ***Enable*** pin is true. The pins ***Id***, ***CommonAddress*** and ***Infor-mationObjectAddress*** are connected directly to the Read and Write function blocks inside. The ***In-formationObjectAddress*** is normally 0 according to the IEC spec.

The termination message is sent after ***TermTime*** has elapsed, unless ***Done*** is set to true before that. The time should be long enough to send all the counter values to the partner before it elapses, ***Done*** can be used to send the termination message faster when no data has to be sent (such as when freezing counters).

Cnt_Request contains the Qualifyer of the counter command (RQT, see the IEC spec).

Cnt_Read, ***Cnt_Freeze*** and ***Cnt_Reset*** are set according to the FRZ part of the Qualifyer.

The Counter Interrogation is treated like a command. Because it has a fixed IOA address, only one CI request can be handled at the same time.

4.6.7.2 Decode_Cnt_Request

This function block assists in decoding of the ***Cnt_Request*** parameter of the Rcv_101_CI_NA block.

The ***Cnt_Request*** pin should be connected to the ***Cnt_Request*** output pin of the Rcv_101_CI_NA block.

One of ***Cnt_Group1***, ***Cnt_Group2***, ***Cnt_Group3***, ***Cnt_Group4*** and ***Cnt_Gen*** is set to true, depending on which counter group was requested in the Qualifyer.

Cnt_COT received the Cause of Transmission to be used when sending the counter values, this pin is usually connected to the ***COT*** pin of the Encode_037_IT_xxx blocks used processing this CI request.

4.6.7.3 Decode_Cnt_Request_Ex

This function block does more extensive decoding than Decode_Cnt_Request. It implements practically all the logic needed to implement Counter Interrogation handling according to the IEC spec.

The pins ***Cnt_Request***, ***Cnt_Read***, ***Cnt_Freeze*** and ***Cnt_Reset*** should be connected to the corresponding output pins of the Rcv_101_CI_NA block.

Conn_Valid should normally be connected to the ***Valid*** output of the Connect block. The counters become "unfrozen" when this pin becomes false.

Send_Time_MS gives the pulse duration (in milliseconds) for the ***Counter_GroupX_Send*** pins in a read request. This should be long enough for at least two task cycles of the IEC program.

Freeze_Reset_Time_MS gives the pulse duration (in milliseconds) for the ***Counter_GroupX_Freeze*** and ***Counter_GroupX_Reset*** pins during a Freeze or Reset request.

Local_Freeze determines if counter freezing/resetting is done by local logic (if true) or by counter requests (if false). If this pin is true, Freeze and Reset requests are ignored by this function block and counters are always considered to be "frozen".

Cnt_GroupX_Send (where X is the counter group; 1 to 4) are pulsed during a Counter Read request. Usually these pins are connected directly to the ***Enable*** pin of the IEC60870Write blocks for this counter group.

Cnt_GroupX_Freeze (where X is the counter group; 1 to 4) are pulsed during a Counter Freeze request. Usually these pins are connected directly to the ***Update_Output*** pin of the Encode_037_IT_TB blocks for this counter group.

Cnt_GroupX_Reset (where X is the counter group; 1 to 4) are pulsed during a Counter Reset request.

One of the above pins is pulsed when the CI command applies to one specific counter group, all four of a type are pulsed in a general counter request.

Cnt_COT received the Cause of Transmission to be used when sending the counter values, this pin is usually connected to the ***COT*** pin of the Encode_037_IT_xxx blocks used processing this CI request.

Done is pulsed for one task cycle at the completion of a Freeze or Reset operation. This pin is usually connected to the ***Done*** pin of the Rcv_101_CI_NA block.

This function block implements counter processing according to Mode B (local freeze with counter interrogation) if ***Local_Freeze*** is true or Mode C (freeze and transmit by counter interrogation commands) if ***Local_Freeze*** is false. It can also be used to implement the freezing part of Mode D (freeze by counter interrogation command, frozen values reported spontaneously).

Mode A (local freeze with spontaneous transmission) and the transmission part of Mode D have to be implemented by other logic.

4.7 Software configuration with IEC60870MasterLib:

This library contains function blocks that build on the function blocks from IEC60870CommLib to data handling functions for a master station; that is a station that receives monitoring messages and sends commands and setpoints. All the following function blocks could be implemented in the application by hand, but it usually makes sense to use the provided ones.

This library contains the following function blocks:

Function Block	Description
Decode_030_SP_TB	Convert a structured variable (IEC ASDU type 30) to simple data types.
Decode_031_DP_TB	Convert a structured variable (IEC ASDU type 31) to simple data types.
Decode_032_ST_TB	Convert a structured variable (IEC ASDU type 32) to simple data types.
Decode_034_ME_TD	Convert a structured variable (IEC ASDU type 34) to simple data types.
Decode_036_ME_TF	Convert a structured variable (IEC ASDU type 36) to simple data types.
Decode_037_IT_TB	Convert a structured variable (IEC ASDU type 37) to simple data types.
	Version 2 Decode blocks:
Decode_030_SP_TB_V2	Convert a structured variable (IEC ASDU type 30) to simple data types.
Decode_031_DP_TB_V2	Convert a structured variable (IEC ASDU type 31) to simple data types.
Decode_032_ST_TB_V2	Convert a structured variable (IEC ASDU type 32) to simple data types.
Decode_036_ME_TF_V2	Convert a structured variable (IEC ASDU type 36) to simple data types.
	Send a command, receive and report the response.
Send_045_SC_NA	Send a command (IEC ASDU type 45).
Send_046_DC_NA	Send a command (IEC ASDU type 46).
Send_050_SE_NC	Send a setpoint (IEC ASDU type 50).
Send_051_BO_NA	Send a bitstring (IEC ASDU type 51).
Send_058_SC_TA	Send a command (IEC ASDU type 58).
Send_050_DC_TA	Send a command (IEC ASDU type 59).
Send_063_SE_TC	Send a setpoint (IEC ASDU type 63).
Send_064_SE_TC	Send a bitstring (IEC ASDU type 64).
Send_xxx_xx_xx_Dual	Send a command/setpoint on two redundant CI864 boards.
Send_xxx_xx_xx_SE	Send a command/setpoint using Select/Execute rather than direct execute.
	Combined command sending blocks:
Cmd-Send_045_058_SC_NA_TA	Send a single command (IEC ASDU type 45 or 58).
Cmd-Send_046_059_DC_NA_TA	Send a double command (IEC ASDU type 46 or 59).
Cmd-Send_047_060_RC_NA_TA	Send a regulating command (IEC ASDU type 47 or 60).
Cmd-Send_048_061_SE_NA_TA	Send a setpoint (IEC ASDU type 48, 49, 61 or 62).
Cmd-Send_050_063_SE_NC_TC	Send a setpoint (IEC ASDU type 50 or 63).

Cmd-Send_051_064_BO_NC_TC	Send a bitstring command (IEC ASDU type 51 or 64).
DetectGATerm	Report the end of a General Acquisition from a slave.
	Functions for handling counter values:
Make_CntReq_Freeze_Read	Generate a Counter Interrogation sequence, freezing and then reading a group of counters.
Send_101_CI_NA	Send a Counter Interrogation request.

4.7.1 Input conversion functions for Monitoring data types

These function blocks take the structured variable of the selected ASDU type and comvert it into individual variables (the fields of a BoolIO variable or a variable calculated in a program).

The functions blocks have a similar function but different input pins, depending on the specific data type. The optput is always a structured variable appropriate to the ASDU type. They are extensible and can accept from 1 to 32 data items.

These function blocks access several external variables that must be defined:

Name	Data Type	Attrib-utes	Description
IEC60870_Rcv_ConnBadDelay	time		Time after which the data points are set to invalid if no valid connection to the IEC partner exists. Used to avoid setting all inputs invalid when there is a short interruption in the communication, such as during an application download or redundancy switchover.

