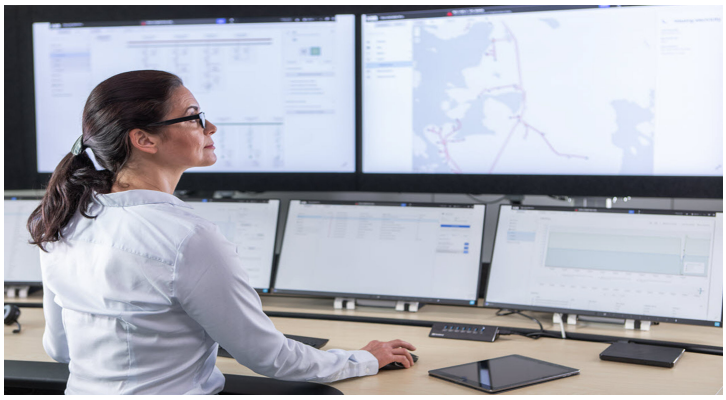
GRID AUTOMATION PRODUCTS

# MicroSCADA X SYS600 10.2

## Communication Programming Interface (CPI)



Document ID: 1MRK 511 469-UEN
Issued: March 2021
Revision: A
Product version: 10.2

# Table of contents

# Section 1 Copyrights

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

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

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

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



**Trademarks**

ABB is a registered trademark of ABB Asea Brown Boveri Ltd. Manufactured by/for a Hitachi Power Grids company. All other brand or product names mentioned in this document may be trademarks or registered trademarks of their respective holders.

**Guarantee**

Please inquire about the terms of guarantee from your nearest Hitachi Power Grids representative.

**Third Party Copyright Notices**

List of Third Party Copyright notices are documented in "3rd party licenses.txt" and other locations mentioned in the file in SYS600 and DMS600 installation packages.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (https://www.openssl.org/). This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

# Section 2 Introduction

## 2.1 This manual

This manual provides thorough information on the Communication Programming Interface (CPI) and needed information related to it. CPI is available for connecting foreign systems to SYS600.

### 2.1.1 CPI interface

Communication Programming Interface CPI is a protocol environment software that is used for externally implemented protocol converters.

CPI is a collection of functions that enables the data transfer between the SYS600 base system and the CPI application program. The CPI library contains functions for sending and receiving messages to or from the SYS600 base system, as well as functions for packing and unpacking data of messages. The CPI application program may be run on different computers as can be seen in Figure 1 or also on the same computer with SYS600.

The CPI application program (gateway program), which uses the CPI interface, emulates the NET containing stations of the RTU, SPA or SPI (RP 570 slave) device type. The SPI device type can also be emulated as a RTU device type. In this case, the SPI station should be configured as a RTU station by the SYS600 base system. CPI supports both the slave and the master communication.

Normally, only those messages sent from the base system to the station devices (RTU, SPA), which have been configured in the CPI application node, are visible for the CPI functions. To be able to emulate a conventional NET unit operation, there is also a need to access messages directed to the CPI application node itself (NET object). To access these messages, filtering must be disabled using the message filtering functions (see Section 5).

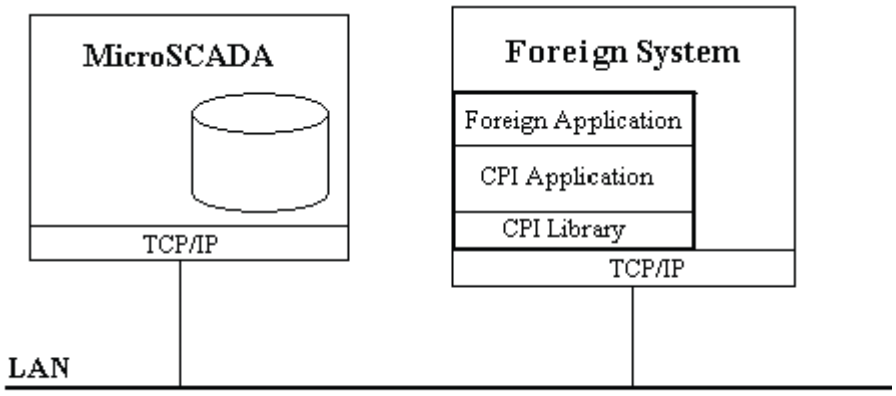



*Figure 1: SYS600 connected to a foreign system that has a CPI application program as a part of it. The data transmission between the CPI application program and the SYS600 base system is implemented through the TCP/IP interface.*

### 2.1.2 TCP/IP interface

The communication between applications using CPI and SYS600 is based on the TCP/IP protocol. Transmission Control Protocol/Internet Protocol is used for data transmission in LAN network to provide communication with several protocols across the connected networks of computers (see Figure 1). On the TCP/IP interfaces, the ACP messages are sent via stream type BSD sockets.

The CPI interface is designed to support connections to several applications running one or more SYS600 base systems. All applications on one certain SYS600 base system can be reached by using the same TCP/IP socket.

## 2.2 Use of symbols

This publication includes warning, caution and information symbols where appropriate to point out safety-related or other important information. It also includes tips to point out useful hints to the reader. The corresponding symbols should be interpreted as follows:

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

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

Information icon alerts the reader to relevant factors and conditions.

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

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

## 2.3 Related documents

The following SYS600 manuals should be available for reference during the use of this manual:

| Name of the manual | Document ID |
|---|---|
| SYS600 10.2 System Objects | 1MRK 511 482-UEN |
| SYS600 10.2 Programming Language SCIL | 1MRK 511 479-UEN |
| SYS600 10.2 System Configuration | 1MRK 511 481-UEN |
| SYS600 10.2 Application Objects | 1MRK 511 467-UEN |
| SYS600 10.2 Status Codes | 1MRK 511 480-UEN |

## 2.4 Document conventions

The following conventions are used for the presentation of material:

- The words in names of screen elements (for example, the title in the title bar of a dialog, the label for a field of a dialog box) are initially capitalized.
- Capital letters are used for file names.
- Capital letters are used for the name of a keyboard key if it is labeled on the keyboard. For example, press the CTRL key. Although the Enter and Shift keys are not labeled they are written in capital letters, e.g. press ENTER.
- Lowercase letters are used for the name of a keyboard key that is not labeled on the keyboard. For example, the space bar, comma key and so on.
- Press CTRL+C indicates that the user must hold down the CTRL key while pressing the C key (in this case, to copy a selected object).
- Press ALT E C indicates that the user presses and releases each key in sequence (in this case, to copy a selected object).
- The names of push and toggle buttons are boldfaced. For example, click **OK**.
- The names of menus and menu items are boldfaced. For example, the **File** menu.
    - The following convention is used for menu operations: **Menu Name/Menu Item/ Cascaded Menu Item**. For example: select **File/Open/New Project**.
    - The **Start** menu name always refers to the **Start** menu on the Windows Task Bar.
- System prompts/messages and user responses/input are shown in the Courier font. For example, if the user enters a value that is out of range, the following message is displayed: `Entered value is not valid`.
  The user may be told to enter the string MIF349 in a field. The string is shown as follows in the procedure: `MIF349`
- Variables are shown using lowercase letters: sequence name

## 2.5 Document revisions

| Revision | Version number | Date | History |
|---|---|---|---|
| A | 10.2 | 31.03.2021 | New document for SYS600 10.2 |

# Section 3 CPI overview

## 3.1 NET unit emulation with CPI

CPI communication resembles the communication between the base system and the NET unit. The CPI application program acts as a communication front-end. This is called the NET unit emulation. Although CPI is an external and independent application, the base system does not see any difference between the messages coming from the NET unit or from the CPI application because of the similar interfaces in both cases.

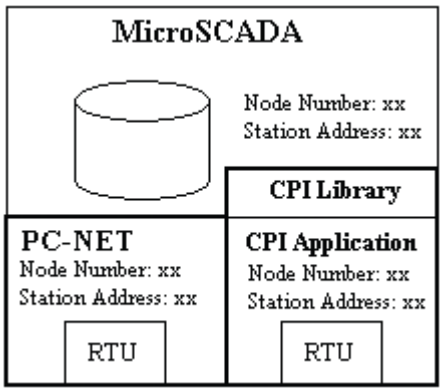



*Figure 2: CPI communication emulates the communication between the SYS600 base system and the NET unit*

## 3.2 Operating principle

The CPI application program usually runs in an infinite loop where it checks with a cpiSelect function if there are any incoming messages from SYS600. The following examples show the basic functionality of the CPI interface. The detailed description of every function presented below can be found in Section 5.

```
/* Example of receiving the commands from the base system*/
Gateway()
{
  cpiMsgID recv_id,reply_id=0;
  cpiBitSet in_set,out_set,exp_set;
  int msg_type,reply_pending;
  attr char[2];

GV_Initialize_Connection()
/* Infinite loop*/
for(;;)
{
 in_set = out_set = exp_set = 0x7;
 cpiSelect( &in_set, &out_set, &exp_set,15);
 if (cpiISSET(1, in_set)) {
    cpiGetMessage(1, &recv_id, &msg_type);
    switch( msg_type ) {
    case cpiWRMSG:

/* processing of the write message */
cpiGetMsgAttr(recv_id, &attr);
if ( GV_ Is_Config_Attr( &attr )) {
   cpiGetReplyBuffer(recv_id,&reply_id);
```

```
   if (reply_id != 0) {
     GV_Process_Write_To_Attribute (recv_id,reply_id,,,);
     if (cpiISSET(1, out_set)) {
       cpiSendMessage(1,reply_id,5);
       reply_pending = 0;
      }
      else {
       reply_pending = 1;
      }
    }
   }
   break;
 case cpiRDMSG:

   /* processing for the read message */
 case cpiRPMSG:

   /* processing for the reply message */
  }
 }
 if (reply_pending) && cpiISSET(1, out_set)) {
     cpiSendMessage(1,reply_id,5);
     reply_pending = 0;
    }
   }
  }

/* Example 2, how to send data to the base system */
Gateway()
{
  cpiMsgID ind_id, fail_id;
  cpiBitSet in_set,out_set,exp_set;
  int reply_ready,retries,status;

GV_Initialize_Connection()
  reply_ready = 1;
  /*Infinite loop*/
  for(;;)
  {
    in_set = out_set = exp_set = 0x1;

    cpiSelect( &in_set, &out_set, &exp_set);

    /* Sending of an indication value */
    if (GV_Indication_To_Send()) && (cpiISSET(1,out_set)) {
       if (reply_ready) {
        GV_Process_Indication_Data(,,&Block,&Bit,&Val);
        cpiSendRtuIndication(ind_id,1,1,Block,Bit,Val,5);
        cpiCLR(1,&out_set);
        retries = reply_ready = 0;
                }
            }

/* Checking the reply message */
  if (cpiISSET(1, in_set)) {
  cpiGetMessage(1, &recv_id, &msg_type);
  switch( msg_type ) {
   case cpiRPMSG:
      cpiGetReplyStatus(recv_id, &repl_id, &status);
         if ((status != OK_STATUS) || (repl_id != ind_id)) {
          display_error_msg("Error reply for indication ", Block, Bit,
Val);
        }
    reply_ready = 1;
   }
 }
```

```
/* Testing the possible exceptions*/
   if (cpiISSET(1, exp_set)) {
     switch( cpiException(1) {
      case TIMEOUT:
        cpiGetFailedMessage(1, &fail_id);
         if (++retries > 3) {
            cpiGiveUp(fail_id);
             display_error_msg("Sending of indication fails", Block, Bit,
Val);
             }
          else {
         cpiRetransmit(fail_id);
        }
       break;
      case TCPIP_ERROR:
     /* no connection to the base system*/
    }
 }
```

## 3.3 Memory management

An application automatically reserves the memory needed when the cpiInitRemoteNode routine is called. CPI reserves memory pools also for message buffers:

- Received messages
- Transmitted messages
- Transmissions failed

These pools are CPI internal and the CPI application program cannot see them directly. It must have its own memory management for communicating with the protocol it emulates.

## 3.4 Application Communication Protocol (ACP)

Communication between the SYS600 base system and front-ends like the NET unit is based on ACP (Application Communication Protocol). It is an internal protocol of SYS600 that is generally used in communication between the SYS600 nodes. The CPI application program can also be seen as one of the nodes.

The NET unit can communicate simultaneously with several applications on the base system. The CPI library using the ACP communication supports the following types of message dialogues presented in Figure 3:

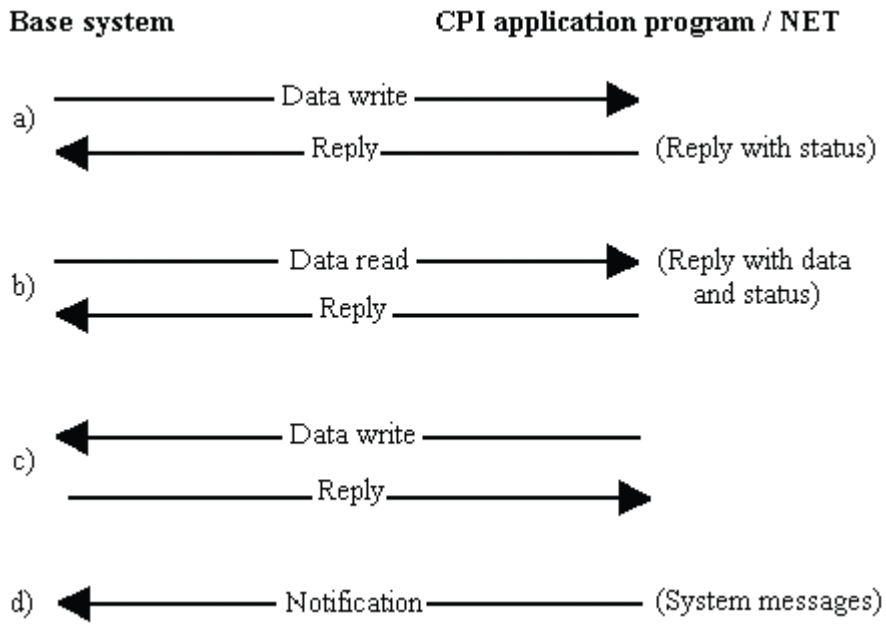



*Figure 3: Message dialogues supported in the ACP communication*

**a)** Write commands: The base system writes the value of an attribute to the CPI application program.

**b)** Read commands: The base system requests the value of an attribute from the CPI application program.

**c)** Write commands: The CPI application program writes data to the base system process database.

**d)** Notifications: The system message is sent from the CPI application program to the base system.

## 3.4.1 ACP messages

Each communication system contains a set of system objects, which specify the line properties, connected devices etc. These objects can be created, modified and deleted by SCIL (Supervisory Control Implementation Language), and setting the attributes of the objects can change the properties. The detailed description of attributes can be found in Section 6 and the creation of protocol converters in Section 4.

The target of the ACP message sent by the SYS600 base system is an attribute of a communication system object (NET, STA), which has been configured in the NET unit. The attributes of the communication system object can be divided into two main categories:

- Configuration attributes
- Communication attributes

Configuration attributes refer to the internal data of the NET unit. The NET unit can reply to these messages immediately. Communication attributes refer to the external data of the device. In this case, the ACP message must first be converted to the protocol of the external device before sending it forward. The replies to the communication attributes are usually not sent back to the base system before the actual reply from the external device is received.

Data transmission from the base system to the CPI application program is done by setting the attributes of the communication system object or the process object. This may happen by using SCIL and its commands.