Compared to the function blocks from the IEC60870SupLib, the ***Enable*** pin has been removed (it does not provide any advantage of using the optional ***EN*** pin) and the function blocks now use BoolIO or RealIO structured types instead of seperate pins for the value and status.

If simple variables are used in a project (bool rather than BoolIO), then a simple "mov" function can be used to copy the Value field from the structured IEC variables to the application variables. The Decode_DP_x function blocks can be used for Double Indication variables.

The ***ConnValid*** pin should normally be connected to the relevant output of the PartnerStatus block and the ***RcvValid*** pin to the***Valid*** pin of the Receive function block. If either of these pins is false, this indicates that no valid data is availiable. After the defined timeout, the status of all variables will be set to indicate an I/O error. The timeout is to be able to avoid setting the variables invalid during a redundancy switchover.

The ***UpdateOut*** pin can be used to indicate if the input variables have changed or not. The default is that the function block assumes that the variables have changed and processes them every cycle. If it is known that the input variables have not changed, then this pin can be used to to save CPU time. The timeout handling for I/O error is still done, even if this pin indicates that the input has not changed. This pin is usually connected to the ***NewData*** pin of the Receive block.

The ***BadSignals*** pin is true if any of the signals has a status code indicating an error.

The ***InVal*** pin is connected to the structured variable written by the Receive function block.

The pins ***OutTime*** has the timestamp of the latest value change, either the time that was received from the IEC partner, was filled in by the CI board (e.g. when the time stamp was marked as invalid) or generated by the function block, when the data is set as invalid. Bits in the status code indicate the status of the timestamp, as described above.

The ***OutCOT*** pin contains the Cause of Transmission. ***OutCOT*** is useful only in special cases, such as when a data point that is received from one IEC 104 connection is sent to another one and the cause of transmission has to be preserved.

These function blocks assume that the Status Conversion is set to "OPC_STD".

This library only contains function blocks for the most commonly used data types. If additional data types are required, one of the existing function blocks can be copied and modified (the library is not locked).

The Version 2 ("_V2") Decode blocks have the same pins as the regular ones, with the following additions:

The input pins ***RcvValid2*** and ***UpdateOut2*** have the same function as ***RcvValid*** and ***UpdateOut***. These pins are used when two Receive blocks are used (redundant CI864 Modules). The pins are or-ed internally so the pins can be directly connected to the Receive blocks.

The ***CmpVal*** pin is normally not used outside the function block.

For the differences in behavior and use between the regular and Version 2 Decode blocks, see the section on Data Consistency.

4.7.1.1 Decode_030_SP_TB

Conversion of single point values. It has the following extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	MSG_IEC101_030_M_SP_TB_Type		in_out (IN)	
OutVal	BoolIO	retain	in_out (OUT)	
OutTime	date_and_time	retain	out	
OutCOT	dword	retain	out	

4.7.1.2 Decode_031_DP_TB

Converts two or three bool input values to a dual point value. It has the following extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	MSG_IEC101_031_M_DP_TB_Type		in_out (IN)	
OutValOn	BoolIO		in_out (OUT)	
OutValOff	BoolIO		in_out (OUT)	
OutTime	date_and_time	retain	out	
OutCOT	dword	retain	out	

4.7.1.3 Decode_034_ME_TD

Conversion of a real value. It has the following extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	MSG_IEC101_034_M_ME_TD_Type		in_out (IN)	
OutVal	RealIO	retain	in_out (OUT)	
OutTime	date_and_time	retain	out	
OutCOT	dword	retain	out	

The measurement value which is sent as an integer with a value between -32768 and 32767 is re-scaled to a value between -1.0 and 1.0.

4.7.1.4 Decode_036_ME_TF

Conversion of a real value. It has the following extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	MSG_IEC101_036_M_ME_TF_Type		in_out (IN)	
OutVal	RealIO	retain	in_out (OUT)	
OutTime	date_and_time	retain	out	
OutCOT	dword	retain	out	

4.7.1.5 Decode_037_IT_TB

Conversion of a counter value. It has the following extensible parameters per data item.

Name	Data Type	Attributes	Direction	Initial Value
InVal	MSG_IEC101_037_M_IT_TB_Type		in_out (IN)	
OutVal	DintIO	retain	in_out (OUT)	
OutTime	date_and_time	retain	out	
OutCOT	dword	retain	out	

4.7.2 Output handling for Command types

These function blocks handle sending a command and receiving and decoding the response message from the partner. They are very similar, except that the output pins are slightly different.

These function blocks access several external variables that must be defined:

Name	Data Type	Attributes	Description
IEC60870_Cmd_AckWaitTime	time		Time to wait for the response from the partner.
IEC60870_Cmd_Command_Term	dint		Number of response messages to the command. This parameter has to be the same as the corresponding one in the HW parameters.
IEC60870_Cmd_SetPoint_Term	dint		Number of response messages to the setpoint command. This parameter has to be the same as the corresponding one in the HW parameters.

The following pins are the same for all the function blocks:

Name	Data Type	Attributes	Direction	Initial Value
Enable	bool	retain	in	
Id	Comm_Channel		in	
CommonAddress	dint	retain	in	
InformationObjectAddress	dint	retain	in	
Ok	bool	retain	out	false
Error	bool	retain	out	false
Running	bool	retain	out	false
Status	bool	retain	out	false

The function block only operates if the ***Enable*** pin is true. The pins ***Id***, ***CommonAddress*** and ***InformationObjectAddress*** are connected directly to the Read and Write function blocks inside.

The ***Ok*** pin is set to true if the command completed successfully.

The ***Error*** pin is set to true for one cycle if an error occurred.

The ***Running*** pin is true while the command is in progress. It is not possible to send the same command again until the previous one is finished.

The ***Status*** pin indicates success or failure of the command. It is 0 while the command is running. It is set to 1 if the command completes successfully. A negative value indicates an error, –1 indicates that no answer was received from the partner, –2 that the partner sent a negative acknowledge, other negative values indicate that the function blocks that actually send or receive data reported an error.

Each function block will have different input pins for the actual command and command data. It reacts to the rising flank on the pin(s) that are used to send the command.

Most of those function blocks exist in two versions that use different data types, without and with a timestamp, but are otherwise the same, e.g. Send_045_SC_NA uses ASDU type 45 without ("_Nx"; No Time) timestamp, Send_058_SC_TA with ("_Tx"; Time).

4.7.2.1 Send_045_SC_NA, Send_058_SC_TA

Handling of a single command.

Name	Data Type	Attributes	Direction	Initial Value
CmdOn	bool	retain	in	false
CmdOff	bool	retain	in	false

A command is sent when one of these inputs becomes true, unless the previous command is still running. If ***CmdOn*** is true the "Single Command State" field in the message is set to 1 (on), if ***CmdOff*** is true the "Single Command State" field in the message is set to 0 (off).

4.7.2.2 Send_046_DC_NA, Send_059_DC_TA

Handling of a double command.

Name	Data Type	Attributes	Direction	Initial Value
CmdOn	bool	retain	in	false
CmdOff	bool	retain	in	false

A command is sent when one of these inputs becomes true, unless the previous command is still running. If ***CmdOn*** is true the "Double Command State" field in the message is set to 1 (on), if ***CmdOff*** is true the "Double Command State" field in the message is set to 2 (off).

4.7.2.3 Send_050_SE_NC, Send_063_SE_TC

Handling of a setpoint.

Name	Data Type	Attributes	Direction	Initial Value
Send	bool	retain	in	false
SPValue	real	retain	in	0.0

The setpoint is sent when the ***Send*** pin becomes true, unless the previous setpoint command is still running. Make sure the value is set to the ***SPValue*** pin before or in the same task cycle the ***Send*** pin is set to true.

4.7.2.4 Send_051_BO_NA, Send_064_BO_TA

Handling of a setpoint.

Name	Data Type	Attributes	Direction	Initial Value
Send	bool	retain	in	false
BOValue	dword	retain	in	0

The setpoint is sent when the ***Send*** pin becomes true, unless the previous setpoint command is still running. Make sure the value is set to the ***BOValue*** pin before or in the same task cycle the ***Send*** pin is set to true.

4.7.3 Redundant Output handling for Command types

This set of function blocks handles command output for a pair of redundant CI modules. The function blocks are similar to the same function blocks for single command output. In fact each of those function blocks encapsules two normal command blocks.

The redundant function blocks are named with an additional "_Dual" in the function block name.

These function blocks access several external variables that must be defined, in addition to those for the single command function blocks:

Name	Data Type	Attrib-utes	Description
IEC60870_Cmd_ResendTime	time		Time within which to hold command for redundancy switchover.

The following pins are the same for all the function blocks:

Name	Data Type	Attributes	Direction	Initial Value
Enable	bool	retain	in	
Id1	Comm_Channel		in	
Id2	Comm_Channel		in	
CommonAddress	dint	retain	in	
InformationOb-jectAddress	dint	retain	in	
Active1	bool	retain	in	
Active2	bool	retain	in	
Ok	bool	retain	out	false
Error	bool	retain	out	false
Status	bool	retain	out	false

The redundant command output function blocks are for redundant command output via a pair of redundant CI boards. ***ID1*** and ***ID2*** are connected to the ***ID*** pins of the two Connect function blocks. ***Active1*** and ***Active2*** indicate which of the two connections is currently active.

The command is output on the active connection. If the command is not executed (acknowledged by the IEC partner) and a switchover occurs before the ResendTime elapses, then the command is output on the other connection.

Otherwise this set of function blocks is the same as the single command blocks.

4.7.4 Output handling for Select/Execute Command types

These function blocks handle sending a command and receiving and decoding the response message from the partner for Select/Execute commands.

These function blocks are named with an additional "_SE" in the function block name and they behave the same as the regular ones for direct execute commands, except with the following additions:

The function blocks have the following extra pins:

Name	Data Type	Attributes	Direction	Initial Value
Execute	bool	retain	in	false
Selected	bool	retain	out	false

When the pins that normally execute the command are set to true (the ***CmdOn*** pin) these function blocks send a Select command, rather than an Execute command. The actual Execute command is sent by setting the ***Execute*** pin to true.

The ***Selected*** pin becomes true when the confirmation message to the Select command has been received. If the command cannot be selected (e.g. because another S/E command is already in progress) the ***Error*** pin will become true instead.

4.7.4.1 DetectGATerm

This function block detects the end (termination message) of a General Acquisition. This usually signifies that all the data from this slave station has been sent. This block is usually used to enable the Receive blocks only after the GA is finished to insure that the Receive blocks do not report out-of-date data.

The function block only operates if the ***Enable*** pin is true. The pins ***Id*** and ***CommonAddress*** are connected directly to the Read function blocks inside.

Partner gives the HW position of the IEC Partner.

Timeout is the time to wait for the completion of the GA.

Start is set to true to indicate that the GA is supposed to be running. Normally this is connected to the output of the PartnerStatus block.

The ***Ok*** pin is set to true once the termination message of the GA has been received.

The ***Error*** pin is set to true if the termination message is not received during the specified timeout time. The ***Error*** is only cleared by setting the ***Start*** pin to false. If the termination of the GA is received after the timeout time has elapsed, then both the ***Ok*** and ***Error*** pins will be true.

4.7.5 Functions for handling counter values:

4.7.5.1 Make_CntReq_Freeze_Read

This function implements a sequence to freeze and read counter values from a slave station using Mode C (freeze and transmit by counter interrogation commands) by sending first a Freeze Request and then a Read Request for the specified counter group.

A rising flank at the ***Start*** pin starts the sequence. The value on the ***CntGroup*** pin specifies which counter group to request, use 1 to 4 request that group, any other value will perform a general counter request. The value of the ***CntGroup*** pin is read once at the beginning of the sequence.

PauseTime gives the time to wait between the Freeze Request and the Read Request.

AckWaitTime gives the maximum time to wait for the termination message to the current request.

ReadTime gives the pulse duration of the pulse on the ***Read_Enable*** pin.

Send_OK and ***Send_Error*** are used to provide feedback from the Send_101_CI_NA block; they should be connected to the ***Ok*** and ***Error*** pins of that function block, respetively.

Send_Send_Req and ***Send_Cnt_Reqest*** are used to control the Send_101_CI_NA block; they should be connected to the ***Send_Req*** and ***Cnt_Request*** pins of that function block, respetively.

Read_Enable is pulsed when the counter values should be read; this pin is usually connected to the ***Enable*** pin of the relevant IEC60870Receive blocks.

The ***Ok*** pin is set to true if the counter request completed successfully.

The ***Error*** pin is set to true for one cycle if an error occured.

The ***Running*** pin is true while the counter request sequence is running.

4.7.5.2 Send_101_CI_NA

This function block handles sending a counter request and receiving and decoding the response message from the partner.

Pins of this function block:

Name	Data Type	Attributes	Direction	Initial Value
Enable	bool	retain	in	
Id	Comm_Channel		in	
CommonAddress	dint	retain	in	
CommonAddressResponse	dint	retain	in	
InformationObjectAddress	dint	retain	in	0
Send_Req	bool	retain	in	false
Cnt_Request	dint	retain	in	0
AckWaitTime	time	retain	in	time#20s
Ok	bool	retain	out	false
Error	dint	retain	out	false
Running	bool	retain	out	false
Status	bool	retain	out	1

The function block only operates if the ***Enable*** pin is true. The pins ***Id***, ***CommonAddress***, ***CommonAddressResponse*** and ***InformationObjectAddress*** are connected directly to the Read and Write function blocks inside. The ***InformationObjectAddress*** is normally 0 according to the IEC spec. The seperate Common Address for the Response is to allow the counter request to be sent to the broadcast address (65535) while receiving the response from the actual slave address.

The request is sent on the rising edge of ***Send_Req***.

Cnt_Req gives the Qualifyer for the counter request (see IEC spec).

AckWaitTime gives the time to wait for the acknowledgement to the counter request. This time has to be long enough to allow for the transmission of all counter values as the final termination message is sent after the counter values have been sent. If the acknowledgement is not received in the specified time, an error is reported.

The ***Ok*** pin is set to true once the termination message of the Counter Interrogation has been received.

The ***Error*** pin is set to true for one task cycle if the termination message is not received during the specified timeout time.

The ***Running*** pin is true while the Counter Interrogation is in progress. It is not possible to send another Counter Interrogation until the previous one is finished.

The ***Status*** pin indicates success or failiure of the command. It is 0 while the command is running. It is set to 1 if the command completes successfully. A negative value indicates an error, –1 indicates that no answer was received from the partner, –2 that the partner sent a negative acknowledge, other negative values indicate that the function blocks that actually send or receive data reported an error.

4.8 Software configuration with IEC60870SupLib:

This library is deprecated and supplied mainly for compatibility with SV3 and SV4 to simplify upgrading an existing SV3 or SV4 project to SV5 and it should not be used in new projects in SV5. Many of the function blocks that were originally in this library have been moved to one of the other libraries, this library contains those function blocks that have been replaced by incompatible function blocks.

This library contains function blocks that build on the function blocks from IEC60870CommLib to offer additional functionality or that add useful functions.

This library contains the following function blocks:

Function Block	Description
ConvM_009_ME_NA	Convert simple types to a structured variable (IEC ASDU type 7).
ConvM_030_SP_TB	Convert simple types to a structured variable (IEC ASDU type 30).
ConvM_031_DP_TB	Convert simple types to a structured variable (IEC ASDU type 31).
ConvM_036_ME_TF	Convert simple types to a structured variable (IEC ASDU type 36).
ConvMS_030_SP_TB	Convert simple types to a structured variable (IEC ASDU type 30).
ConvMS_031_DP_TB	Convert simple types to a structured variable (IEC ASDU type 31).
ConvMS_036_ME_TF	Convert simple types to a structured variable (IEC ASDU type 36).
ConvMS_037_IT_TB	Convert simple types to a structured variable (IEC ASDU type 37).
Decode_DP	Decode an IEC DP value into two bool variables.
Decode_DP1	Decode an IEC DP value into three bool variables.
DelayBool	Delay a bool signal for one task cycle.