ACP implements the layers presented in Table 1

*Table 1: ACP-OSI layer mapping*

| OSI | ACP Feature | | | | Remark | ACP layers |
|---|---|---|---|---|---|---|
| **Application** | Varying | | | | Not explicitly specified | ACP application layer |
| | Proc. database and SCIL interface | | | | Device profiles | |
| **Presentation** | ACP Data Array formats | | | | Data coding rules | |
| **Session** | Attribute read/write | | | | Connectionless session model comparable with remote proc. calls | |
| **Transport** | ACP Transport | | | | ACP end-to-end acknowledge flow control by delayed acknowledge | |
| **Network** | Device level network info | | | | Device level addressing | ACP network layer |
| | ANSI X3.28 | | | | Node level addressing | |
| **Link** | TCP/IP | ANSI X3.28 | Common RAM | Integrated LINK | According to system components and topology | ACP link layer |
| **Physical** | LAN HW | RS-232-C | Computer HW | Computer HW | According to link | ACP physical layer |

The ACP network layer consists of two sublayers:

- ANSI X 3.28 Network layer handles low level (link layer) message routing (node-to-node).
- ACP Network layer information is used for routing on the device level (application layer).

The following shows a list of fields of the ACP protocol messages. The CPI functions that can be used to get referred information are presented as well.

The first list shown in Table 2 presents the fields, which are used for the low level message routing. The second list in Table 3 consists of fields used for routing on the device level.

*Table 2: Fields used for the low level message routing*

| Field | Description | CPI function |
|---|---|---|
| DST | Destination node address | cpiGetDestination |
| SRC | Source node address | cpiGetSource |
| CMD | Command/reply code | cpiGetMessageType (not read directly) |
| STS | Status | cpiGetReplyStatus (not read directly) |
| TNS | Transaction number | |

*Table 3: Fields used for message routing on the device level*

| Field | Description | CPI function |
|---|---|---|
| LDS | Logical destination | cpiGetRtuNo (not read directly), cpiGetDeviceType, cpiGetDeviceNumber |
| LSR | Logical source | Connection number |
| CNT | Transaction count | |

Table continues on next page

| Field | Description | CPI function |
|---|---|---|
| FNC | Function | cpiGetMessageType<br>(not read directly) |
| MBX | Mailbox | |
| ATR | Attribute | cpiGetAttribute |
| IND1 | First index | cpiGetIndex |
| IND2 | Last index | cpiGetIndex |
| DATA | Device data | cpiGetNextData, cpiGetNextDataType,<br>cpiGetNextAddrData |

The following examples describe the relationship between SCIL and fields of the ACP message:

**Writing data to the CPI application:**

```
#SET STA1:SDT(1..2)=(1,2)
```

| | |
|---|---|
| SET | Message type (command + FNC write) |
| STA1 | RTU number (device number and type packed to LDS) |
| SDT | Attribute |
| (1... | First index |
| ...2) | Last index |
| (1,2) | Data |

**Reading data from the application:**

```
@A = STA1:SAM(10)
```

| | |
|---|---|
| @A | Message type (command + read) |
| STA1 | RTU number (device number and type packed to LDS) |
| SAM | Attribute |
| (10) | First index |

## 3.4.2 Reserved attributes

Because the CPI application program emulates STA objects in the NET unit, the use of attributes is not completely free. Some attribute names are reserved for the RTU process data communication. They cannot be used as the attributes of an application unless they have been used exactly in the same way as in the RTU or SPA devices.

The following attributes are used by the system and therefore they are not allowed to be used as the attributes of the application program. Except for the attributes below, attributes can be formed by using any abbreviation that consist of two letters.

*Table 4: The attributes not allowed to be used as the attributes of the application program*

| Attribute | Description |
|---|---|
| DA | Process data |
| SM | System status message |

The messages to the DA attribute are generated automatically by the base system when the output process objects are set. If this method is used with the CPI application program, it must emulate the DA attribute handling exactly in the same way as the RTU device does.

The attributes IU, SA, AL, AS, MI and MS are used with the RTU type station. It is recommended that these attributes should be used in the same way as documented in SYS600 System Objects manual.

## 3.5 Transmission error handling

When using the CPI programming interface, there are possibilities to handle different kinds of communication errors between the CPI application program and the base system. The following is a list of transmission error types on the CPI application program level. Examples of how the application reacts to each situation are presented as well:

- The TCP/IP connection is not open when the CPI application program tries to send a message to the base system (or for some other reason the message is not actually sent): An error status is returned by one of the ACP Basic communication functions, for example cpiSendMessage or cpiSendRtuMessage.
- SYS600 base system does not reply to the ACP data message: The failed application connection is identified by the exception flags of the cpiSelect function. The function cpiGetTransmissionBuffer cannot get a new buffer if failed transmission exists. It is therefore possible to retransmit or delete the message that has failed.
- SYS600 base system replies with an error status to the data message: The application program can read the status of the reply message by using the cpiGetReplyStatus function. In this case, there is no automatic retransmission mechanism included to the CPI retransmission mechanism. If the base system replies with the error status message, it usually will not help to transmit the same message again for it is very possible that it replies with the error status again.

## 3.6 Base system configuration

Each base system has a set of objects that specify the base system and its environment, hardware and software, as well as the physical and logical connections of the base system and its applications.

Base system objects are defined with SCIL commands in the SYS_BASCON.COM file, which is executed every time the base system is started. With a few limitations, the user can also define and modify the base system objects any time when SYS600 is running. During the operation, the base system objects are in the primary memory of the base system computer.

The CPI application program emulates the RTU or SPA type of stations, which means that the interface towards the base system is similar to that of the RTU or SPA stations. The following describes the configuration of the base system for the CPI application program which emulates the RTU type of station. The CPI application program requires a node of its own. Communication between this node and the base system takes place via a LAN link.

### 3.6.1 Configuration steps

To configure SYS_BASCON.COM:

1. Define the base system.
2. Define a LAN link.
3. Define an application.
4. Define nodes.
5. Define the RTU stations. The number of RTU stations can be the same as the number of the connected process units. This way, the information related to a certain process unit can be routed to the corresponding station in the NET unit. This way, the information can be differentiated between different process units. In the COM 500i configuration, it is possible to use a maximum of 8 NCC stations

The definitions are made in the example below. For more information on the system objects, see SYS600 System Objects. Steps 1..3 are necessary, but they are configured using the standard SYS_BASCON.COM template. Steps 4..5 are necessary for the CPI application.

## 3.6.2 Example

The following is an example of the SYS_BASCON.COM file for communication with the Modbus slave protocol. An application CPI_TEST together with two stations is defined in the SYS600 base system.

```
;*********************************************************
;
; SYS_BASCON.COM
; BASE SYSTEM CONFIGURATION TEMPLATE
;
;*********************************************************

#CREATE SYS:B = LIST(-
          SA = 209,- ;STATION ADDRESS OF BASE SYSTEM
          ND = 9,- ;NODE NUMBER OF BASE SYSTEM
          DN = 3,- ;DEFAULT NET NODE NUMBER
          DS = "RTU",- ;STA TYPES: E.G. STA,RTU,SPA,REX
          FS = "NEVER") ;FILE SYNCH CRITERIA

;*********************************************************
;
;  COMMUNICATION LINKS

  #CREATE LIN:V = LIST(- ;REQUIRES TCP/IP
                        LT = "LAN") ;FOR SYS-SYS AND LAN FRONTEND
 #CREATE LIN1:B = %LIN

;*********************************************************
;
;  COMMUNICATION NODES

#CREATE NOD:V = LIST(-
             LI = 1,-
             SA = 203)
#CREATE NOD3:B = %NOD

;*********************************************************
;
;  PRINTERS

;*********************************************************
;
;  MONITORS

#LOOP_WITH I = 1..5
      #CREATE MON'I':B = LIST(-
                TT = "LOCAL",- ;TRANSLATION TYPE
                DT = "X") ;X MONITOR
      @MON_MAP(%I) = -1
#LOOP_END

#LOOP_WITH I = 6..10
      #CREATE MON'I':B = LIST(-
                TT = "LOCAL",- ;TRANSLATION TYPE
                DT = "VS") ;VISUAL SCIL MONITOR
      @MON_MAP(%I) = -1
#LOOP_END

;*********************************************************
;
; APPLICATIONS
```

```
#CREATE APL:V = LIST(-
        TT = "LOCAL",- ;TRANSLATION TYPE
        NA = "CPI_TEST",- ;NAME OF APPLICATION DIRECTORY
        AS = "HOT",- ;APPLICATION STATE: COLD,WARM,HOT
        HB = 2000,- ;HISTORY BUFFER SIZE)
        RC = VECTOR("FILE_FUNCTIONS_CREATE_DIRECTORIES"),-
        AP = (1,2),-
        MO = %MON_MAP,- ;MONITOR MAPPING
        PR = (1,2,3)) ;PRINTER MAPPING
#CREATE APL1:B = %APL

;*********************************************************
; STATIONS

#CREATE STA1:B = LIST(-
             TT = "EXTERNAL",-
             ST = "RTU",-
             ND = 3,-
             TN = 1)

#CREATE STA2:B = LIST(-
             TT = "EXTERNAL",-
             ST = "RTU",-
             ND = 3,-
             TN = 2)
;*********************************************************
```

# Section 4 Creating protocol converters with CPI

When a protocol converter is implemented using the CPI library, the project normally consists of the following parts:

- The application programming required in SYS600.
- The program that utilizes the CPI library for data transfer with the SYS600 base system.
- The program that converts the messages between the foreign protocol and the CPI application program.
- The program that handles the foreign protocol functions.
- An interface between the CPI part and the foreign protocol part of the program.

Furthermore, the protocol converters done with CPI can usually be divided into two categories. In the first case, the SYS600 base system is seen as a master and the slave devices connected via CPI collect data into the SYS600 process database. In the second case, the SYS600 base system is seen as a slave device and data is sent from the SYS600 process database to the master via CPI. The use of CPI and the work needed in different parts of the project differs significantly in these two cases. The following sections explain the differences and give information about the possible solutions of implementing protocol converters with CPI.

## 4.1 Slave devices connected to SYS600 via CPI

When slave devices are connected to the SYS600 base system via the CPI, the data flow is typically from the slave devices into SYS600 as shown in Figure 4. Only commands (e.g. breaker operation) are transferred from the SYS600 base system to the slave devices. From the SYS600 base system's point of view, the slave devices are seen either as RTU or SPA type stations.

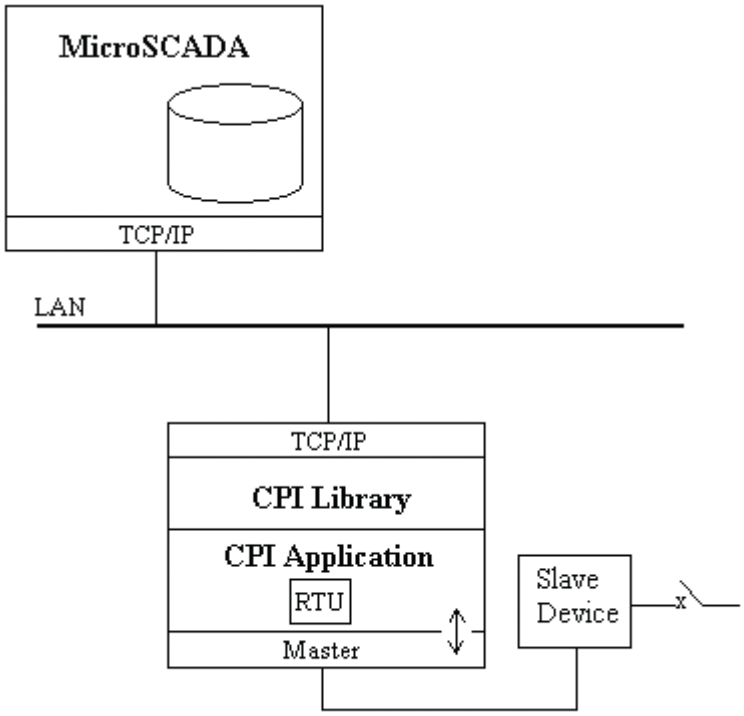



*Figure 4: The CPI application program acts as a master collecting data from the slave device to the SYS600 process database. The slave device is seen as a RTU station by the SYS600 base system.*

### 4.1.1 SYS600 base system configuration

The CPI application is configured into the SYS600 base system as a node. This node is placed on a TCP/IP LAN link. The devices connected to the CPI are configured as RTU or SPA stations. See Section 3.6 for more information about base system configuration.

### 4.1.2 Application programming in SYS600

The need for special application programming in this case is usually very small. The library can be used to create a process database and picture functions.

### 4.1.3 Data transfer with the SYS600 base system using the CPI library

The process data received from the slave devices can be sent to the SYS600 process database using the following CPI functions (detailed descriptions of each function can be found in Section 5). Notation “XXX” means that there are several functions with variable timestamps.

| | |
|---|---|
| Indications: | cpiSendRtuIndicationXXX |
| Analog values: | cpiSendRtuAnalogValueXXX |
| Pulse counters: | cpiSendRtuPulseCounterXXX |
| Digital values: | cpiSendRtuDigitalValue |
| BitStream values: | cpiSendRtuBitStream |

The commands from the SYS600 base system can be handled using the following cpi function:

cpiGetNextAddrData

With this function, it is possible to read the command object type, address and the value. The object type of the command can be one of the following:

- Object command (binary output).
- Regulation command.
- Digital setpoint.
- Analog setpoint.
- General persistent output.

### 4.1.4 Converting messages between CPI and the foreign protocol

It is difficult to give any specific instructions for converting messages between CPI and the foreign protocol. However, the following questions need to be answered:

- How does the addressing of the data differ from the SYS600 base system addressing method? Is a cross-reference table needed or could it be possible to map the addresses directly into the SYS600 base system address space?
- How closely do the data types of the foreign protocol match the available types in the SYS600 base system? What kind of a conversion or scaling is needed?
- Is there a need for intermediate database in the CPI to reduce the data transfer between the CPI application program and the SYS600 process database?

### 4.1.5 Implementation of the foreign protocol

The foreign protocol is normally implemented as a separate thread (process), which simplifies the program structure and makes it easier to handle time critical tasks both in the CPI part and the foreign protocol part of the program.

### 4.1.6 Interface between the CPI part and the foreign protocol part of the program

There are several ways of handling the communication between the CPI part and the foreign protocol part of the program. The example program delivered with the CPI installation package named CPI_EXAMPLE uses common global variables for transferring data between the program parts (threads).

## 4.2 Master device connected to SYS600 via CPI

When a master device is connected to the SYS600 base system via the CPI, the data flow is normally from the base system to the master as shown in Figure 5. Only commands (e.g. breaker operation) are transferred from the master into the SYS600 base system. It is suggested that the CPI application program emulates the RP 570 slave device (SPI) as closely as possible in order to be able to use the existing applications in the SYS600 base system. These applications are used for data transfer between the SYS600 base system and the CPI application program (COM 500*i*).