For a detailed description of the function blocks in this library, see the configuration guide for SV4.

4.9 Software configuration with IEC60870RcvLib:

This library is deprecated and supplied mainly for compatibility with SV3 and SV4 to simplify upgrading an existing SV3 or SV4 project to SV5 and it should not be used in new projects in SV5. The function blocks that were originally in this library have been replaced by incompatible function blocks in IEC60870MasterLib.

This library contains function blocks that build on the function blocks from IEC60870CommLib to offer additional functionality or that add useful functions.

This library contains the following function blocks:

Function Block	Description
RcvM_030_SP_TB	Convert a structured variable (IEC ASDU type 30) to simple data types.
RcvM_031_DP_TB	Convert a structured variable (IEC ASDU type 31) to simple data types.
RcvM_036_ME_TF	Convert a structured variable (IEC ASDU type 36) to simple data types.

For a detailed description of the function blocks in this library, see the configuration guide for SV4.

4.10 Example Application

This is a small demo application that implements an IEC Master and Slave with one controller and two CI864 boards. This example is in structured text, mainly because that is easier to use in this document in a readable form. For somebody with experience in programming 1131 applications, it should not be a problem to translate the program into FBD.

For the HW only the parameters that have a value different from the default value are listed here.

4.10.1 Master

The Master uses the CI864 board at position 1; the 1131 program is called "Program4_IEC_Master" and runs in the Normal task (250ms).

4.10.1.1 Hardware - 1 CI864

Parameter	Value	Comment
Primary IP Address	172.18.100.1	IP address of this CI864 board.
IP Subnet Mask	255.255.0.0	

4.10.1.2 Hardware - 1.1 IEC60870 Partner

Parameter	Value	Comment
Partner primary IP Address	172.18.100.61	IP address of the slave.
Initiating communication	true	
Control connection	true	
Common address	356	Common address of the master, same as variable CA_Own in the 1131 program.
Use local time	true	Local time is used in the IEC 104 messages.
Report invalid time	true	
Status conversion	OPC_STD	Should usually be set to this value, the Encode/Decode function blocks in the supplied libraries assume this setting.
Data type translation	BOTH	Use this setting if GA is handled according to the IEC spec (data items sent without timestamp during GA).
Request GA	ACTIVATE	Request a GA when the connection is established or becomes active.
Use delay buffer	true	To ensure short signal pulses sent from the slave are not lost.
Msg delay	600	A bit more than twice the task time (Normal task with 250 ms cycle time used here).

4.10.1.3 Hardware - 1.1.1 IEC60870 Station

Parameter	Value	Comment
Partner Common Address	15716	Common address of the slave, same as variable CA_Partner in the 1131 program.

4.10.1.4 Variables

Name	Data Type	Attributes	Initial Value
Step_IOA3	dint	retain	
CA_Own	dint	retain	
CA_Partner	dint	retain	
IEC_Enable	bool	coldretain	true
IEC_Id	Comm_Channel	retain	
IEC_Ok	bool	retain	
IEC_Conn_Ok	bool	retain	
IEC_In_Ena	bool	retain	
IEC_Cmd_Out_Ena	bool	retain	
MeasVal1	RealIO	retain	
MeasVal2	RealIO	retain	
BreakerOn	BoolIO	retain	
BreakerOff	BoolIO	retain	
MeasVal1_IEC	MSG_IEC101_036_M_ME_TF_Type	retain	
MeasVal2_IEC	MSG_IEC101_036_M_ME_TF_Type	retain	
Breaker_IEC	MSG_IEC101_031_M_DP_TB_Type	retain	
BreakerCmdOn	bool	retain	
BreakerCmdOff	bool	retain	

4.10.1.5 Function blocks

Name	Function Block Type	Task Connection	Description
Calc_Step_IOA3	Conv_IEC_IOA		
Calc_CA_Own	Conv_IEC_CA		
Calc_CA_Partner	Conv_IEC_CA		
IECConnect	ConnectIEC60870		
IECPartnerStatus	PartnerStatus		
IECDetectGaTerm	DetectGATerm		
Calc_MeasVal_IOA	Conv_IEC_IOA		
Read_MeasVal	IEC60870Receive[2]		
Decode_MeasVal	Decode_036_ME_TF[2]		
Calc_Breaker_IOA	Conv_IEC_IOA		
Read_Breaker	IEC60870Receive[1]		
Decode_Breaker	Decode_031_DP_TB[1]		
Calc_BreakerCmd_IOA	Conv_IEC_IOA		
Send_BreakerCmd	Send_059_DC_TA		

4.10.1.6 Code

```
(* **** Calculate Step Sizes and Common Addresses *)
Calc_Step_IOA3( IOA1 := 0,
                IOA2 := 0,
                IOA3 := 1,
                IEC_Addr => Step_IOA3 );

Calc_CA_Own( CA1 := 100,
             CA2 := 1,
             IEC_Addr => CA_Own );
Calc_CA_Partner( CA1 := 100,
                 CA2 := 61,
                 IEC_Addr => CA_Partner );

(* **** Set up connection, enable data transfer, ... *)
IECConnect( En_C := IEC_Enable,
            CIPos := 1,
            IEC60870PartnerPos := 1,
            Valid => IEC_Ok,
            Id := IEC_Id );

IECPartnerStatus( Enable := IEC_Ok,
                  IEC60870PartnerPos := 1,
                  Id := IEC_Id,
                  Status1 => IEC_Conn_Ok );

IECDetectGaTerm( Enable := IEC_Ok,
                 Id := IEC_Id,
                 CommonAddress := CA_Partner,
                 Partner := 1,
                 Start := IEC_Conn_Ok,
                 Ok => IEC_In_Ena );
IEC_Cmd_Out_Ena := IEC_Conn_Ok;


(* ***** Receive Measurements *)
Calc_MeasVal_IOA( IOA1 := 36,
                  IOA2 := 1,
                  IOA3 := 1 );

Read_MeasVal( Enable := IEC_In_Ena,
              Id := IEC_Id,
              CommonAddress := CA_Partner,
              InformationObjectAddr := Calc_MeasVal_IOA.IEC_Addr,
              InfoObjAddr_StepSize := Step_IOA3,
              Rd[1] := MeasVal1_IEC,
              Rd[2] := MeasVal2_IEC );

Decode_MeasVal( ConnValid := IEC_Conn_Ok,
                RcvValid := Read_MeasVal.Valid,
                UpdateOut := Read_MeasVal.NewData,
                InVal[1] := MeasVal1_IEC,
                OutVal[1] := MeasVal1,
                InVal[2] := MeasVal2_IEC,
                OutVal[2] := MeasVal2 );

(* ***** Receive Breaker *)
Calc_Breaker_IOA( IOA1 := 31,
                  IOA2 := 1,
                  IOA3 := 1 );

Read_Breaker( Enable := IEC_In_Ena,
              Id := IEC_Id,
              CommonAddress := CA_Partner,
              InformationObjectAddr := Calc_Breaker_IOA.IEC_Addr,
              InfoObjAddr_StepSize := Step_IOA3,
              Rd[1] := Breaker_IEC );

Decode_Breaker( ConnValid := IEC_Conn_Ok,
                RcvValid := Read_Breaker.Valid,
                UpdateOut := Read_Breaker.NewData,
                InVal[1] := Breaker_IEC,
                OutValOn[1] := BreakerOn,
                OutValOff[1] := BreakerOff );

(* ***** Send Breaker command *)
Calc_BreakerCmd_IOA( IOA1 := 59,
                     IOA2 := 1,
                     IOA3 := 1 );

Send_BreakerCmd( Enable := IEC_Cmd_Out_Ena,
                 Id := IEC_Id,
                 CommonAddress := CA_Partner,
                 InformationObjectAddr := Calc_BreakerCmd_IOA.IEC_Addr,
                 CmdOn := BreakerCmdOn,
                 CmdOff := BreakerCmdOff );
```

4.10.2 Slave

The Slave uses the CI864 board at position 2; the 1131 program is called "Program5_IEC_Slave" and runs in the Normal task (250ms).

4.10.2.1 Hardware - 2 CI864

Parameter	Value	Comment
Primary IP Address	172.18.100.61	IP address of this CI864 board.
IP Subnet Mask	255.255.0.0	

Use common listentask	true	Usually set to true in any CI864 board that acts as the listener (Server).

4.10.2.2 Hardware - 2.1 IEC60870 Partner

Parameter	Value	Comment
Partner primary IP Address	172.18.100.1	IP address of the master.
Initiating communication	false	
Control connection	false	
Common address	15716	Common address of the slave, same as variable CA_Own in the 1131 program.
Use local time	true	Local time is used in the IEC 104 messages.
Report invalid time	false	
Status conversion	OPC_STD	Should usually be set to this value, the Encode/Decode function blocks in the supplied libraries assume this setting.
Data type translation	BOTH	Use this setting if GA is handled according to the IEC spec (data items sent without timestamp during GA).
Request GA	NONE	Slave does not receive GA data.
Use delay buffer	false	Slave does not receive signals that use this buffer.

4.10.2.3 Hardware - 1.2.1 IEC60870 Station

Parameter	Value	Comment
Partner Common Address	356	Common address of the master, same as variable CA_Partner in the 1131 program.

4.10.2.4 Variables

Name	Data Type	Attributes	Initial Value
DataInitTime	time	retain	time#1s
Step_IOA3	dint	retain	
CA_Own	dint	retain	
CA_Partner	dint	retain	
IEC_Enable	bool	coldretain	true
IEC_Id	Comm_Channel	retain	
IEC_Id2	Comm_Channel	retain	
IEC_Ok	bool	retain	
IEC_Conn_Ok	bool	retain	
IEC_Out_Ena	bool	retain	
IEC_Out_Update	bool	retain	
IEC_Cmd_In_Ena	bool	retain	
MeasVal1	real	retain	
MeasVal2	real	retain	
MeasVal3	real	retain	
MeasVal4	real	retain	

BreakerOn	bool	retain	
BreakerOn2	bool	retain	
MeasVal1_IEC	MSG_IEC101_036_M_ME_TF_Type	retain	
MeasVal2_IEC	MSG_IEC101_036_M_ME_TF_Type	retain	
Breaker_IEC	MSG_IEC101_031_M_DP_TB_Type	retain	

4.10.2.5 Function blocks

Name	Function Block Type	Task Connection	Description
Calc_Step_IOA3	Conv_IEC_IOA		
Calc_CA_Own	Conv_IEC_CA		
Calc_CA_Partner	Conv_IEC_CA		
IECConnect	ConnectIEC60870		
IECPartnerStatus	PartnerStatus		
IEC_DataDelay	DataDelay		
Calc_MeasVal_IOA	Conv_IEC_IOA		
Encode_MeasVal	Encode_036_ME_TF_BNIO[2]		
Send_MeasVal	IEC60870WriteCont[2]		
EncWrite_MeasVal	EncWrite_013_036_ME_NC_TF_NIO[2]		
Calc_Breaker_IOA	Conv_IEC_IOA		
Encode_Breaker	Encode_031_DP_TB_BNIO[1]		
Send_Breaker	IEC60870WriteCont[1]		
EncWrite_Beaker2	EncWrite_003_031_DP_NA_TB_NIO[1]		
Calc_Break-erCmd_IOA	Conv_IEC_IOA		
Rcv_BreakerCmd	Rcv_059_DC_TA		
CmdRcv_Breaker2	CmdRcv_046_059_DC_NA_TA		

4.10.2.6 Code

```
(* **** Calculate Step Sizes and Common Addresses *)
Calc_Step_IOA3( IOA1 := 0,
                IOA2 := 0,
                IOA3 := 1,
                IEC_Addr => Step_IOA3 );

Calc_CA_Own( CA1 := 100,
             CA2 := 61,
             IEC_Addr => CA_Own );
Calc_CA_Partner( CA1 := 100,
                 CA2 := 1,
                 IEC_Addr => CA_Partner );

(* **** Set up connection, enable data transfer, ... *)
IECConnect( En_C := IEC_Enable,
            CIPos := 2,
            IEC60870PartnerPos := 1,
            Valid => IEC_Ok,
            Id := IEC_Id );

IECPartnerStatus( Enable := IEC_Ok,
                  IEC60870PartnerPos := 1,
                  Id := IEC_Id,
                  Status1 => IEC_Conn_Ok );

IEC_DataDelay( Enable := IEC_Ok,
               DataDelay := DataInitTime,
               DataInitTime := DataInitTime,
               DataEnable1 => IEC_Out_Ena,
               DataEnable1New => IEC_Out_Update );
IEC_Cmd_In_Ena := IEC_Conn_Ok;

(* ***** Send Measurements *)
Calc_MeasVal_IOA( IOA1 := 36,
                  IOA2 := 1,
                  IOA3 := 1 );

Encode_MeasVal( UpdateOutput := IEC_Out_Update,
                InVal[1] := MeasVal1,
                Hys[1] := 1.0,
                OutVal[1] := MeasVal1_IEC,
                InVal[2] := MeasVal2,
                Hys[2] := 2.0,
                OutVal[2] := MeasVal2_IEC );

Send_MeasVal( Enable := IEC_Out_Ena,
              Id := IEC_Id,
              CommonAddress := CA_Own,
              InformationObjectAddr := Calc_MeasVal_IOA.IEC_Addr,
              InfoObjAddr_StepSize := Step_IOA3,
              ChangedValues := Encode_MeasVal.ChangedValues,
              Sd[1] := MeasVal1_IEC,
              Sd[2] := MeasVal2_IEC );

(* ***** Send Measurements using a combined Encode/Write block*)
EncWrite_MeasVal( Enable1 := IEC_Out_Ena,
                  Id1 := IEC_Id,
                  Enable2 := FALSE,
                  Id2 := IEC_Id2,
                  CommonAddress := CA_Own,
                  IOA1 := 36,
                  IOA2 := 2,
                  IOA3 := 1,
                  InfoObjAddr_StepSize := Step_IOA3,
                  UseTimeStamp := TRUE,
                  UpdateOutput := IEC_Out_Update,
                  InVal[1] := MeasVal3,
                  HysA[1] := 2.0,
                  InVal[2] := MeasVal4,
                  HysA[2] := 5.0,
                  HysI[2] := 10.0 );

(* ***** Send Breaker *)
Calc_Breaker_IOA( IOA1 := 31,
                  IOA2 := 1,
                  IOA3 := 1 );

Encode_Breaker( UpdateOutput := IEC_Out_Update,
                InValOn[1] := BreakerOn,
                InValOff[1] := NOT BreakerOn,
                OutVal[1] := Breaker_IEC );

Send_Breaker( Enable := IEC_Out_Ena,
              Id := IEC_Id,
              CommonAddress := CA_Own,
              InformationObjectAddr := Calc_Breaker_IOA.IEC_Addr,
              InfoObjAddr_StepSize := Step_IOA3,
              ChangedValues := Encode_Breaker.ChangedValues,
              Sd[1] := Breaker_IEC );

(* ***** Send Breaker 2 using a combined Encode/Write block*)
EncWrite_Beaker2( Enable1 := IEC_Out_Ena,
                  Id1 := IEC_Id,
                  Enable2 := FALSE,
                  Id2 := IEC_Id2,
                  CommonAddress := CA_Own,
                  IOA1 := 31,
                  IOA2 := 2,
                  IOA3 := 1,
                  InfoObjAddr_StepSize := Step_IOA3,
                  UseTimeStamp := TRUE,
                  UpdateOutput := IEC_Out_Update,
                  InValOn[1] := BreakerOn2,
                  InValOff[1] := NOT BreakerOn2 );

(* ***** Receive Breaker command *)
Calc_BreakerCmd_IOA( IOA1 := 59,
                     IOA2 := 1,
                     IOA3 := 1 );

Rcv_BreakerCmd( Enable := IEC_Cmd_In_Ena,
                Id := IEC_Id,
                CommonAddress := CA_Own,
                InformationObjectAddr := Calc_BreakerCmd_IOA.IEC_Addr,
                Blocked := FALSE,
                CmdAutoAck := TRUE );

IF (Rcv_BreakerCmd.CmdOn)  THEN BreakerOn := TRUE;  END_IF;
IF (Rcv_BreakerCmd.CmdOff) THEN BreakerOn := FALSE; END_IF;

(* ***** Receiver Breaker 2 command using a combined command block*)
CmdRcv_Breaker2( Enable1 := IEC_Out_Ena,
                 Id1 := IEC_Id,
                 Enable2 := FALSE,
                 Id2 := IEC_Id2,
                 CommonAddress := CA_Own,
                 IOA1 := 59,
                 IOA2 := 2,
                 IOA3 := 1,
                 UseTimeStamp := TRUE,
                 UseSelectExecute := FALSE );

IF (CmdRcv_Breaker2.CmdOn)  THEN BreakerOn2 := TRUE;  END_IF;
IF (CmdRcv_Breaker2.CmdOff) THEN BreakerOn2 := FALSE; END_IF;
```