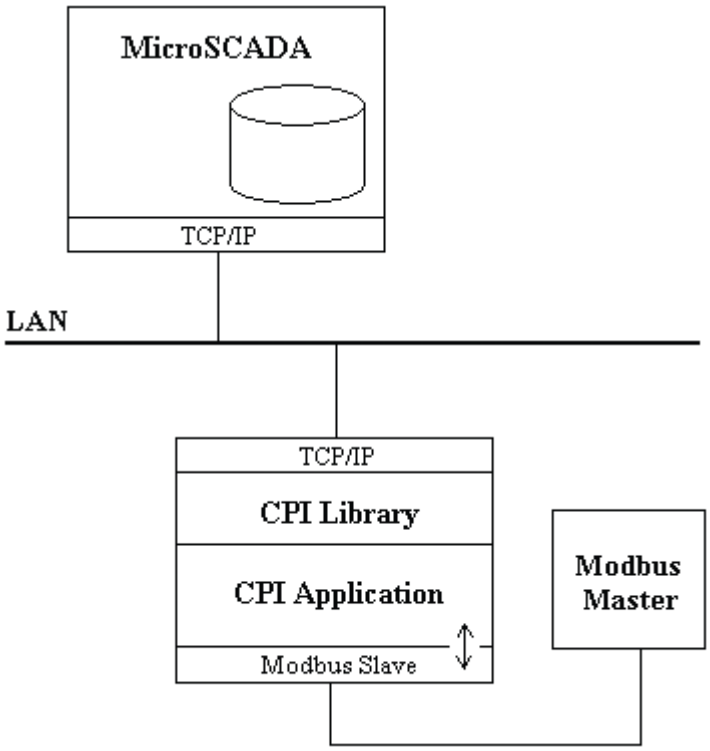



*Figure 5: SYS600 connected with the CPI application (e.g. Modbus Slave). The CPI application acts as a slave providing data from the SYS600 database for the master.*

### 4.2.1 SYS600 base system configuration

The CPI application is configured into the SYS600 base system as a node. This node is placed on a TCP/IP LAN link. The master device connected to the CPI is configured as a RTU or SPA station although its behaviour resembles more that of the SPI station. See Section 3.6 for more information about the base system configuration.

### 4.2.2 Application programming in SYS600

The application program in SYS600 must take care of forwarding the data from the process database to the CPI application program. This is usually done by using event channels in the process objects that are connected to command procedures. The commands from the master are received into the input process objects activating a command procedure that will then operate the actual output process object. When the interface to the CPI is done according the SPI (RP 570 Slave device) profile, it is possible to use the existing command procedures and tools (COM 500*i*). See SYS600 System Objects for details of the SPI station attribute interface.

### 4.2.3 Data transfer with SYS600 using the CPI library

The data received from SYS600 can be unpacked with the following CPI functions (detailed descriptions of each function can be found in Section 5):

cpiGetRtuNo

cpiGetAttribute

cpiGetIndex

cpiGetNextDataType

cpiGetNextData

cpiGetNextS32Data

The commands received from the master can be sent to the SYS600 base system by using the following CPI function:

cpiSendRtuDigitalValue

### 4.2.4 Converting messages between the CPI and the foreign protocol

It is difficult to give any specific instructions for converting messages between the CPI application program and the foreign protocol. Normally the following questions need to be answered:

- How does the addressing of the data differ from the SYS600 base system addressing method? Do we need a cross-reference table or can we map the addresses directly into the SYS600 base system address space?
- How closely do the data types of the foreign protocol match the available types in the SYS600 base system? What kind of a conversion or scaling is needed?
- Is there a need for intermediate database in the CPI to reduce the data transfer between the CPI application and the SYS600 process database?

### 4.2.5 Implementation of the foreign protocol

The foreign protocol is usually implemented as a separate thread (process), which simplifies the program structure and makes it easier to handle time critical tasks both in the CPI part and the foreign protocol part of the program.

### 4.2.6 Interface between the CPI part and the foreign protocol part of the program

There are several ways of handling the communication between the CPI part and the foreign protocol part of the program. The example program delivered with the CPI installation package named CPI_EXAMPLE uses common global variables for transferring data between the program parts (threads).

# Section 5 CPI library

## 5.1 General

CPI library is a collection of functions for sending and receiving messages between SYS600 and the CPI application, and also for message handling, packing and unpacking. The functions contained in the CPI library are specified in this section. A list of the program files is presented in Table 5 below.

*Table 5: Program files for CPI*

| File Name | Description |
|---|---|
| cpint.lib | Microsoft Developer Studio Win32 Static Library file. |
| cpi.h | Header file for the CPI functions. |
| cpicl.h | CPI internal definition, type and function header file. |
| bufms.h | CPI buffer handling definitions. |

Part of the definition file cpi.h, which contains the definition for different data types:

```
typedef int cpiMsgID;

typedef unsigned char cpiObjectType;
/* Object type numbers are same as in the ACP */

     #define cpiU8            0
     #define cpiS8            1
     #define cpiU16           2
     #define cpiS16           3
     #define cpiU32           4
     #define cpiS32           5
     #define cpiF32           6
     #define cpiADDR          7
     #define cpiBV            8
     #define cpiB             9


typedef struct {
      cpiObjectType ObjectType;
      unsigned int ObjectAddress;
      int ObjectValue;
}addressItem;

typedef struct {
   cpiDataType item_type;
   union { /* this definition is cpu specific */
```

```
    unsigned char                          U8;
    unsigned char                          S8;
    unsigned short int                     U16;
    short int                              S16;
    unsigned long int                      U32;
    unsigned int                           S32;
    float                                  F32;
    unsigned short                         A;
    addressItem                            ADDR;
  } value;
 }cupidity;

 /* Message types */
 #define chaperons  1
 #define cpiMsgID   2
 #define cpiMsgID   3
 #define cabinets   4


typedef struct {
  unsigned char (year,month,day,hour,minute,second,ms)
  } cpiTime;
```

## 5.2 Basic ACP Communications

The communication between the SYS600 base system and the NET unit is based on the ACP, internal communication protocol of SYS600.

*Table 6: Basic ACP Communications*

| Function | Description |
|---|---|
| cpiSelect | Checks the status of the TCP/IP socket. |
| cpiException | Determines the cause of exception. |
| cpiSendMessage | Sends an ACP formatted message to SYS600. |
| cpiSendRtuMessage | Sends an ACP formatted message to SYS600 from RTU. |
| cpiSendSystemMessage | Sends a system message from any device. |
| cpiSendStandardSystemMessage | Sends a system message which activates RUNNING/SUSPENDED events for APL_EVENT event channel. |
| cpiSendStandardSystemMessageWithBinary | Sends a system message which activates RUNNING/SUSPENDED events for APL_EVENT event channel and updates process objects used by the System Self Supervision. |
| cpiSendNodeSystemMessage | Sends a system message for the CPI node itself, analog status point updated. |
| cpiSendNodeSystemMessage WithBinary | Sends a system message for the CPI node itself, analog and binary status points are updated. Binary status point is used by the System Self Supervision. |
| cpiSendStandardUserActivity Message | Sends a user activity event system message from the application using CPI to any device. |
| cpiGetMessage | Reads an ACP formatted message from the received message queue. |
| cpiClearExceptionStatus | Clearing the status when an exception has occurred. |
| cpiSetApplicationCold | Set the "data not accepted flag" of the application. |
| cpiGetApplicationCold | Returns the "data not accepted flag" of the application. |

## 5.2.1 cpiSelect

This function checks the TCP/IP socket in case there are any messages coming from the base system application, and if there are any connections ready for writing. It also returns information if any of the connections have failed or any message has not got a reply in time. Some of the communication between the base system and application is actually done inside this function, so it must be called in the main loop of the CPI application. The cpiSelect function returns when one of the specified input connections has received a message, or after the specified time has elapsed.

When this function is called, its arguments specify which connections the user is interested in. Then the function modifies the arguments to indicate the status of each connection.

*Table 7: The bit position indicates the status of the connection*

| RD | WR | EX | Description |
|---|---|---|---|
| bit0 | bit0 | bit0 | connection 1 flags |
| bit1 | bit1 | bit1 | connection 2 flags |
| bit2 | bit2 | bit2 | connection 3 flags |

| | |
|---|---|
| RD | Bit is set when CPI has received a message from this connection. |
| WR | Bit is set when CPI does not wait any reply from this connection. |
| EX | Bit is set when CPI has lost this connection or data message does not get reply from this connection. The actual cause of exception can be determined by using the cpiException function. |

```
int cpiSelect(cpiBitSet *RD, cpiBitSet *WR, cpiBitSet *EX, long MS);
```

| | |
|---|---|
| RD | Received message flags (max 16 connections) |
| WR | Ready for writing flags |
| EX | Failed connections flags |
| MS | Max. waiting time in milliseconds |

Return values:

| | |
|---|---|
| 0 | : OK |

## 5.2.2 cpiException

This function determines the cause of exception (cpiSelect EX flags).

```
int cpiException(int CONNECTION);
```

Return values:

| | |
|---|---|
| 1 | : TCP/IP error<br>- Connection to this node has been lost |
| 2 | : No reply error<br>- Message sent by CPI has not received a reply from SYS600 |
| 6 | : Application error<br>- Message sent by CPI has received a reply with error status from SYS600 |

## 5.2.3 cpiSendMessage

This function sends an ACP formatted message to SYS600. Before using this function, the message buffer must be reserved by using the functions cpiGetTransmissionBuffer or GetReplyBuffer, and formatted by using some of the cpiPutXXXData functions presented in Section 5.6.2. (Notation XXX means that any function of this type is available.)

```
int cpiSendMessage(int DESTIN, cpiMsgID ID, int TIMEOUT);
```

| | |
|---|---|
| DESTIN | Destination connection number |
| ID | Message buffer identification |
| TIMEOUT | Waiting time for the base system reply in milliseconds,<br>if 0 -> no time-out supervision |

Return values:

| | |
|---|---|
| 0 | : OK |
| -2 | : Invalid MsgID ; the buffer ID is not free, or it is unknown |

## 5.2.4 cpiSendRtuMessage

This function sends an ACP formatted message to SYS600 from the RTU. Before using this function, the message buffer must be reserved by using the function cpiGetTransmissionBuffer and formatted by using some of the cpiPutXXXData functions presented in Section 5.6.2. (Notation XXX means that any function of this type is available.)

```
int cpiSendRtuMessage(int DESTIN, int SOURCE, cpiMsgID ID, int TIMEOUT);
```

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source RTU number |
| ID | Message buffer identification |
| TIMEOUT | Waiting time for the base system reply<br>in milliseconds,<br>if 0 -> no time-out supervision |

Return values:

| | |
|---|---|
| 0 | : OK |
| -2 | : Invalid MsgID |

## 5.2.5 cpiSendSystemMessage

The following function can be used to send a system message from any device. This function is present for backward compatibility and the usage of it is currently discouraged. Function cpiSendStandardSystemMessage is preferred instead.

int cpiSendSystemMessage(int con, int source, int address, int dataValid, short status)

| | |
|---|---|
| source | Source logical device, the following macro<br>LOGICAL_ADDRESS(deviceType,deviceNumber) |
| address | address of the ANSI analog input process object to be updated |
| dataValid | cpiDATA_VALID<br>;The updated data values from the device are valid<br>cpiDATA_NOT_VALID |
| status | status value for the ANSI analog input process object |

Table continues on next page

Example:
;Sending NET1 line status 12602 to address 6103:

```
cpiSendSystemMessage(   con,
                        LOGICAL_ADDRESS(NET_DEVICE,1),
                        6103,
                        0,
                        12602)
```

## 5.2.6 cpiSendStandardSystemMessage

This function can be used to send a system message from any device. State values cpiDEVICE_STATUS_CHANGES_TO_SUSPENDED and cpiDEVICE_STATUS_CHANGES_TO_RUNNING will activate the predefined event channel APL_EVENT. For more information, see SYS600 Application Objects. When cpiSendStandardSystemMessage function is called, an ANSI analog input status point with settings UN=0, OA=address in the receiving application is updated also. The usage of this function is preferred instead of cpiSendSystemMessage.

```
int cpiSendStandardSystemMessage(int con, int source, int address, int newState, short status)
where
```

| | |
|---|---|
| source | source Source logical device, the following macro should be used to calculate the value LOGICAL_ADDRESS(deviceType, deviceNumber) |
| address | address of the ANSI analog input process object to be updated |
| newState | cpiDEVICE_STATUS_NOT_CHANGED<br>;Device status is not changed<br>cpiDEVICE_STATUS_CHANGES_TO_SUSPENDED<br>;Device status changes to suspended<br>cpiDEVICE_STATUS_CHANGES_TO_RUNNING<br>;Device status changes to running |
| status | status value for the ANSI analog input process object |

Example:
; Send a suspension status 13251 from SPA device 2.

```
cpiSendStandardSystemMessage( con,
                              LOGICAL_ADDRESS(SPA_DEVICE, 2)
                              1002,
                              cpiDEVICE_STATUS_CHANGES_TO_SUSPENDED,
                              13251)
```

## 5.2.7 cpiSendStandardSystemMessageWithBinary

This function can be used to send a system message from any device. State values cpiDEVICE_STATUS_CHANGES_TO_SUSPENDED and cpiDEVICE_STATUS_CHANGES_TO_RUNNING will activate the predefined event channel APL_EVENT, see System Application Objects manual for more information. When cpiSendStandardSystemMessageWithBinary function is called, an ANSI analog input status point with settings UN=0, OA=address and an ANSI binary input status point with settings UN=0, OA=address+16777216 (1000000hex) in the receiving application are updated. Values for the binary object are 1=OK, 0=not OK. The usage of this function is preferred instead of cpiSendSystemMessage.

int cpiSendStandardSystemMessageWithBinary(int con, int source, int address, int newState, short status)
where

| | |
|---|---|
| con | Connection number to destination application |
| source | Source logical device, the following macro should be used to calculate the value LOGICAL_ADDRESS(deviceType, deviceNumber) |
| address | address of the ANSI analog input process object to be updated. The address of binary input status object is address+16777216 |
| newState | cpiDEVICE_STATUS_NOT_CHANGED<br>;Device status is not changed (binary input status object not updated)<br>cpiDEVICE_STATUS_CHANGES_TO_SUSPENDED<br>;Device status changes to suspended (binary input status object updated to 0)<br>cpiDEVICE_STATUS_CHANGES_TO_RUNNING<br>;Device status changes to running (binary input status object updated to 1) |
| status | status value for the ANSI analog input process object |

Example:

; Send a 'RUNNING' status from SPA device 2, analog input status object will be updated with value 0 (no error) and the binary input will be updated with value 1 (connection OK)

```
cpiSendStandardSystemMessageWithBinary( con,
                                        LOGICAL_ADDRESS(SPA_DEVICE, 2
                                        1002,
                                        cpiDEVICE_STATUS_CHANGES_TO_RUNNING,
                                        0);
```

## 5.2.8 cpiSendNodeSystemMessage

This function can be used to send a system message from the CPI node itself. Calling this function will update the ANSI analog input process object with the setting UN=0, OA=address. The value of the process object can be given with the parameter STATUS but for correct co-operation with System Self Supervision, it is recommended to use the value 10001=NETP_SYSTEM_INITIALIZED. SSS uses this process object in its displays to indicate the status of the node.

int cpiSendNodeSystemMessage(int con, int address, short status)
where

| | |
|---|---|
| con | Connection number to destination application |
| address | address of the ANSI analog input process object to be updated, value used by SSS is 6000+node number |
| status | status value for the ANSI analog input process object |

Example:

; Send NETP_SYSTEM_INITIALIZED status message indicating that the CPI node has started and established connection to the basesystem.

```
cpiSendNodeSystemMessage( con,
                          6002,
                          10001);
```

## 5.2.9 cpiSendNodeSystemMessageWithBinary

This function can be used to send a system message from the CPI node itself. Calling this function will update the ANSI analog input process object with the setting UN=0, OA=address, and the ANSI binary input process object with setting UN=0, OA=address + 1000000hex. The value of the analog input process object can be given with the parameter STATUS but for correct co-operation with System Self Supervision, it is recommended to use the value 10001=NETP_SYSTEM_INITIALIZED. The value of the binary input object will always be 1.