5 Section 5 - Advanced Topics

This section addresses several advanced topics. It is not possible to discuss these topics exhaustively since there are many different ways to address them in the context of any specific application.

We (ABB Austria) can provide support to address these topics when needed, but we will usually need to charge extra for this if the required support requires more work than answering a few questions via telephone or email.

5.1 Redundancy

Redundancy can be implemented in a lot of different ways, it is not possible to describe all of them here; this chapter can only give an overview of some of the possible tools the CI864 has to be used in redundant configurations. How the redundancy is implemented usually depends on the priorities and philosophy of the customer and the capabilities of the system at the other end of the IEC connection.

The CI864 does not impelment the normal CI board redundancy, but it provides a number of tools to support redundancy, so that a fully redundant system can be impemented with application support. The appropriate configuration has to be determined on a case by case basis as the IEC partner systems will have different requirements.

Controller redundancy is supported and is transparent to the CI864 board, that is it makes no difference for the IEC connection redundancy if a redundant PM8xx is used or not or if a BC810 CEX-Bus splitter is used.

The data transfer in a redundant setup is normally done in such a way that there is always exactly one active connection that transfers data. This can be achieved either by sending the data only on one connection or by sending the data on both connections and only using the data from one connection at the receiving end.

In a full redundancy setup with a total of 4 connections, these two options are usually combined (the active slave sends data to both masters, and only the active master processes the data).

The data transfer can be defined differently depending on the data type.

Monitoring data can be handled in two different ways on the passive connection on the receiving side. Either the CI864 board does not process the data or the CI864 board does process the data but it is not read (the Receive blocks are not enabled).

In the first case, the HW Parameter "Request GA" would be set to "ACTIVE" and the Receive blocks should only be enabled after the GA has been performed.

In the second case, the HW Parameter "Request GA" would be set to "CONNECT". In this case the passive connection has up-to-date data and the Receive blocks can be enabled right after switch-over.

Each IEC ASDU type is defined to use one of 4 different redundancy types in the Conversion List. The types are Monitoring and Command (the behavior of these two types is defined by the HW parameter Redundancy), Active (sent/received on the active connection) and System (always sent/received).

When sending data a monitoring data type that is not sent (because the connection is not active) is still entered into the data cross reference so it is available for a GA. Data types that are not entered into the cross reference (commands) are silently discarded.

When receiving data, data types that are not to be handled are silently discarded on the passive connection.

The effect of the redundancy type from the Conversion List and the HW parameter Redundancy on a passive connection is as follows:

	NONE	STANDBY	SHADOW	REVERSED	DUAL	INCOMING	OUTGOING
System	Y	Y	Y	Y	Y	Y	Y
Active							

Incoming Monitoring	Y			Y	Y	Y	
Incoming Commands	Y		Y			Y	
Outgoing Monitoring	Y		Y				Y
Outgoing Commands	Y			Y			Y

"Y" means that a message of this type is processed, otherwise it is silently discarded.

The active connection always handles all messages. Messages that are sent in response to another message use the settings of the original message

The CI864 does not currently implement any buffers to buffer events during a break in the communication or on a passive connection. Events will be lost during a break in the communication or a redundancy switchover. It is only guaranteed that the state of the system will be consistent after the switchover or initialization is complete, this usually means after the General Interrogation has completed.

5.1.1 Simple System

The trivial case, no redundancy of any kind is implemented. A single CI864 connects to a single IEC partner. The following HW settings are suggested:

Redundancy NONE, the CI board is always active

Request GA CONNECT or ACTIVATE, if we receive monitoring data to request a GA when the connection is established. NO if we implement a pure slave and don't receive monitoring data.

BG Data Send NO; CONNECT or ACTIVATE if we send monitoring data and the IEC partner does not request a GA when the connection is established.

5.1.2 One CI864 connected to a redundant Partner

In this case a single CI864 connects to two different partners. A single IEC Partner object is defined with two Partner IP addresses. The IEC Partner object can handle the redundancy switchover automatically, unless the default switchover logic is not appropriate, for example if the secondary connection is slower and should always be used only as backup. In this case the active connection has to be determined by the application and set using the SetPartnerActive function block.

Possible values for the Redundancy parameter:

NONE No redundancy, that is both connections are always fully active. This is usually paired with the redundancy type STANDBY on the other side.

STANDBY The passive connection does not handle any data. This is usually paired with redundancy type NONE.

SHADOW Monitoring data is sent on both connections and only accepted by the active connection. Command are only sent on by the active side but accepted by both connections. This is paired with itself, that is SHADOW redundancy is selected on both sides.

REVERSED Monitoring data is sent on the active connection and accepted on both connections. Commands are sent on both connections and accepted by the active side. This is paired with itself and is basically the opposite of SHADOW.

INCOMING Data is sent on both connections and only accepted by the active connection. This is paired with itself, that is INCOMING redundancy is selected on both sides.

OUTGOING Data is sent on the active connections and accepted by both connections. This is paired with itself, that is OUTGOING redundancy is selected on both sides.

DUAL is usually used in systems with redundant CI boards.

5.1.3 Redundant CI864 boards

Two CI864 boards are used as a redundant pair. It does not usually make a big difference for the configuration if the other side is redundant as well or not (for example, a RedundancyDual function block would be used if the other side is redundant, otherwise a Redundancy block would be used) as this is mostly handled internally in the CI864 board. As the two CI864 boards do not know anything about each other, the active board has to be determined by the application and set using the SetPartnerActive function block.

Possible values for the Redundancy parameter:

NONE, SHADOW, REVERSED and OUTGOING would be used only if the other side is not redundant, much as in the case above.

STANDBY The passive connection does not handle any data. If the other side is not redundant, it would use the redundancy type NONE, if the other side is redundant, it would also use redundancy type STANDBY.

DUAL The passive board only handles incoming monitoring data. To keep the incoming monitoring data consistent, only the Receive blocks of the active CI864 board are enabled. This enables a faster switchover because the passive CI864 board processes the data and has it ready to be read by the Receive blocks. This setting would be paired with either itself or INCOMING.