SSS uses these process objects in its displays to indicate the status of the node.

int cpiSendNodeSystemMessageWithBinary(int con, int address, short status)
where

| | |
|---|---|
| con | Connection number to destination application |
| address | address of the ANSI analog input process object to be updated, value used by SSS is 6000+node number |
| status | status value for the ANSI analog input process object |

Example:

; Send NETP_SYSTEM_INITIALIZED status message indicating that the CPI node has started and established connection to the basesystem.

```
cpiSendNodeSystemMessageWithBinary( con,
                                    6002,
                                    10001);
```

## 5.2.10 cpiSendStandardUserActivityMessage

This function can be used to send a user activity event system message from the station object to any device. When cpiSendStandardUserActivityMessage function is called, a corresponding ACP message is sent to the basesystem node. The content of the message is based on the information in the structure pointer by the pUAL_Event. Function is available only with separate agreement.

Parameter pathstr should refer to attribute MN

int cpiSendStandardUserActivityMessage (int con, UAL_EventType *pUALEvent, char *pathstr, int timeout, cpiMsgID *id);

| | |
|---|---|
| pUAL_EVENT | Pointer UAL_EventType Structure containing the user activity event information |
| pathstr | pointer to NULL-terminated string of max. 16 characters to identify the sender STA object |
| TIMEOUT | Timeout in ms |

## 5.2.11 cpiGetMessage

This function reads an ACP formatted message from the received message queue.

This function must not be called before the unpacking of the previous message is completely read, because the data of the previous message will be overwritten by this function.

int cpiGetMessage( int SOURCE, cpiMsgID *ID, *MSGTYPE);

| | |
|---|---|
| SOURCE | Source connection number |
| ID | Message buffer identification |
| MSGTYPE | Type of the received message,<br>1=write, 2=read, 3=reply,<br>4= notification |

Return values:

| | |
|---|---|
| 0 | : OK |
| -4 | : No messages from this source |

## 5.2.12 cpiClearExceptionStatus

This function clears a certain status code when an exception has occurred. If, for example, the function cpiException has returned 6=Application Error, the status can be cleared in order to retry the data transmission to the base system as well as enable the accessing of CPI Node from other MicroSCADA application. This function is needed, for example, in the Hot Standby implementation, if the main application receiving the data is not ready to accept data and causes an exception, access from WD application is still needed.

```
int cpiClearExceptionStatus(int CONNECTION, short c_status);
```

| | |
|---|---|
| Example: | if cpiException has returned 6 = application error, the status should be cleared with a call |

```
cpiClearExceptionStatus(mConn, SYST_APPLICATION_DOES_NOT_EXIST);
```

Return values:

| | |
|---|---|
| 0 | : OK |

## 5.2.13 cpiSetApplicationCold

This function sets the value of the "data not accepted" flag of the given application. This function is not necessarily used but it can be used if there is a need clear the mentioned flag. See also cpiGetApplicationCold.

```
void cpiSetApplicationCold( int CONNECTION, int STATE);
```

| | |
|---|---|
| CONNECTION | Source connection number |
| STATE | 0=application accepts data<br>1=application does not accept data |

Return values: none

## 5.2.14 cpiGetApplicationCold

This function returns the value of the "data not accepted" flag of the given application. This function is needed in Hot Standby implementations in order to know if the next transmission to the application should be done or not. In case an error SYST_APPLICATION_DOES_NOT_EXIST=7049 has been returned to the previous data transmission, the "data not accepted" flag of the receiving application is set to 1 to indicate that the application is not ready to accept data. The retransmission can be done after a delay to see if the situation has changed. In MicroSCADA version 9.3 and newer, the application which is in COLD state may also accept data since it uses its own buffering.

```
int cpiGetApplicationCold( int CONNECTION);
```

| | |
|---|---|
| CONNECTION | Source connection number |

Return values:

| | |
|---|---|
| 0 | : application accepts data |
| 1 | : application does not accept data |

# 5.3 Communication control services

These functions control the failed message transmission.

*Table 8: Communication control services*

| Function | Description |
|---|---|
| cpiGetFailedMessage | Returns the ID of the failed message. |
| cpiRetransmitMessage | Retransmits the message, which is in the failed message queue. |
| cpiGiveUp | Deletes a message from the failed message queue. |

## 5.3.1 cpiGetFailedMessage

This function returns the id of the failed message. The failed message has not received a reply from the application.

int cpiGetFailedMessage(int DESTIN, cpiMsgID *ID);

| | |
|---|---|
| ID | Id for failed message |
| DESTIN | Destination connection number |

Return values:

| | |
|---|---|
| 0 | : OK |
| -4 | : No failed messages in this connection |

## 5.3.2 cpiRetransmitMessage

This function retransmits the message, which is in the failed message queue.

```
int cpiRetransmit(cpiMsgId ID);
```

Return values:

| | |
|---|---|
| 0 | : OK, retransmission started |
| -2 | : Invalid MsgID |

## 5.3.3 cpiGiveUp

This function deletes a message from the failed message queue.

```
int cpiGiveUp(cpiMsgId ID);
```

Return values:

| | |
|---|---|
| 0 | : OK |
| -2 | : Invalid MsgID |

# 5.4 Message filtering services

Normally, only messages from the base system to station devices (RTU, SPA) configured in the CPI application node are visible for the CPI functions. To be able to emulate a conventional NET

unit operation, there is a need to also access messages directed to the CPI application node itself (NET object). This filtering can be controlled and monitored with the following functions.

When the filtering is switched off, all messages except diagnostic messages (the NET object attribute names NS and DS are reserved for the internal use of CPI) directed to the CPI node are passed through the interface and can be further processed with the CPI functions.

*Table 9: Message filtering*

| Function | Description |
|---|---|
| cpiNetMessageFiltering | Monitors the filtering of messages sent by the base system. |
| cpiSetNetMessageFiltering | Controls the filtering of messages sent by the base system. |

### 5.4.1 cpiNetMessageFiltering

The current state of the filtering can be read by this function.

```
int cpiNetMessageFiltering();
```

Return values:

```
0 = off
1 = on
```

### 5.4.2 cpiSetNetMessageFiltering

Controls filtering of messages sent by the base system.

```
void cpiNetMessageFiltering(int value);

value:
0 = off
1 = on (default)
```

## 5.5 RTU and SPA process data sending functions

The following functions are also considered to be automatic functions for building and sending certain ACP messages to the SYS600 process database. Each **Send...** function reserves a buffer for message, fills it and sends it to the base system.

Each one of the following functions can also be made manually. Instructions for this can be found in Section 5.6 and in Section 5.7.

### 5.5.1 RTU – application level data transmission services

The RTU Object Model is based on an object type and object block number concept (see SYS600 Application Objects). For input process data the object block number range is 1…255. For commands (from the SYS600 application to the CPI application) the range is 1…2047.

*Table 10: RTU - application level data transmission services*

| Function | Description |
|---|---|
| cpiSendRtuStatusMessage | Sends an ACP formatted notification message to SYS600. |
| cpiSendRtuIndication | Sends indication values to the base system process database. |
| cpiSendRtuIndicationBlock | Sends an indication value block to the base system process database. |
| Table continues on next page | |

| Function | Description |
|---|---|
| cpiSendRtuIndicationWithTS | Sends the indication values with timestamp to the base system process database. |
| cpiSendRtuAnalogValue | Sends an analog value to the base system process database. |
| cpiSendRtuAnalogValueWithTS | Sends an analog value with timestamp to the base system process database. |
| cpiSendRtuPulseCounter | Sends a pulse counter value to the base system process database. |
| cpiSendRtuPulseCounterWithTS | Sends a pulse counter value with timestamp to the base system process database. |
| cpiSendRtuBitStream | Sends a bit string to the process object database of the base system. |

### 5.5.1.1 cpiSendRtuStatusMessage

This function sends an ACP formatted notification message to SYS600. A reply to this message is not expected.

int cpiSendRtuStatusMessage(int DESTIN, int SOURCE, int STYPE, int DATAVALID, short STATUS );

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source RTU number |
| STYPE | Status type<br>0 -> Device status message<br>1 -> Terminal status message<br>2 -> Terminal event message<br>3 -> Terminal message |
| DATAVALID | 0 -> data is not longer valid in this RTU.<br>1-> data is valid in this RTU |
| STATUS | Status code |

Return values:

| | |
|---|---|
| 0 | : OK |
| -13 | : No buffer -> CPI cannot get a buffer for the message |

After starting the application program (the connections to the base systems and the remote RTU should be established), it is important to send the system status message to the base system. The status parameter should be set as OK_STATUS (or RTUP_DEVICE_STARTED) and the DATAVALID parameter as TRUE. The base system marks the object status related to the process data of the source RTU to OK (OS=0) or OBSOLETE_STATUS (OS=2) based on the DATAVALID parameter.

The following status codes are valid when the front-end emulates the RTU device (see SYS600 Status Codes):

**RTU level status codes:**

| | |
|---|---|
| 0 | OK_STATUS |
| 12602 | RTUP_DEVICE_SUSPENDED |
| 12603 | RTUP_TIMEOUT_WHILE_WAITING_RESPONSE |
| 12604 | RTUP_DEVICE_STOPPED |

Table continues on next page

| | |
|---|---|
| 12605 | RTUP_DEVICE_STARTED |
| 12658 | RTUC_INVALID_INDEX_RANGE |
| 12662 | RTUC_ATTRIBUTE_IS_WRITE_ONLY |
| 12665 | RTUC_NOT_EXECUTED_RESPONSE |
| 12676 | RTUC_UNEXPECTED_VALUE |

### Descriptions:

0 OK_STATUS

The purpose of this status message is to indicate that the CPI application program considers the RTU connection to be working properly. This should always be sent right after the start-up situation, when a connection has been established. It should also be sent after a communication break, when the connection has been re-established.

12602 RTUP_DEVICE_SUSPENDED

When the CPI application program activates an RTU device image for the process communication, this is the first system message to be sent. It indicates that the CPI application program does not currently have the connection to this RTU but it is trying to establish it since this is the RTU connection start-up situation.
It is also sent when the CPI application program detects that the connection to a certain RTU is lost. The DATAVALID parameter is set to false to indicate that the process database image of this RTU is no longer valid (see above).

12603 RTUP_TIMEOUT_WHILE_WAITING_RESPONSE

This status message indicates that the RTU did not react to the message initiated from a SYS600 application. Note that this means no response at all. The negative acknowledgements are handled by the "Not executed response" status code.

12604 RTUP_DEVICE_STOPPED

The RTU device emulated in the CPI application program can be stopped by setting its IU attribute to zero. This means that the connection to the RTU is properly disconnected. When this is done, the RTU image should send this status message to the SYS600 application. The DATAVALID flag should be set to FALSE since the updating of the SYS600 process database now stops from this RTU image.

12605 RTUP_DEVICE_STARTED

When the IU attribute of a device is set to one (1), the RTU image should respond by this system message. If this is the first time the RTU image is started after start-up, the DATAVALID should be set to FALSE. If the RTU image receives an IU attribute while the device connection is OK, the DATAVALID is set to TRUE.

12658 RTUC_INVALID_INDEX_RANGE

If the SYS600 application attempts to update an attribute with an invalid index (e.g. index equal to 5 when the highest allowed index is 4), the RTU image responds with this status message.

12662 RTUC_ATTRIBUTE_IS_WRITE_ONLY

The RTU image responds with this status message for attempts to read values from attributes that can only be written. A typical example might be some of the command type attributes.

12665 RTUC_NOT_EXECUTED_RESPONSE

This status message is sent to the SYS600 application when the RTU responds to the command with a negative acknowledgement from the application.

12676 RTUC_UNEXPECTED_VALUE

This status message indicates the attempts to write an invalid value to a RTU image attribute, e.g. a text value to an integer type attribute.

### Node level status codes:

| | |
|---|---|
| 14016 | NETW_UNKNOWN_DESTINATION_DEVICE |
| 14018 | NETW_UNKNOWN_DEVICE_ATTRIBUTE |
| 14044 | NETW_INVALID_STATION_ADDRESS |

### 5.5.1.2 cpiSendRtuIndication

This function sends an indication value to the base system process database.

```
int cpiSendRtuIndication(int DESTIN, int SOURCE, int BLOCK, int BIT, int VAL, int STATUS, int TIMEOUT,
cpiMsgID *ID);
```

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source RTU number |
| BLOCK | Object address of the process object |
| BIT | Bit number destination word (0..15) |
| VAL | Value of bit 0,1 for single indications, 10,11,12,13 for double indications |
| BVAL | 16 bit value |
| TS | Timestamp |
| TIMEOUT | Timeout in ms; |
| STATUS | Status code for data (0= ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : buffer full, the filling of the ACP message has been failed |
| -13 | : cannot get buffer for transmission message |
| -20 | : connection waiting<br>Sending failed because of the reply for the previously sent message |
| -22 | : Invalid value |

### 5.5.1.3 cpiSendRtuIndicationBlock

This function sends an indication value block to the base system process database.

```
int cpiSendRtuIndicationBlock(int DESTIN, int SOURCE, int BLOCK, int BVAL, int STATUS, int TIMEOUT,
cpiMsgID *ID);
```

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source RTU number |
| BLOCK | Object address of the process object |
| BIT | Bit number destination word (0..15) |
| VAL | Value of bit 0,1 for single indications, 10,11,12,13 for double indications |
| BVAL | 16 bit value |
| TS | Timestamp |
| TIMEOUT | Timeout in ms; |
| STATUS | Status code for data (0= ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : buffer full, the filling of the ACP message has been failed |
| -13 | : cannot get buffer for transmission message |
| -20 | : connection waiting<br>The sending failed because the reply for the previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.1.4 cpiSendRtuIndicationWithTS

This function sends an indication value with timestamp to the base system process database.

int cpiSendRtuIndicationWithTS(int DESTIN, int SOURCE, int BLOCK, int BIT, int VAL, cpiTime TS, int STATUS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source RTU number |
| BLOCK | Object address of the process object |
| BIT | Bit number destination word (0..15) |
| VAL | Value of bit 0,1 for single indications, 10,11,12,13 for double indications |
| BVAL | 16 bit value |
| TS | Timestamp |
| TIMEOUT | Timeout in ms; |
| STATUS | Status code for data (0= ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : buffer full, the filling of the ACP message has been failed |
| -13 | : cannot get buffer for transmission message |
| -20 | : connection waiting<br>The sending failed because the reply for the previously sent message |
| -22 | : Invalid value |

### 5.5.1.5 cpiSendRtuAnalogValue

This function sends an analog value to the base system process database.

int cpiSendRtuAnalogValue(int DESTIN, int SOURCE, int BLOCK, int VAL,int STATUS, int FLAGS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source RTU number |
| BLOCK | Object address of process object |
| VAL | Value of an analog value, the value range is [-2000...+2000].<br>Incoming values must be scaled to this range. |
| TS | Timestamp |

Table continues on next page

| | |
|---|---|
| TIMEOUT | Timeout in ms; |
| STATUS | Status code for data (0= ok) |
| FLAGS | Status flags of analog value process objects<br>bit0 -> Lo Alarm flag<br>bit1 -> Lo Warning flag<br>bit2 -> Hi Warning flag<br>bit3 -> Hi Alarm flag |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : buffer full, the filling of the ACP message has been failed |
| -13 | : cannot get buffer for transmission message |
| -20 | : connection waiting<br>The sending failed because of the reply for the previously sent message |
| -22 | : Invalid value |