INCOMING This is similar to DUAL, except that commands are also processed by the receiving side. This is usually not a problem as the output of the two function blocks that receive commands are "or-ed" anyway.

The difference between the DUAL and INCOMING are that the priority for commands is different. With DUAL, it is possible that a command is lost during switchover, with INCOMING it is possible that commands are executed twice during switchover.

5.1.4 Data Consistency of received data

The protocol handler for the CI864 keeps a buffered list of all the received data items for each IEC Partner. This list is queried by the Receive blocks; the information that matches the IOA addresses for this Receive block is reported to the 1131 application. When new data is received for a data item, the information in this list is overwritten and the new data is reported to the Receive block.

This list is cleared when the connection to the IEC partner object in the CI864 block is broken (the Connect block is disabled or goes invalid because of an error); the list is not cleared when the connection to the partner is lost.

This is not a problem in a simple system or when one CI864 board connects to two partners (the two connections of one IEC Partner share a buffer). In this case the Receive blocks will either not report any data or will report the latest received data.

When redundant CI864 boards are used, it is possible that the buffer of the passive connection contains outdated data; this depends on the chosen redundancy scheme.

There are several methods that can be used to address this problem and to ensure that only "current" information is reported to the 1131 application. This is only relevant for monitoring data types that use General Interrogation / spontaneous transmission or cyclic transmission. Queried data (counters) and commands use different schemes to address this problem.

Sequence of events during switchover:

- Normal operation; both connections are working, connection A is active. Connection B is working, but it is not considered to have current data.
- An error is detected on connection A and switchover is performed. Connection A is set to passive (SetPartnerActive function block).
- Connection B is set to active (SetpartnerActive) function block. This causes a general interrogation to be requested from the partner (Request GA is set to ACTIVATE).
- Wait for the end of the general interrogation (DetectGATerm function block). When the end of the general interrogation is received, the data is considered to be consistent and the Receive function blocks are enabled.

If the passive connection is assumed to have current data (Redundancy is set to DUAL or INCOMMING, Request GA is set to CONNECT and the partner sends data on all connections), then the Receive blocks can be enabled right after the connection is set to active. It is still a good idea to enable them only after the end of a GA has been detected (in this case right after the connection is established; not when it becomes active) or it has been ensured by some other means that all data items have been sent on the connection.

This logic will work in many cases, but there are some possible problems:

- Some partners will send an "end of general interrogation" message in response to a GA Request but will not send any actual data when they are passive. In this case the DetectGA-Term function block can not be considered a reliable indicator that the data on the connection is current.
- Some data items are not sent during a GA, but only sent spontaneously. This is typically not a problem for IEC-101 or -104, but it is possible in -103. These data items may still have obsolete data after the GA. This usually includes data items for which only one flank is sent; see "Use Second Messages for one flank items" in the Options setting of a Station.
- The GA may take a long time to execute and any message pulses (one data item sent with "on" status and then with an "off" status) will be ignored. This is typically not a problem with the IEC-104 protocol since data transfer is fast. This can be a big problem with -101 and -103, especially in a party-line configuration. This is especially a problem if important messages are buffered (e.g. a Breaker Trip) during a break in the communication and sent right after the connection is established (during the GA).

The "Version 2" Decode blocks (name ends with "_V2") are designed to address these problems. They are not necessarily "better" than the normal blocks, but they use a different method to insure data consistency. The V2 Decode blocks are suited for redundancy schemes where the second connection does not have current data (such as Line Sharing Redundancy) as this method always assumes data on the passive line to be outdated.

The V2 Decode blocks use a Compare Buffer to check for data changes.

Sequence of events during switchover:

- Normal operation; both connections are working, connection A is active. Connection B may be working, but it is not considered to have current data.
- An error is detected on connection A and switchover is performed. Connection A is set to passive (SetPartnerActive function block).
- The Receive blocks of connection B are activated. The status of the connection will be reported as "bad" to the Decode blocks (***ConnValid*** pin is FALSE).
  Any data reported by the Receive blocks at this time will be stored in the Compare Buffer of the Decode blocks, but will be considered outdated and will not be reported to the rest of the 1131 application. There should be a short delay before the next step (at least two task cycles) to ensure that all Receive blocks report their data to the Decode blocks and the Decode blocks process the data.
- Connection B is set to active (SetpartnerActive) function block. The status of the connection will be reported as "good" to the Decode blocks (***ConnValid*** pin is TRUE) as long as it is actually "good" (when using Line Sharing Redundancy the connection can only become "good" after it is set active). From this point on all data received by the Decode blocks will be considered current and will be reported to the 1131 application.
  This causes a general interrogation to be requested from the partner (Request GA is set to ACTIVATE).
  If a data item is not sent after the switchover (e.g. because it is only sent spontaneously), it will still contain the data from the last time it was received (or it will remain "bad").

In both methods there is a time window during switchover during which spontaneously sent data will be discarded.

There are other possible methods to ensure data consistency; at the moment only the two methods described above are implemented. It is possible to implement other methods if needed to meet specific customer requirements. Depending on the specifics of the method, it could be possible to implement them with customized Decode blocks or it could require changes to the CI864 firmware or Protocol Handler.

5.2 Library Design

The standard libraries supplied with the 800xA system are not prepared for remote communication. This is acceptable in a plant where there is relatively little external communication (specifically using the IEC 60870-5-104 protocol, but also other similar protocols), but when there is a lot of such communication, this leads to loss of functionality and/or requires a lot of extra work to implement workarounds to add the required functionality outside the "standard" library objects.

5.2.1 Example:

A breaker in a substation: The breaker is represented by a UniSimpleM control module in the substation. There is another such module in the central station, from where the same breaker can be controlled remotely.

Communication between the two stations is via the IEC 60870-5-104 protocol. The state of the breaker is represented by a data item of the IEC type 31 (dual pole with timestamp), the commands from the main station are sent as a command of type 46 (dual pole command) and there extra data items of type 30 or 31 associated with the breaker, such as an error indicator and control location (is the breaker in local or remote control).

5.2.1.1 Remote Control:

The first problem is how to handle commands in remote mode. Remote mode means that the breaker is controlled from the central station, rather than from the substation. Local mode can have two different meanings, depending on the context; either control from the hardwired buttons (LocMode) or control from any location in the substation. Typically a switch or lock is used in the substation to determine if the breaker is controlled from the substation or from the central station.

First, the UniSimpleM control module does not have a "Remote" mode. If there is no panel control, then it would be possible to "abuse" the panel mode for remote control. If panel mode (and auto mode) is already used, the remote mode has to be "piggybacked" on one of those two modes: when the breaker is in remote control, it is switched to AutoMode, but the usual AutoCmd signals from logic are disabled. Conversely, when the breaker is in actual AutoMode (controlled from logic in the substation) then it is also in AutoMode, but the AutoCmd signals from the IEC program are disabled. A problem with this solution is that it isn't apparent in the faceplate if the breaker is under logic control or if it is controlled from the central station.

A solution for this problem would be to create an extended version of UniSimpleM (or UniM) that includes RemExists, RemMode, RemCmd0 and RemCmd1 inputs on the control module plus logic to handle them and the indicators to display the mode in the faceplate and graphic elements.

The whole problem is much simpler in the central station, whenever the breaker can not be controlled from the central station it is in LocMode.

There is a second problem in the central station:

Executing any command over the IEC connection takes a certain amount of time. It is not possible to give a second command (for the same data item) until the first command is finished (or it can lead to problems). Ideally the UniSimpleM control module should have an input that is connected to the Running output pin of the Send_xxx function block that sends the command. This input should delay output of further commands until the first command has concluded.