### 5.5.1.6 cpiSendRtuAnalogValueWithTS

This function sends an analog value with timestamp to the base system process database.

int cpiSendRtuAnalogValueWithTS(int DESTIN, int SOURCE, int BLOCK, int VAL, cpiTime TS, int STATUS, int FLAGS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source RTU number |
| BLOCK | Object address of process object |
| VAL | Value of an analog value, the value range is [-2000...+2000].<br>Incoming values must be scaled to this range. |
| TS | Timestamp |
| TIMEOUT | Timeout in ms; |
| STATUS | Status code for data (0= ok) |
| FLAGS | Status flags of analog value process objects<br>bit0 -> Lo Alarm flag<br>bit1 -> Lo Warning flag<br>bit2 -> Hi Warning flag<br>bit3 -> Hi Alarm flag |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : buffer full, the filling of the ACP message has been failed |
| -13 | : cannot get buffer for transmission message |
| -20 | : connection waiting<br>The sending failed because of the reply for the previously sent message |
| -22 | : Invalid value |

### 5.5.1.7 cpiSendRtuPulseCounter

This function sends a pulse counter value to the base system process database.

int cpiSendRtuPulseCounter(int DESTIN, int SOURCE, int BLOCK, long VAL,int STATUS,int FLAGS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source RTU number |
| BLOCK | Object address of the process object |
| VAL | Value of the pulse counter |
| TS | Timestamp |
| TIMEOUT | Timeout in milliseconds |
| STATUS | Status code for data (0= ok) |
| FLAGS | Status flags of the pulse counter process objects<br>bit4 -> End of the period flag |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : Buffer full, the filling of the ACP message has been failed |
| -13 | : Cannot get buffer for transmission message |
| -20 | : Connection waiting<br>The sending failed because the reply to the previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.1.8 cpiSendRtuPulseCounterWithTS

This function sends a pulse counter value with timestamp to the base system process database.

int cpiSendRtuPulseCounterWithTS(int DESTIN, int SOURCE, int BLOCK, long VAL, cpiTime TS, int STATUS, int FLAGS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source RTU number |
| BLOCK | Object address of the process object |
| VAL | Value of the pulse counter |
| TS | Timestamp |
| TIMEOUT | Timeout in milliseconds |
| STATUS | Status code for data (0= ok) |
| FLAGS | Status flags of the pulse counter process objects<br>bit4 -> End of the period flag |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : Buffer full, the filling of the ACP message has been failed |
| -13 | : Cannot get buffer for transmission message |
| -20 | : Connection waiting<br>-The sending failed because the reply to the previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.1.9 cpiSendRtuBitStream

This function sends a bit string to the process object database of the base system.

```
int cpiSendRtuBitStream(int DESTIN, int SOURCE, int ADDR, int LEN, char STREAM[], int TIMEOUT, cpiMsgID
*ID);
```

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source RTU number |
| ADDR | Address of the process object |
| LEN | Length of bitstream in bits !! |
| STREAM | Array of bits stored on the char array<br>(Bitstream begins fromv bit0 of first char) |
| TIMEOUT | Timeout in milliseconds |
| FLAGS | Status flags of the pulse counter process objects<br>bit4 -> End of the period flag |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : Buffer full, the length of the bit stream is too long. |

## 5.5.2 SPA - application level data transmission services

The following functions reserve a buffer for a message, fill and send it from the SPA type device to the base system.

*Table 11: SPA - application level data transmission services*

| Function | Description |
|---|---|
| cpiSendSpaAnalogValueWithTSandFlags | Sends an analog value with alarm flags and timestamp to the base system process database. |
| cpiSendSpaAnalogValueWithTS | Sends an analog value with timestamp to the base system process database. |
| cpiSendSpaAnalogValue | Sends an analog value to the base system process database. |
| cpiSendSpaDigitalValueWithTSandFlags | Sends a digital value with alarm flags and timestamp to the base system process database. |
| cpiSendSpaDigitalValueWithTS | Sends a digital value with timestamp to the base system process database. |
| cpiSendSpaDigitalValue | Sends a digital value to the base system process database. |
| cpiSendSpaIndicationWithTS | Sends an indication value with timestamp to the base system process database. |
| cpiSendSpaIndicationBlockWithTS | Sends an indication value block with timestamp to the base system process database. |
| cpiSendSpaIndication | Sends an indication value to the base system process database. |
| cpiSendSpaIndicationBlock | Sends an indication value block to the base system process database. |
| cpiSendSpaPulseCounterWithTS | Sends a pulse counter value with timestamp to the base system process database. |
| Table continues on next page | |

| Function | Description |
|---|---|
| cpiSendSpaStatusMessage | Sends an ACP formatted notification message to SYS600. |
| cpiSendSpaMessage | Sends an ACP formatted message to SYS600 from SPA. |
| cpiSendSpaBitStream3 | Updates a bitstream process object in the base system process database. |
| cpiSendSpaBitStream3WithTS | Updates a bitstream process object with timestamp in the base system process database. |
| cpiSendBufferedSpaIndicationWithTS | Sends an indication value with timestamp to the base system process database. Indication is marked as buffered. |
| cpiSendSpaBufferedIndicationBlockWithTS | Sends an indication value block with timestamp to the base system process database. Indication block is marked as buffered. |

### 5.5.2.1 cpiSendSpaAnalogValueWithTSandFlags

This function sends an analog value with timestamp to the base system process database. The parameter 'FLAGS' is used to provide alarm information of the measured value.

```
int cpiSendSpaAnalogValueWithTSandFlags(int DESTIN, int SOURCE, int BLOCK, float VAL, cpiTime TS, int
STATUS, int FLAGS, int TIMEOUT, cpiMsgID *ID);
```

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| BLOCK | Object address of process object |
| VAL | Value of an analog value |
| TS | Timestamp |
| TIMEOUT | Timeout in milliseconds |
| STATUS | Status code for data (0= ok) |
| FLAGS | Status flags of analog value process objects<br>bit0 -> Lo Alarm flag<br>bit1 -> Lo Warning flag<br>bit2 -> Hi Warning flag<br>bit3 -> Hi Alarm flag |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : Buffer full, the filling of the ACP message failed |
| -13 | : Cannot get buffer for transmission message<br>-The sending failed because the reply to the previously sent message has not been received |
| -20 | : Connection waiting |
| -22 | : Invalid value |

### 5.5.2.2 cpiSendSpaAnalogValueWithTS

This function sends an analog value with timestamp to the base system process database.

int cpiSendSpaAnalogValueWithTS(int DESTIN, int SOURCE, int BLOCK, float VAL, cpiTime TS, int STATUS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| BLOCK | Object address of process object |
| VAL | Value of an analog value |
| TS | Timestamp |
| TIMEOUT | Timeout in milliseconds |
| STATUS | Status code for data (0= ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : Buffer full, the filling of the ACP message failed |
| -13 | : Cannot get buffer for transmission message |
| -20 | : Connection waiting, previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.2.3 cpiSendSpaAnalogValue

This function sends an analog value without timestamp to the base system process database.

int cpiSendSpaAnalogValue(int DESTIN, int SOURCE, int BLOCK, float VAL, int STATUS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| BLOCK | Object address of the process object |
| VAL | Value of an analog value |
| TIMEOUT | Timeout in milliseconds |
| STATUS | Status code for data (0 = ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : Buffer full, the filling of the ACP message failed |
| -13 | : Cannot get buffer for transmission message |
| -20 | : Connection waiting<br>-The sending failed because the reply to the previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.2.4 cpiSendSpaDigitalValueWithTSandFlags

This function sends a digital value with timestamp to the base system process database. The parameter FLAGS is used to provide alarm information of the measured value.

int cpiSendSpaDigitalValueWithTSandFlags(int DESTIN, int SOURCE, int BLOCK, short VAL, cpiTime TS, int STATUS, int FLAGS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| BLOCK | Object address of the process object |
| VAL | Digital value |
| TS | Timestamp |
| TIMEOUT | Timeout in milliseconds |
| STATUS | Status code for data (0 = ok) |
| FLAGS | Status flags of analog value process objects<br>bit0 -> Lo Alarm flag<br>bit1 -> Lo Warning flag<br>bit2 -> Hi Warning flag<br>bit3 -> Hi Alarm flag |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : Buffer full, the filling of the ACP message failed |
| -13 | : Cannot get buffer for transmission message |
| -20 | : Connection waiting<br>-The sending failed because the reply to the previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.2.5 cpiSendSpaDigitalValueWithTS

This function sends a digital value with timestamp to the base system process database.

int cpiSendSpaDigitalValueWithTS(int DESTIN, int SOURCE, int BLOCK, short VAL, cpiTime TS, int STATUS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| BLOCK | Object address of the process object |
| VAL | Digital value |
| TS | Timestamp |
| TIMEOUT | Timeout in milliseconds |
| STATUS | Status code for data (0 = ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : Buffer full, the filling of the ACP message failed |
| -13 | : Cannot get buffer for transmission message |
| -20 | : Connection waiting<br>-The sending failed because the reply to the previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.2.6 cpiSendSpaDigitalValue

This function sends a digital value without timestamp to the base system process database.

int cpiSendSpaDigitalValue(int DESTIN, int SOURCE, int BLOCK, short VAL, int STATUS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| BLOCK | Object address of process object |
| VAL | Digital value |
| TS | Timestamp |
| TIMEOUT | Timeout in ms; |
| STATUS | Status code for data (0= ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : buffer full, the filling of the ACP message failed |
| -13 | : cannot get buffer for transmission message |
| -20 | : connection waiting<br>-The sending failed because the reply to the previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.2.7 cpiSendSpaIndicationWithTS

This function sends an indication value with timestamp to the base system process database.

int cpiSendSpaIndicationWithTS(int DESTIN, int SOURCE, int BLOCK, int BIT, int VAL, cpiTime TS, int STATUS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| BLOCK | Object address of process object |
| BIT | Bit number destination word (0..15) |
| VAL | Value of bit 0,1 for single indications, 10,11,12,13 for double indications |
| BVAL | 16 bit value |
| TS Timestamp | Timestamp |
| TIMEOUT | Timeout in ms; |
| STATUS | Status code for data (0= ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : buffer full, the filling of the ACP message failed |
| -13 | : cannot get buffer for transmission message |
| -20 | : connection waiting<br>-The sending failed because the reply to the previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.2.8 cpiSendSpaIndicationBlockWithTS

This function sends an indication value block with timestamp to the base system process database.

int cpiSendSpaIndicationBlockWithTS(int DESTIN, int SOURCE, int BLOCK, int BVAL, cpiTime TS, int STATUS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| BLOCK | Object address of process object |
| BIT | Bit number destination word (0..15) |
| VAL | Value of bit 0,1 for single indications, |
| BVAL | 16 bit value |
| TS | Timestamp |
| TIMEOUT | Timeout in ms; |
| STATUS | Status code for data (0= ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : buffer full, the filling of the ACP message failed |
| -13 | : cannot get buffer for transmission message |
| -20 | : connection waiting<br>-The sending failed because the reply to the previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.2.9 cpiSendSpaIndication

This function sends an indication value to the base system process database.

int cpiSendSpaIndication(int DESTIN, int SOURCE, int BLOCK, int BIT, int VAL, int STATUS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| BLOCK | Object address of process object |
| BIT | Bit number destination word (0..15) |
| VAL | Value of bit 0,1 for single indications, |
| BVAL | 16 bit value |
| TIMEOUT | Timeout in ms; |
| STATUS | Status code for data (0= ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : Buffer full, the filling of the ACP message failed |
| -13 | : Cannot get buffer for transmission message |
| -20 | : connection waiting<br>-The sending failed because the reply to the previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.2.10 cpiSendSpaIndicationBlock

This function sends an indication value block to the base system process database.

int cpiSendSpaIndicationBlock(int DESTIN, int SOURCE, int BLOCK, int BVAL, int STATUS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| BLOCK | Object address of process object |
| BIT | Bit number destination word (0..15) |
| VAL | Value of bit 0,1 for single indications,<br>10,11,12,13 for double indications |
| BVAL | 16 bit value |
| TIMEOUT | Timeout in ms; |
| STATUS | Status code for data (0= ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : Buffer full, the filling of the ACP message failed |
| -13 | : Cannot get buffer for transmission message |
| -20 | : connection waiting<br>-The sending failed because the reply to the previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.2.11 cpiSendSpaPulseCounterWithTS

This function sends a pulse counter value with timestamp to the base system process database.

int cpiSendSpaPulseCounterWithTS(int DESTIN, int SOURCE, int BLOCK, long VAL, cpiTime TS, int STATUS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| BLOCK | Object address of process object |
| VAL | Value of the pulse counter |
| TS | Timestamp |
| TIMEOUT | Timeout in milliseconds |
| STATUS | Status code for data (0 = ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : Buffer full, the filling of the ACP message failed |
| -13 | : Cannot get buffer for transmission message |
| -20 | : Connection waiting<br>-The sending failed because the reply to the previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.2.12 cpiSendSpaStatusMessage

This function sends an ACP formatted notification message to SYS600.

int cpiSendSpaStatusMessage(int DESTIN, int SOURCE, int STYPE, int DATAVALID, short STATUS );

| | |
|---|---|
| SOURCE | Destination connection number |
| STYPE | Status type<br>0 -> Device status message<br>1 -> Terminal status message<br>2 -> Terminal event message<br>3 -> Terminal message |
| DATAVALID | 0 -> Data is not longer valid in this spa unit.<br>1-> Data is valid in this spa |
| STATUS | Status code |

Return values:

| | |
|---|---|
| 0 | : OK |
| -13 | : No buffer -> cpi cannot get buffer for message |

The following status codes are valid when the front-end emulates the SPA device (see SYS600 Status Codes):

**SPA status codes:**

| | |
|---|---|
| 0 | OK_STATUS |
| 13201 | SPAC_SC_DATA_OVERFLOW |
| 13202 | SPAC_TOO_LONG_REPLY_RECEIVED |
| 13203 | SPAC_SPA_ADDRESS_NOT_CONFIGURED |
| 13204 | SPAC_DEVICE_MUST_BE_ALLOCATED |
| 13205 | SPAC_UNKNOWN_DIAGNOSTIC_COUNTER |
| 13206 | SPAC_INVALID_INDEX_RANGE |
| 13207 | SPAC_INVALID_RT_ATTRIBUTE_VALUE |
| 13208 | SPAC_UNKNOWN_SPA_ATTRIBUTE |
| 13209 | SPAC_ILLEGAL_APPLICATION_FOR_OPERATION |
| 13210 | SPAC_ATTRIBUTE_IS_WRITE_ONLY |
| 13211 | SPAC_ILLEGAL_BROADCAST_ACTION |
| 13212 | SPAC_UNEXPECTED_RESPONSE |
| 13213 | SPAC_INVALID_ATTRIBUTE_VALUE |
| 13214 | SPAC_CHAR_TYPE_EXPECTED |
| 13215 | SPAC_INTERNAL_ERROR |