5.2.1.2 Timestamps:

The second problem is that timestamps will not be consistent between the substation and the central station. This is not so much a problem for commands, where it can be argued that the difference is actually useful (the timestamp in the central station denotes when the command was issued and the timestamp in the substation denotes when the command was processed), but it may not be acceptable to have a (possibly relatively big) difference in the timestamps for the feedback information. It depends very much on the application if the difference in the timestamps between the logs in the substation and the central station is a problem or not. It also depends on the customer, if he requires accurate logs in the central station or if he is content to look at the logs in the substation when accurate timestamps are required.

Below is a list of the steps involved in transferring a signal change from the source to the central station. Most of these steps represent a point at which a timestamp can be generated and possibly sent for use in later steps.

Also given is the typical delay between each step and the previous one and on what this delay depends. In some cases the delay can be much higher under adverse conditions (very high load, network problems that require retransmissions ...)

1) I/O Module (SOE).

2) Control Module in the Slave. (I/O delay plus Task cycle, 10 ms to 1 sec)

3) Encode/Write block in IEC program of the slave. (Task Cycle, 0 ms to 1 sec)

4) Protocol Handler in the slave (sends data from PMxxx controller to the CI864 module). (Task execution time plus Load of controller, 0 to 10 ms)

5) Firmware in CI864 in the slave (sends the data). (Load of the CI board, 10 ms to 100 ms)

6) Firmware in CI864 in the master (receives the data). (Transmission time on the wire plus Load of the CI board, 10 ms to 100 ms)

7) Protocol Handler in the master (processes the data from the CI864 and makes it available for the application in the PMxxx controller) (Load of controller, 0 to 10 ms)

8) Receive/Decode block in the IEC program of the master. (Task Cycle, 0 ms to 1 sec)

9) Control Module in the Master. (Task Cycle, 0 ms to 1 sec)

Using the standard libraries, the log in the slave would use the timestamp generated in step 2 (step 1 if SOE input modules are used) and the log in the master would use the timestamp generated in step 9. As one can see, the timestamp in the master can be off by a significant amount.

It is currently (SV 5.0 SP2) not possible to access the timestamp from a SOE input module in the application; therefore it cannot be used in any of the later steps or sent via the IEC 60870-5-104 protocol.

Here are some ideas on how to improve the accuracy of the timestamp in the master, working backwards from step 9 (it makes no sense to generate and send an accurate timestamp if it is not used in a later step).

Step 9: The Control Module or Function Block that logs the event in the master has to be able to use an external timestamp (has an input for it). UniSimpleM and UniM do not have such an input, so either an extended variant of these Control Modules has to be used or the event has to be logged using a different method (such as using a SimpleEventDetector).

If a timestamp generated in an earlier step is used, the task cycle of the modules can be increased without sacrificing accuracy, reducing CPU usage in the master and possibly allowing more data items to be used per controller.

It is possible to generate timestamps in either step 6 or step 8, but this is somewhat unusual; if an accurate timestamp is wanted, it is typically generated in the source station. This would only be used if the source station sends data types that do not include a timestamp.

Without further logic to generate a timestamp manually, the provided libraries will generate the timestamp in step 4. This is always the case if the "Basic" Encode function blocks are used; the "full" Encode function blocks allow a timestamp that was generated earlier to be used.

The Encode function blocks in the provided libraries are not protected, so it is possible to add logic to the blocks to monitor the data items and generate a timestamp when it changes (step 3).

The best solution is to add the code that generates the timestamp to step 2 and put it into the control module. This allows the timestamp to be consistent between the slave and the master station (unless SOE inputs are used). With properly optimized control modules it should be possible to execute the control modules at a fast rate (10 to 100 ms) without overloading the CPU of the slave station.

Alternatively the timestamp could be generated by seperate logic that runs in a very fast task.

Putting the logic that generates/uses the timestamp into the control modules has the advantage that it simplifies engineering. This is highly recommended if this functionality is used extensively (for more than a few signals).

If timestamps are generated by logic, then care has to be taken that they are generated correctly; that they are generated whenever the signal changes. This includes changes in the status as well as changes of the value. Note that the Encode function blocks for analog values ignore the timestamp if Hysteresis processing is used.

5.2.1.3 Optimization and Customization:

The provided libraries are designed to be generic so they can be used with different libraries or application design. On the other hand, this means that they are not optimized for any particular library. This is acceptable if the IEC communication makes up only a small part of the whole project, but it will lead to reduced functionality, higher resource usage (CPU load) and increased engineering cost.

The provided libraries (e.g. IEC60870SlaveLib) are not protected, so it is possible to modify the function blocks in them. It is recommended not to change the libraries directly, but to copy any function blocks that need to be modified to a seperate library with a different name.

Tailoring the IEC function to fit the particular requirements of a particular library or project will of course cause up-front costs, but it can decrease total cost and make the application better readable, simpler and faster. This will usually only make sense if the library the IEC functions are used together with is also modified; see the discussion about timestamp accuracy above.

If the decision is taken to include functionality for the IEC connection in the standard library (for example to generate timestamps), it is a good idea to make structured types that include the timestamp together with the I/O signal and to customize the Encode / Decode functions to handle this as this simplifies engineering and reduces CPU load.

6 Appendix A - Diagnostics

6.1 LED Indicators

There are six LED indicators on the front of the CI864 Expansion Module.

- **FAIL** A red LED that indicates an error of some kind.
- **RUN** A green LED that indicates that the board is running.
- **Tx1/Rx1** Two yellow LEDs that indicate traffic on the first network port.
- **Tx2/Rx2** Two yellow LEDs that indicate traffic on the second network port. (Not used).

The FAIL and RUN LED are used together to impart information about the state of the CI module. When the CI board starts up, it goes through several stages that are índicated by different combinations of the two LEDs.

- **Selftest** FAIL is on, RUN is off.
  After Power-up the CI864 module performs a selftest. This takes approximately 40 seconds. When the selftest is finished the FAIL LED is switched off and the RUN LED blinks once or twice (if the firmware is MAC address locked).
- **Wait for Network Configuration** FAIL is off, RUN is off.
  The CI864 is initialized and waits for the hardware configuration from the AC800M controller. When it receives a valid configuration, it continues with the next stage.
  MAC address lock: If the firmware is MAC address locked and loaded on a wrong CI module, both FAIL and RUN are on. The CI864 will not enter **Run** stage.
  Configuration Error: Some configuration errors will prevent the CI module from entering the **Run** state; others will be reported after the CI board enters the **Run** stage.
- **Run** FAIL is off, RUN is on.
  Configuration Error: Most of the hardware configuration is done at this stage. If there is an error, both FAIL and RUN are on. Depending on the nature of the configuration error, either all or only the affected IEC partner connections will not work until the error has been corrected. (The Connect function block(s) will fail).
  Firmware Expired: If the firmware is time-restricted and the time limit has expired, then FAIL and RUN are on. The CI board will not process most messages.

7 Appendix B - Interoperability List

See [CI864_iec60870-5-104-Clause9.doc].

8 Appendix C - Conversion List

To minimize the amount of protocol-specific information that has to be hard-coded into the CI firmware and to make it possible to quickly add new data types, information about the various ASDU types is configured in the conversion list. ASDU types that are not in the conversion list cannot be handled by the CI864. New ASDU types can be added through the conversion list as long as they obey the same syntactic rules as the existing types, have a fixed size and contain no more than 7 data fields of up to 4 bytes each. An ASDU type may contain additional constant data fields.

Detailed knowledge of the protocol and the CI firmware is required to add new ASDU types.

See [Conversion List.doc] for more information.

9 Appendix D - Implementation Limits

One CI864 module can handle up to 8 IEC partners or 8 pairs of redundant partners. The CPU of the currently used hardware limits this to 8 partners total, e.g. 8 single partners, 4 pairs of redundant partners or any combination of the two.

Each partner can have up to 2000 data items. CPU power and available memory will limit the total number of data items

One CI864 module can handle up to 50 telegrams with a total of 300 data items per second. It can handle short bursts in excess of that by buffering data items until they can be sent to the IEC partner.