Table continues on next page

| | |
|---|---|
| 13216 | SPAC_INVALID_POINT_DEFINITION |
| 13217 | SPAC_DATABASE_UPDATE_COMPLETED |
| 13218 | SPAC_TOO_MANY_ARGUMENTS |
| 13219 | SPAC_ARGUMENT_EXPECTED |
| 13220 | SPAC_UNABLE_TO_ALLOCATE_MEMORY |
| 13221 | SPAC_POINT_DEFINITION_NOT_FOUND |
| 13222 | SPAC_NO_ACKNOWLEDGE_RESPONSE |
| 13223 | SPAC_UNEXPECTED_VALUE_TYPE |
| 13224 | SPAC_ILLEGAL_OBJECT_TYPE |
| 13225 | SPAC_ONLY_WRITE_IS_POSSIBLE |
| 13226 | SPAC_NO_ACKNOWLEDGE_REPLY |
| 13227 | SPAC_EVENT_DATA_DISCREPANCY_DETECTED |
| 13251 | SPAP_DEVICE_SUSPENDED |
| 13252 | SPAP_DEVICE_STOPPED |
| 13253 | SPAP_DEVICE_STARTED |
| 13254 | SPAP_OUT_OF_BUFFERS |
| 13255 | SPAP_TIMEOUT_WHILE_WAITING_RESPONSE |
| 13256 | SPAP_UNEXPECTED_BC_REPLY |
| 13257 | SPAP_BROADCAST_FAILURE |
| 13258 | SPAP_DATABASE_UPDATE_COMPLETED |
| 17201 | SPCP_TIMEOUT_WHILE_WAITING_CTS |
| 17202 | SPCP_CTS_LINE_FAILURE_WHILE_TRANSM |
| 17203 | SPCP_REDUNDANCY_ERRORS_IN_RESPONSE |
| 17205 | SPCP_TIMEOUT_WHILE_WAITING_RESPONSE |
| 17211 | SPPC_INVALID_ATTRIBUTE_VALUE |

**Node level:**

| | |
|---|---|
| 14016 | NETW_UNKNOWN_DESTINATION_DEVICE |
| 14018 | NETW_UNKNOWN_DEVICE_ATTRIBUTE |

### 5.5.2.13 cpiSendSpaMessage

This function sends an ACP formatted message to SYS600 from SPA. Before using this function, the message buffer must be reserved by using the function cpiGetTransmissionBuffer and formatted by using some of the cpiPutXXXData functions presented in Section 5.6.2. (Notation XXX means that any function of this type is available.)

int cpiSendSpaMessage(int DESTIN, int SOURCE, cpiMsgID ID, int TIMEOUT);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| ID | Message buffer identification |
| TIMEOUT | Waiting time for the base system reply in milliseconds,<br>0-> no time-out supervision |

Return values:

| | |
|---|---|
| 0 | : OK |
| -2 | : Invalid MsgID |

### 5.5.2.14 cpiSendSpaBitStream3

This function updates a bitstream process object in the base system process database.

```
int cpiSendSpaBitStream3(int DESTIN, int SOURCE, int ADDR, int LEN, char STREAM[], int STATUS, int
TIMEOUT, cpiMsgID *ID);
```

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| ADDR | Address of the process object |
| LEN | Length of bitstream in bits |
| STREAM | Array of bits stored on the char array (Bitstream begins from bit 0 of first char) |
| STATUS | Status code for data (0 = OK) |
| TIMEOUT | Timeout in milliseconds |
| FLAGS | Status flags of the pulse counter |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : Buffer full, the length of the bit stream is too long |

### 5.5.2.15 cpiSendSpaBitStream3WithTS

This function updates a bitstream process object in the base system process database. Timestamp of the process object is updated with a given time.

```
int cpiSendSpaBitStream3(int DESTIN, int SOURCE, int ADDR, int LEN, char STREAM[], int STATUS, cpiTime TS,
int TIMEOUT, cpiMsgID *ID);
```

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| ADDR | Address of the process object |
| LEN | Length of bitstream in bits |
| STREAM | Array of bits stored on the char array (Bitstream begins from bit 0 of first char) |
| STATUS | Status code for data (0 = OK) |
| TS | Timestamp |
| TIMEOUT | Timeout in milliseconds |
| FLAGS | Status flags of the pulse counter |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : Buffer full, the length of the bit stream is too long |

### 5.5.2.16 cpiSendBufferedSpaIndicationWithTS

This function sends an indication value with timestamp to the base system process database. The indication is marked with BE_SUBITEM, which can be used to indicate that the data has been buffered in the application.

int cpiSendSpaBufferedIndicationWithTS(int DESTIN, int SOURCE, int BLOCK, int BIT,int VAL, cpiTime TS, int STATUS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| BLOCK | Object address of process object |
| BIT | Bit number destination word (0..15) |
| VAL | VAL Value of bit 0,1 for single indications,10,11,12,13 for double indications |
| BVAL | 16 bit value |
| TS | Timestamp |
| TIMEOUT | Timeout in ms; |
| STATUS | Status code for data (0= ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : buffer full, the filling of the ACP message failed |
| -13 | : cannot get buffer for transmission message |
| -20 | : connection waiting<br>-The sending failed because the reply to the previously sent message has not been received |
| -22 | : Invalid value |

### 5.5.2.17 cpiSendSpaBufferedIndicationBlockWithTS

This function sends an indication value block with timestamp to the base system process database. The indication is marked with BE_SUBITEM, which can be used to indicate that the data has been buffered in the application.

int cpiSendSpaBufferedIndicationBlockWithTS(int DESTIN, int SOURCE, int BLOCK, int BVAL, cpiTime TS, int STATUS, int TIMEOUT, cpiMsgID *ID);

| | |
|---|---|
| DESTIN | Destination connection number |
| SOURCE | Source SPA number |
| BLOCK | Object address of process object |
| BVAL | 16 bit value |
| TS | Timestamp |
| TIMEOUT | Timeout in ms; |
| STATUS | Status code for data (0= ok) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -7 | : buffer full, the filling of the ACP message failed |
| -13 | : cannot get buffer for transmission message |
| -20 | : connection waiting<br>-The sending failed because the reply to the previously sent message has not been received |
| -22 | : Invalid value |

# 5.6 Generic ACP packet building

All of the functions for sending RTU or SPA process data presented in the previous section can be done manually by using the following functions. It is possible to build up messages with them also for special purposes, usually when the functions needed do not exist. This can occur, for example, when several analog values need to be sent in a single message.

Before any of the following packet building functions can be called, the buffer where the message is going to be built up must be reserved by using the cpiGetTransmissionBuffer function.

## 5.6.1 Message header information

In order to be able to emulate a conventional NET unit operation, the following functions can be freely used for setting the logical destination and source.

*Table 12: Message header source*

| Function | Description |
|---|---|
| cpiPutSource | Sets the logical source of the message. |
| cpiPutDestination | Sets the logical destination of the message. |

### 5.6.1.1 cpiPutSource

With this function the logical source of the message can be set.

int cpiPutSource(cpiMsgID id, int SOURCE)

| | |
|---|---|
| SOURCE | SOURCE Source logical device, the following macro should be used to calculate the value LOGICAL_ADDRESS(deviceType, deviceNumber) |

Example:
;Setting the logical source for NET1:
cpiPutSource(id, LOGICAL_ADDRESS(NET_DEVICE,1));

Return values:

| | |
|---|---|
| 0 | : OK |
| -5 | : Unknown Message ID |

### 5.6.1.2 cpiPutDestination

With this function the logical destination of the message can be set.

int cpiPutDestination(cpiMsgID id, int DESTINATION)

| | |
|---|---|
| DESTINATION | Destination logical device, the following macro should be used to calculate the value LOGICAL_ADDRESS(deviceType,deviceNumber) |

Example:
;Setting the logical destination for APL1:
cpiPutSource(id, LOGICAL_ADDRESS(APL_DEVICE,1));

Return values:

| | |
|---|---|
| 0 | : OK |
| -5 | : Unknown Message ID |

## 5.6.2 Data filling functions

The ACP message can be filled with the process data by using the following functions.

*Table 13: Data filling functions*

| Function | Description |
|---|---|
| cpiPutData | Puts data to the ACP message. |
| cpiPutRtuData | Puts the process data with status to the ACP message. |
| cpiPutRtuDataWithoutStatus | Puts the process data without status to the ACP message. |
| cpiPutRtuDataWithTS | Puts the process data with timestamp to the ACP message. |

The object type determines also the type of the value. If the object type is 0=NO_OBJECT_TYPE, the type of the value is free.

### 5.6.2.1 cpiPutData

Puts data to the ACP message (See Table 5 and file cpi.h).

```
int cpiPutData(cpiMsgID id, cpiData *data)
```

### 5.6.2.2 cpiPutRtuData

This function adds the process data with status to the ACP message.

int cpiPutRtuData (cpiMsgID ID, cpiObjectType OT, int BLOCK, cpiData VAL, int STATUS);

| | |
|---|---|
| ID | Identification number for message |
| OT | Object type of RTU process object |
| BLOCK | Object address of RTU process object (1 ... 255) |
| VAL | Value for process object |
| TS | Time stamp |
| STATUS | Status code 0 -> OK |

Return values:

| | |
|---|---|
| 0 | : OK |
| -2 | : Invalid MsgID |
| -7 | : Message buffer is full, adding the data has not succeeded |

### 5.6.2.3 cpiPutRtuDataWithoutStatus

This function adds the process data without status to the ACP message.

int cpiPutRtuDataWithoutStatus(cpiMsgID ID, cpiObjectType OT, int BLOCK, cpiData VAL);

| | |
|---|---|
| ID | Identification number for message |
| OT | Object type of RTU process object |
| BLOCK | Object address of RTU process object<br>(1 ... 255) |
| VAL | Value for process object |
| TS | Time stamp |
| STATUS | Status code<br>0 -> OK |

Return values:

| | |
|---|---|
| 0 | : OK |
| -2 | : Invalid MsgID |
| -7 | : Message buffer is full, adding the data has not succeeded |

### 5.6.2.4 cpiPutRtuDataWithTS

This function adds the process data with timestamp to the ACP message.

int cpiPutRtuDataWithTS (cpiMsgID ID, cpiObjectType OT, int BLOCK, cpiData VAL, cpiTime TS, int STATUS);

| | |
|---|---|
| ID | Identification number for message |
| OT | Object type of RTU process object |
| BLOCK | Object address of RTU process object<br>(1 ... 255) |
| VAL | Value for process object |
| TS | Time stamp |
| STATUS | Status code<br>0 -> OK |

Return values:

| | |
|---|---|
| 0 | : OK |
| -2 | : Invalid MsgID |
| -7 | : Message buffer is full, adding the data has not succeeded |

### 5.6.3 Reply status filling functions

*Table 14: Reply status filling functions*

| Function | Description |
|---|---|
| cpiPutReplyStatus | Adds the status of the reply to the reply message. |

#### 5.6.3.1 cpiPutReplyStatus

The following function adds the status of the reply to the reply message. If the reply status is OK, there is no need to use this function.

```
int cpiPutReplyStatus (cpiMsgID ID, int STATUS);
```

| | |
|---|---|
| ID | Identification number for message |
| STATUS | Status code |

Return values:

| | |
|---|---|
| 0 | : OK |
| -2 | : invalid MsgID |

## 5.7 ACP packet unpacking

The message received from the SYS600 base system can be unpacked by using the following functions.

### 5.7.1 Message header information

The following functions are used to read the header information of the received message. See also Section 3.4.

*Table 15: Message header information*

| Function | Description |
|---|---|
| cpiGetDestination | Reads the station address of the destination node from the message (CPI application node). |
| cpiGetSource | Reads the station address of the source node from the message (SYS600 node). |
| cpiGetRtuNo | Reads the station number from the message. |
| cpiGetAttribute | Reads the attribute from the message. |
| cpiGetIndex | Reads the index from the message. |
| cpiGetReplyStatus | Reads the reply status of the message. |
| cpiGetReplyDefinition | Reads the reply definition of the message. |
| cpiGetMessageType | Reads the type of the incoming message. |
| cpiGetMessageLength | Reads the length of the received message. |
| cpiGetDeviceType | Retrieves the logical device type message. |
| cpiGetDeviceNumber | Retrieves the logical device number message. |

### 5.7.1.1 cpiGetDestination

This function reads the station address of the destination node from the message (CPI application node).

```
int cpiGetDestination( cpiMsgID ID, int *DESTIN);
```

| | |
|---|---|
| ID | Identification number for the message |
| DESTIN | The CPI application node station address |

Return values:

| | |
|---|---|
| 0 | : OK |
| -5 | : Invalid message buffer |

### 5.7.1.2 cpiGetSource

This function reads the station address of the source node from the message (SYS600 node).

```
int cpiGetSource( cpiMsgID ID, int *SOURCE);
```

| | |
|---|---|
| ID | Identification number for the message |
| SOURCE | Station address of the source |

Return values:

| | |
|---|---|
| 0 | : OK |
| -5 | : Invalid message buffer |

### 5.7.1.3 cpiGetRtuNo

This function checks which RTU station the message is meant to.

```
int cpiGetRtuNo(cpiMsgID ID, int *RTU);
```

| | |
|---|---|
| ID | Identification number for the message<br>RTU number |
| RTU | RTU number |

Return values:

| | |
|---|---|
| 0 | : OK |
| -5 | : Invalid message buffer |

### 5.7.1.4 cpiGetAttribute

This function reads the attribute of the message.

```
int cpiGetAttribute( cpiMsgID ID, char *ATTRIBUTE);
```

| | |
|---|---|
| ID | Identification number for the message |
| ATTRIBUTE | Attribute of the message |

Return values:

| | |
|---|---|
| 0 | : OK |
| -5 | : Invalid message buffer |

### 5.7.1.5 cpiGetIndex

This function reads the index of the message.

```
int cpiGetIndex( cpiMsgID ID, int *FIRST, int *LAST);
```

| | |
|---|---|
| ID | Identification number for the message |
| FIRST | First index of the message |
| LAST | Last index of the message |

Return values:

| | |
|---|---|
| 0 | : OK |
| -5 | : Invalid message buffer |

### 5.7.1.6 cpiGetReplyStatus

This function reads the reply status of the message.

```
int cpiGetReplyStatus( cpiMsgID ID, cpiMsgID *REPLYID, int *STATUS);
```

| | |
|---|---|
| ID | Identification number for the message |
| REPLYID | Id of the reply message, which is a response for the sent message |
| STATUS | Status of the reply |

Return values:

| | |
|---|---|
| 0 | : OK |
| -5 | : Invalid message buffer |

### 5.7.1.7 cpiGetReplyDefinition

This function reads the reply definition of the message.

```
int cpiGetReplyDefinition(cpiMsgID ID, cpiMsgHeader *HEADER);
```

| | |
|---|---|
| ID | Identification number for the message |
| HEADER | Header information of the message |

Return values:

| | |
|---|---|
| 0 | : OK |
| -5 | : Invalid message buffer |

### 5.7.1.8 cpiGetMessageType

This function reads the type of the message.

int cpiGetMessageType (cpiMsgID ID, int *MSGTYPE); 1=write, 2=read, 3=reply, 4= notification

| | |
|---|---|
| ID | Identification number for the message |
| MSGTYPE | Type of the incoming message |

Return values:

| | |
|---|---|
| 0 | : OK |
| -5 | : Invalid message buffer |

### 5.7.1.9 cpiGetMessageLength

This function reads the length of the message.

int cpiGetMessageLength (cpiMsgID ID, int *LENGTH);

| | |
|---|---|
| ID | identification number for the message |
| LENGTH | Length of the received message (the whole ACP message) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -5 | : Invalid message buffer |

### 5.7.1.10 cpiGetDeviceType

A new function will be implemented to retrieve the logical device type message.

```
int cpiGetDeviceType(cpiMsgID *msg, int *deviceType)
```

Return values (constants defined in cpi.h):

```
APL_DEVICE = 1
NET_DEVICE = 2
RTU_DEVICE = 4
IEC_DEVICE = 29
```

### 5.7.1.11 cpiGetDeviceNumber

A new function will be implemented to retrieve the logical number message.

```
int cpiGetDeviceNumber(cpiMsgID *msg, int *deviceNumber)
```

Return values:

| | |
|---|---|
| 0 | : OK |
| -1 | : If fails |

## 5.7.2 Data scanning functions

The data part of the message can be read with the following functions.

*Table 16: Data scanning functions*

| Function | Description |
|---|---|
| cpiGetNextData | Reads the next data item from the received message. |
| cpiGetNextDataType | Reads the type of the next item in the message. |
| cpiGetNextS32Data | Reads the next item from the message. |
| cpiGetTimeData | Retrieves the time information from the message written in RTU_ATIME format. |

### 5.7.2.1 cpiGetNextData

The following function reads the next data item from the received message. The function is used to read items one by one from the message to the cpiData structure. The function returns the value of the same type as the values in the message. Because the type of data may vary between messages, the type information of cpiData must be used when the data of the message is used by the CPI application program.

```
int cpiGetNextData (cpiMsgID ID, cpiData *VAL);
```

| | |
|---|---|
| VAL | Value (structure!) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -1 | : Connection error, the reading of the next item from the message has failed.<br>More detailed information can be read from the global variable **cpi_errno**.<br>The explanations for cpi_errno values can be read from the Status Codes manual. |
| -2 | :Invalid MsgID |
| -10 | : No data items left in the buffer |

### 5.7.2.2 cpiGetNextDataType

This function reads the type of the next item in the message. The function does not remove the item from the message.

```
int cpiGetNextDataType (cpiMsgID ID, cpiDataType *VAL);
```

| | |
|---|---|
| VAL | Value |

Return values:

| | |
|---|---|
| 0 | : OK |
| -1 | : Connection error, the reading of the next item from the message has failed for some reason.<br>More detailed information can be read from the global variable cpi_errno.<br>The explanation for cpi_errno values can be read from the Status Codes manual. |
| -2 | : Invalid MsgID |
| -10 | : No data items left in the buffer |

### 5.7.2.3 cpiGetNextS32Data

The following function reads the next item from the message. The item is converted to the long type. The items of the binary type cannot be read by using this function.

int cpiGetNextS32Data (cpiMsgID ID, long *VAL);

| | |
|---|---|
| VAL | VAL Value |

Return values:

| | |
|---|---|
| 0 | : OK |
| -1 | : Connection error, the reading of the next item from the message has failed for some reason.<br>More detailed information can be read from the global variable cpi_errno.<br>The explanation for cpi_errno values can be read from the Status Codes manual. |
| -2 | : Invalid MsgID |
| -10 | : No data items left in the buffer |

### 5.7.2.4 cpiGetTimeData

This function can be used to retrieve time information from the message written in RTU_ATIME format.

```
int cpiGetTimeData(cpiMsg *msg, cpiTime *time);
```

Return values:

| | |
|---|---|
| 0 | if ok |
| -1 | if fails |

## 5.7.3 RTU commands

### 5.7.3.1 General handling

This function can be used to read command data from the message.

*Table 17: Data scanning functions*

| Function | Description |
|---|---|
| cpiGetNextAddrData | Reads the next Address Type item from the message |

**cpiGetNextAddrData**
The following function reads the next Address Type item from the message. This can be used for unpacking the process commands from SYS600, which are sent via the process database. All of the commands that are sent via the SYS600 process database use the DA attribute and the Address Type data item. The data is divided into three parts: object type, object address and object value.

int cpiGetNextAddrData (cpiMsgID ID, cpiData *VAL);

| | |
|---|---|
| VAL | Value (structure!) |

Return values:

| | |
|---|---|
| 0 : OK | : OK |
| -1 : | : Connection error, the reading of the next item from the message has failed.<br>More detailed information can be read from the global variable **cpi_errno**.<br>The explanations for cpi_errno values can be read from the Status Codes manual. |
| -2 | : Invalid MsgID |
| -10 | : No data items left in the buffer |

Example of use:

```
cpiGetAttribute(id, AT);
if (!strcmp(AT,"DA"))
/* compare AT to see if it is DA = command via process database*/
{
   i = 0;
   printf("DA message received \n");
   while(cpiGetNextAddrData (id, &data) == 0) {
       printf(" OT = %d", data.value.ADDR.ObjectType );
       printf(" OA = %d", data.value.ADDR.ObjectAddress);
       printf(" OV = %d\n", data.value.ADDR.ObjectValue);
       switch(data.value.ADDR.ObjectType) {
       case cpiOBJECT_COMMAND:
           printf("Object Command received\n");
           switch(data.value.ADDR.ObjectValue) {
                case 0: printf("IMMEDIATE COMMAND OFF \n"); break;
                case 1: printf("IMMEDIATE COMMAND ON \n"); break;
                case 16: printf("INHIBIT COMMAND \n"); break;
                case 17: printf("CHECK BACK COMMAND \n"); break;
                case 32: printf("EXECUTE COMMAND OFF \n"); break;
                case 33: printf("EXECUTE COMMAND ON \n"); break;
            }
            break;
         case cpiANALOG_SETPOINT:
            printf("Analog Setpoint received\n");
            break;
             }
            }
}
```

### 5.7.3.2 RTU object command handling

The (byte) value received when a command is activated is byte coded as shown in Table 18 (see also the example of use in Section 5.7.3.1):

*Table 18: The received values and descriptions*

| Object Command | Status | High Nibble Value | Low Nibble Value |
|---|---|---|---|
| IMMEDIATE COMMAND | | 0 | |
| | ON | | 1 |
| | OFF | | 0 |
| CHECK BACK COMMAND | | 1 | |
| | INHIBIT | | 0 |
| | CHECK BACK | | 1 |
| EXECUTE COMMAND | | 2 | |
| | ON | | 1 |
| | OFF | | 0 |

*Table 19: Regulation commands*

| Object Command | Status | | |
|---|---|---|---|
| IMMEDIATE COMMAND | | 0 | 1 / 0 |
| COMMAND STOP | | 3 | Ignored |

### 5.7.3.3 RTU analog setpoint handling

The value of the analog setpoint is stored into the ObjectValue field of the cpiData structure.

## 5.8 Communication supervision

*Table 20: Communication supervision*

| Function | Description |
|---|---|
| cpiConnectionStatus | Checks the status of an application connection. |

### 5.8.1 cpiConnectionStatus

This function checks the status of an application connection. It also returns the status code of the last reply message, if it is rejected by the base system.

```
int cpiConnectionStatus (int DESTIN);
```

| | |
|---|---|
| DESTIN | Connection number of the destination |

Return values:

| | |
|---|---|
| 0 | : OK |
| 1 | : Waiting connection |
| others | : Refer to the Status Codes manual |

## 5.9 Buffer management

The following functions reserve buffers for different message types.

*Table 21: Buffer management*

| Function | Description |
|---|---|
| cpiGetTransmissionBuffer | Reserves a buffer for transmission. |
| cpiGetReplyBuffer | Reserves a buffer for a reply message transmission. |
| cpiGetDefinedReplyBuffer | Reserves a buffer for a reply message. |
| cpiGetNotificationBuffer | Reserves a buffer for a notification message. |

### 5.9.1 cpiGetTransmissionBuffer

This function reserves a buffer for transmission. The function returns with an error status if the transmission queue is full or if the failed transmission queue contains messages. The length of the transmission queue is 1 in this version, so the previously sent message must get a reply from the base system before a new buffer can be reserved.

int cpiGetTransmissionBuffer(int CON, cpiMsgID *ID);

| | |
|---|---|
| CON | Connection number of the destination |
| ID | Id for a message buffer |

Return values:

| | |
|---|---|
| 0 | : OK |
| -13 | : No buffers left |
| -20 | : Connection is still waiting for a reply for the previously sent message |

## 5.9.2 cpiGetReplyBuffer

This function reserves a buffer for a reply message transmission. It is used when it is possible to send the reply message back to the base system immediately.

int cpiGetReplyBuffer(cpiMsgID RQ, cpiMsgID *ID);

| | |
|---|---|
| RQ | Id of the message which is a response for the sent message |
| ID | Id for a reply message |

Return values:

| | |
|---|---|
| 0 | : OK |
| -13 | : No buffers left |
| -15 | : Invalid source ID |

## 5.9.3 cpiGetDefinedReplyBuffer

This function reserves a buffer for a reply message. It is used when the reply message cannot immediately be sent back to the base system.

int cpiGetDefinedReplyBuffer(cpiMsgHeader *HEADER, cpiMsgID *ID);

| | |
|---|---|
| HEADER | Header data of the message to which this message is a reply |
| ID | Id for a reply message |

Return values:

| | |
|---|---|
| 0 | : OK |
| -13 | : no buffers left |

## 5.9.4 cpiGetNotificationBuffer

This function reserves a buffer for a notification message. It is used to send system status messages to the base system.

int cpiGetNotificationBuffer(int CON, cpiMsgID *ID);

| | |
|---|---|
| CON | Destination connection |
| ID | Id for reply message |

Return values:

| | |
|---|---|
| 0 | : OK |
| -13 | : No buffers left |

# 5.10 Initializations

*Table 22: Initializations*

| Function | Description |
|---|---|
| cpiInitThisNode | Initialises data of this node. |
| cpiInitRemoteNode | Gives information of the SYS600 node to CPI. |
| cpiCreateConnection | Creates a connection and starts the communication between CPI and SYS600. |
| cpiCloseConnection | Removes the connection between CPI and SYS600. |

These functions initialize the TCP/IP connections between the SYS600 applications and the CPI application program. Figure 6 explains what kind of initialization is needed when connecting CPI to SYS600.

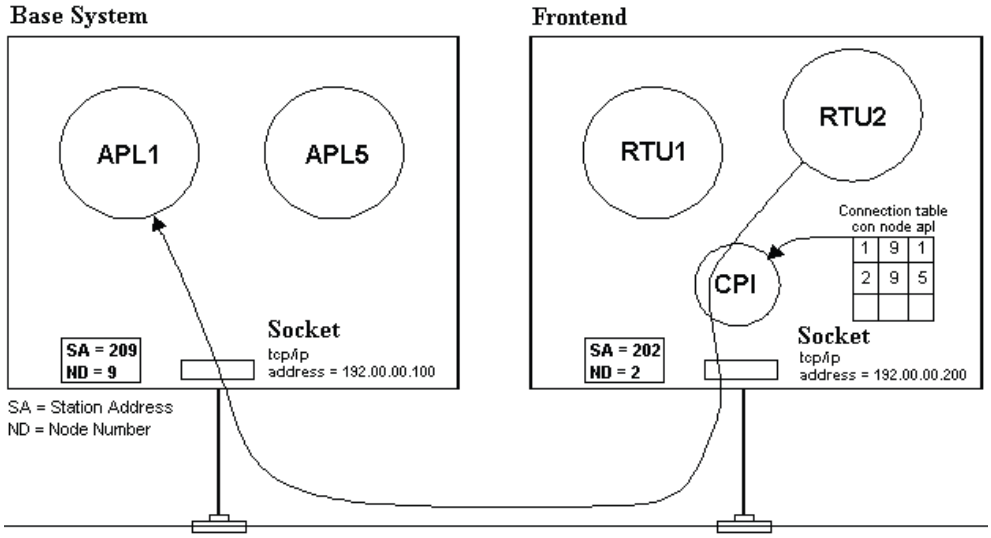



*Figure 6: Initialisation needed when connecting CPI to SYS600.*

Example of initialization needed for CPI:

```
basesystem.sin_family = AF_INET;
basesystem.sin_port = SPIDERSERVER;
strcpy(&basesystem.sin_addr,”192.00.00.100”);

cpiInitThisNode(202, 2);
cpiInitRemoteNode(basesystem, 209, 9);
cpiCreateConnection(9, 1, &con1);
cpiCreateConnection(9, 5, &con2);
```

CPI sends an analog value from RTU2 to application 1 by using the following function:

```
cpiSendAnalogValue(con1, 2, block, val, status, &ID);
```

## 5.10.1 cpiInitThisNode

This function initializes the data of this node.

int cpiInitThisNode(cpiNodeAddress SA, cpiNodeNumber NA);

| | |
|---|---|
| SA | Node address of this socket (1...254). |
| NA | Node number of this socket (1...250) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -26 | : Invalid data |

## 5.10.2 cpiInitRemoteNode

This function gives information of the SYS600 node to CPI.

int cpiInitRemoteNode(cpiTCPIPAddress TA, cpiNodeAddress NA, cpiNodeNumber NN);

| | |
|---|---|
| TA | SYS600 TCP/IP address |
| NA | Node address of SYS600 socket (1...254) |
| NN | Node number of SYS600 (1...250) |

Return values:

| | |
|---|---|
| 0 | : OK |
| -1 | : Connection error |
| -8 | : Invalid address |
| -14 | : Invalid node |
| -17 | : Invalid tcp address |

## 5.10.3 cpiCreateConnection

This function creates a connection and starts the communication between CPI and SYS600.

int cpiCreateConnection( cpiNodeNumber NN, int AN, int *CN,);

| | |
|---|---|
| NN | Node number of SYS600 |
| AN | Application number in SYS600, where this connection is created. |
| CN | Connection number. This return value is used as a destination number when sending messages to SYS600. |

Return values:

| | |
|---|---|
| 0 | : OK |
| -1 | : Connection error |
| -14 | : Invalid node number |

## 5.10.4 cpiCloseConnection

This function removes the connection between CPI and SYS600.

int cpiCloseConnection(int CN);

| | |
|---|---|
| CN | Connection number |

Return values:

| | |
|---|---|
| 0 | : OK |
| -1 | : Connection error, invalid connection number |

## 5.11 Bitset functions

The following functions are used to handle the bitset data type. The bitset data type is used in the cpiSelect function.

*Table 23: Bitset functions*

| Function | Description |
|---|---|
| cpiSET | Sets one bit in a bit set. |
| cpiCLR | Clears one bit from a bit set. |
| cpiISSET | Checks the value of one bit and returns it. |

### 5.11.1 cpiSET

This function sets one bit in the bit set.

int cpiSET(int BIT, cpiBitSet *BITSET)

| | |
|---|---|
| BIT | A number of bit which is set to 1, (0 = first bit) |
| BITSET | Bitmask to use with cpiSelect |

Return values:

| | |
|---|---|
| 0 | : If the tested bit does not belong to the bitset |
| 1 | : If the tested bit belongs to the bitset |

### 5.11.2 cpiCLR

This function clears one bit from the bit set.

int cpiCLR(int BIT, cpiBitSet *BITSET)

| | |
|---|---|
| BIT | A number of bit which is cleared to 0, (0 = first bit) |
| BITSET | Bitmask to use with cpiSelect |

Return values:

| | |
|---|---|
| 0 | : If the tested bit does not belong to the bitset |
| 1 | : If the tested bit belongs to the bitset |

### 5.11.3 cpiISSET

This function checks the value of one bit and then returns it.

int cpiISSET(int BIT, cpiBitSet *BITSET)

| | |
|---|---|
| BIT | A number of bit whose state is tested |
| BITSET | Tested bit mask |

Return values:

| | |
|---|---|
| 0 | : If the tested bit does not belong to the bitset |
| 1 | : If the tested bit belongs to the bitset |

# Section 6 Support for System Configuration Tool

This section provides instructions on how to provide support for the CPI communication protocol configuration in the System Configuration Tool.

When using the System Configuration Tool for configuring the CPI application program, the SYS600 base system revision needs to be 8.4.3 or newer.

**Definitions:**

- Configuration attribute: a value that is stored into the permanent protocol configuration file by the System Configuration Tool. This value is later utilised when the CPI application protocol is started in SYS600.
- Attribute definitions: the information about all the CPI configuration attributes. It includes names, data types, data structures, description texts, help texts and description texts for different attribute values of all CPI configuration attributes to be used by the System Configuration Tool.

## 6.1 Dependencies on other systems

The base system and communication system are configured during system start-up by using the functionality included in the System Configuration Tool. The CPI application program utilizes its own permanent protocol configuration file during start-up. For more information about base system configuration, see Section 3.

## 6.2 Functionality

The included CPI protocols are recognised from the SYS600 computer in System Configuration Tool based on following requirement:

Under \sc\prog folder, each sub-folder is assumed to be named according to the CPI application program, if

- it includes the permanent protocol configuration file named **Config.ini**.
- it includes the attribute definitions file named **Attr_Def.scl**.
- it includes the attribute mappings file named **Attr_Map.ini**.

The station type to be attached into the CPI protocol communication line is either RTU or SPA.

The CPI application program, i.e. the executable file, may either exist in the same computer where the System Configuration Tool is running or on an external computer. However, the permanent protocol configuration file, attribute definitions and mappings files need to exist in a sub-folder under the \sc\prog folder on the same computer where the System Configuration Tool is being used. If the CPI application program is running on an external computer, one of its drives can be mapped through LAN to the computer in which SYS600 is running. Through this kind of drive mapping, the files for configuring CPI application program are located only in the SYS600 computer. The protocol to be used for the CPI application program is concluded from the sub-folder name under the \sc\prog folder.

Figure 7 shows the MODBUS SLAVE communication protocol included in system configuration. The folder \sc\prog\modbus_slave exists in SYS600 computer, where the System Configuration Tool is located. The name of the protocol is displayed as MODBUS SLAVE in the object tree (the included underscore character is removed and the sub-folder name is uppercased).

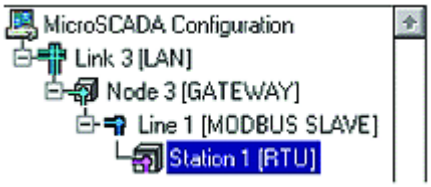



*Figure 7: The CPI protocol in the Object Tree of the System Configuration Tool*

The permanent protocol configuration file must be named **Config.ini**. This file contains the configuration parameters to be used by the CPI application program. The format of this file follows the Windows Initialisation File Format specification. Those parameters, which are shown and configured in the System Configuration Tool as a property of the CPI communication line, should be included under the section name Configuration.

The attribute definitions file must be named **Attr_Def.scl**. This file contains e.g. limit values and descriptions for the configuration parameters to be used by the System Configuration Tool. The format of the file follows the SCIL syntax.

## 6.3 Permanent protocol configuration file

The format of this file follows the Windows Initialisation File Format specification. All the configuration attributes, that are needed to be shown in the System Configuration Tool, need to exist as key names and values under the section name Configuration. Other section names may be used to store CPI application program related information, which is not needed to be shown in the System Configuration Tool.

```
[Configuration]
key_name_1=key value 1
key_name_2=key value 2
…
key_name_n=key name i
[Section_name_2]
key_name_1=key value 1
key_name_2=key value 2
…
key_name_n=key name n
…
[Section_name_m]
key_name_1=key value 1
key_name_2=key value 2
…
key_name_n=key name n
```

The key name is the same as the attribute name in the Attribute definitions file Attr_Def.scl.

### 6.3.1 Example of Config.ini

```
[Configuration]
own_node_number=4
own_station_number=204
base_system_node_number=9
base_system_station_number=209
tcp_ip_address=194.142.148.36
```

```
application_number=1
communication_port_name=COM1
baud_rate=9600
parity=0
stop_bit_count=0
data_bit_count=8
flow_control=0
```

## 6.4 Attribute definitions file

The format of each attribute definition follows the SCIL syntax. The following common definitions are valid for all data types:

```
<Attribute name> = LIST(-
                data_type=<attribute data type>,-
                modifiable=<attribute editable>,-
                descriptive_text=<name text for attribute>,-
                default_value=<default value for attribute>,-
                help_text=<help text for attribute>)
```

The name of the attribute is unique for all the CPI configuration attributes. All spaces must be replaced with the underscore character. For example, the attribute TCP/IP Address is presented as TCP_IP_ADDRESS as valid CPI attribute name for the System Configuration Tool.

Possible data types for the attributes are:

- Integer
- Real
- Text
- Time
- Boolean

If an attribute is modifiable, the value is TRUE. Otherwise the attribute value is FALSE. If the attribute is defined to be modifiable, its value can be edited in the edit area of the System Configuration Tool. If the attribute is non-modifiable, the editing of the attribute is disabled in the edit area of the System Configuration Tool. Figure 8 shows the Attribute Tree of the System Configuration Tool, where all the CPI attributes are included in the View named All Attributes and under a group name CPI Configuration Attributes.

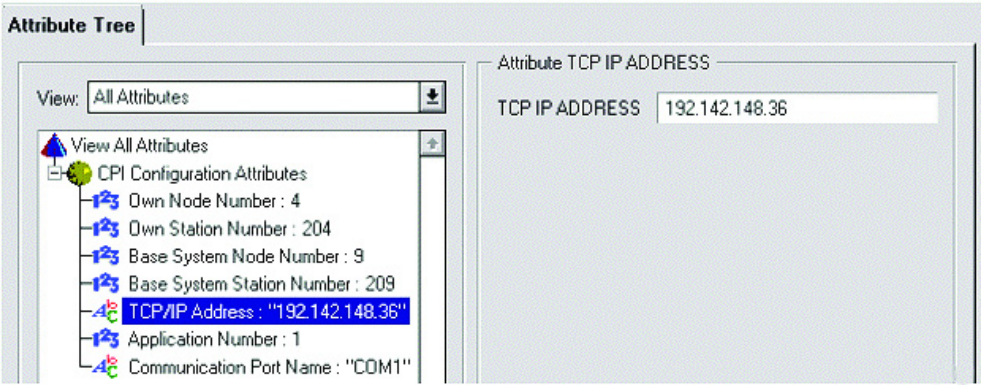



*Figure 8: The Attribute Tree and Edit area of the System Configuration Tool*

The Attribute Tree uses descriptive text. When a CPI attribute has been selected in the object tree, the Status bar (in Figure 9) displays the help text at the bottom of the System Configuration Tool.

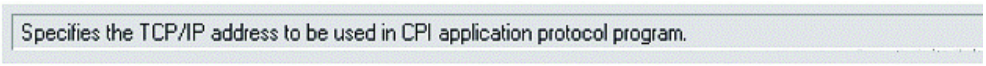



*Figure 9: The Status bar*

A default value is utilised for an attribute, when the CPI application protocol is added into the System Configuration Tool. It is possible for to change the attribute value according to its attribute definitions by using the edit area.

## 6.4.1 Definitions for different attribute data types

*Table 24: Integer*

| INTEGER | | |
|---|---|---|
| **Attribute definition** | **Description data type** | **Example** |
| Data_type | TEXT | ”INTEGER" |
| Min_value | INTEGER | 1 |
| Max_value | INTEGER | 7 |
| Modifiable | BOOLEAN | TRUE |
| Special_values | VECTOR of INTEGERs | (1, 3, 7)* |
| Value_descriptions | VECTOR of TEXTs | (”Not in Use”, ”Synhcronised”, ”Asynchronised”)* |
| Descriptive_text | TEXT | "Example Integer Attribute" |
| Default_value | INTEGER | 3 |
| Help_text | TEXT | "This is an example of an integer attribute." |

*Table 25: Real*

| REAL | | |
|---|---|---|
| **Attribute definition** | **Description data type** | **Example** |
| Data_type | TEXT | ”REAL” |
| Min_value | REAL | 1.0 |
| Max_value | REAL | 1.5 |
| Modifiable | BOOLEAN | TRUE |
| Descriptive_text | TEXT | “Example Real Attribute” |
| Default_value | REAL | 1.1 |
| Help_text | TEXT | “This is an example of a real attribute.” |

*Table 26: Text*

| TEXT | | |
|---|---|---|
| **Attribute definition** | **Description data type** | **Example** |
| Data_type | TEXT | ”TEXT” |
| Min_length | INTEGER | 4 |
| Max_length | INTEGER | 5 |
| Modifiable | BOOLEAN | TRUE |
| Table continues on next page | | |

| | | |
|---|---|---|
| Descriptive_text | TEXT | “Example Text Attribute” |
| Default_value | TEXT | ”COM1” |
| Help_text | TEXT | “This is an example of a text attribute.” |

*Table 27: Time*

| TIME | | |
|---|---|---|
| **Attribute definition** | **Description data type** | **Example** |
| Data_type | TEXT | ”TIME” |
| Min_value | TIME | Pack_time(1978,1,1,0,0,0)** |
| Max_value | TIME | Pack_time(2045,12,31,23,59,59)** |
| Modifiable | BOOLEAN | TRUE |
| Descriptive_text | TEXT | “Example Time Attribute” |
| Default_value | TIME | Pack_time(1999,12,20,12,0,0)** |
| Help_text | TEXT | “This is an example of a time attribute.” |

*Table 28: Boolean*

| BOOLEAN | | |
|---|---|---|
| **Attribute definition** | **Description data type** | **Example** |
| Data_type | TEXT | ”BOOLEAN” |
| Modifiable | BOOLEAN | TRUE |
| Descriptive_text | TEXT | “Example Boolean Attribute” |
| Default_value | BOOLEAN | TRUE |
| Help_text | TEXT | “This is an example of a boolean attribute.” |

* The VECTOR data type can be recognised from the parentheses around the enumerated INTEGER or TEXT values. If only one item is included in the VECTOR, there is a need to specifically use the VECTOR statement around the attribute value, VECTOR(1)

** The PACK_STR is a SCIL function, which can be used to convert the year/month/day and hour/minute/seconds representation to the TIME data type format of SCIL. E.g. PACK_STR(YEAR, MONTH, DAY,HOUR,MINUTE,SECOND).

## 6.4.2 Example of Attr_Def.scl

```
#create GL_ATTRIBUTES:V = LIST(-
OWN_NODE_NUMBER = LIST(-
                data_type=”INTEGER”,-
                min_value=1,-
                max_value=99,-
                modifiable=TRUE,-
                descriptive_text=”Own Node Number”,-
                default_value=4,-
                help_text=”Specifies the own node number to be
                used in CPI application program.”),-

OWN_STATION_NUMBER = LIST(-
                data_type=”INTEGER”,-
                min_value=1,-
                max_value=255,-
                modifiable=TRUE,-
```

```
                descriptive_text=”Own Station Number”,-
                default_value=204,-
                help_text=”Specifies the own station number to
                be used in CPI application program.”),-

BASE_SYSTEM_NODE_NUMBER = LIST(-
                data_type=”INTEGER”,-
                min_value=1,-
                max_value=99,-
                modifiable=TRUE,-
                descriptive_text=”Base System Node Number”,-
                default_value=9,-
                help_text=”Specifies the base system node number
                to be used in CPI application program.”),-

BASE_SYSTEM_STATION_NUMBER = LIST(-
                data_type=”INTEGER”,-
                min_value=1,-
                max_value=255,-
                modifiable=TRUE,-
                descriptive_text=”Base System Station Number”,-
                default_value=209,-
                help_text=”Specifies the base system station
                number to be used in CPI application program.”),-

TCP_IP_ADDRESS = LIST(-
                data_type=”TEXT”,-
                min_length=7,-
                max_length=15,-
                modifiable=TRUE,-
                descriptive_text=”TCP/IP Address”,-
                default_value=”192.142.148.36”,-
                help_text=”Specifies the TCP/IP address to be
                used in CPI application program.”),-

APPLICATION_NUMBER = LIST(-
                data_type=”INTEGER”,-
                min_value=1,-
                max_value=99,-
                modifiable=TRUE,-
                descriptive_text=”Application Number”,-
                default_value=1,-
                help_text=”Specifies the application number,
                where CPI application program is running.”),-

COMMUNICATION_PORT_NAME = LIST(-
                data_type=”TEXT”,-
                min_length=4,-
                max_length=5,-
                modifiable=TRUE,-
                descriptive_text=”Communication Port Name”,-
                default_value=”COM1”,-
                help_text=”Specifies the communication port
                name, which CPI application program is
                utilising.”))

                #return %GL_ATTRIBUTES
```

The format of this file needs to follow the SCIL syntax. There is always a risk that due to a typing error the Attr_Def.scl file is not applicable to the SCIL syntax. In the SCIL Editor, it is possible to define the Syntax Check mechanism into use automatically during the saving operation by selecting Options – Check Syntax at Save. Therefore, the editing this file should be done by using the SCIL Editor.

# 6.5 Attribute mappings file

The section name refers to the SYS600 object where the attribute name belongs.

| | | |
|---|---|---|
| [NODE] | Base System Node Object | (NOD:B) |
| [NODE_LINK] | Communication Line Object | (NET:S) |
| [STATION] | Communication Station Object | (STA:S) |

The key name in this file is the same as the attribute name in the attribute definitions file, as well as the key name in the permanent protocol configuration file. The key value in this file represents the two-char attribute name to be used for defining the property to the SYS600 object.

For example, if a key name own_station_number with a key value SA exists inside the NODE section, it means that the configuration attribute is referenced to the NOD'n':BSA attribute in SYS600, where 'n' equals the node number object.

## 6.5.1 Example Attr_Map.ini

```
[NODE]
own_station_number=SA
[NODE_LINK]
communication_port_name=SD
baud_rate=BR
stop_bit_count=SB
[STATION]
in_use=IU
```

# Section 7 Terminology

| Term | Description |
|---|---|
| Application Communication Protocol (ACP) | An internal protocol of SYS600 that is generally used in communication between the SYS600 nodes. |
| ACP Message | Communication between the SYS600 nodes is implemented by using ACP. It supports three types of message dialogs: write commands, read commands and notifications. |
| Attribute | Individual data items that form a part of an object are called attributes. Each object has a set of attributes that store information and describe the qualities of the object. Attributes contain, for example measured values, texts, program lines, time stamps etc. depending on the object type. |
| Base System | The Base System offers services to the applications. It provides database structures, database handling mechanisms and file handling functionality. The Base System also offers an application programming interface for attaching functions as separate programs. |
| BSD Socket | Socket interface defined in the Unix version developed by Berkeley Software. |
| Communication Programming Interface (CPI) and CPI Library | Protocol environment software that is used for externally implemented protocol converters. The CPI Library is a collection of functions for sending and receiving message to or from the SYS600 base system, as well as functions for packing and unpacking data of messages. |
| CPI Application Program | In some cases called a Gateway program.<br>An application program made by using the CPI interface exchanges data between SYS600 and a foreign system. It emulates the NET containing stations of the RTU, SPA or SPI device type. |
| Communication System | Handles the data transmission in the SYS600 system as well as between the devices. The communication can be divided into upper level communication and process communication. |
| Local Area Network | Used for upper level communication between the base system and workplaces. It has a large capacity for data transmission. |
| Message Buffer | A place in the memory to store the ACP messages to or from the SYS600 base system. The CPI Buffer Management functions reserve buffers for different message types. |
| NET | SYS600 communication unit. |
| NET Emulation | A CPI application program that resembles the behavior of the NET unit./ |
| Node | Communication object in SYS600, for example, the SYS600 base system or a CPI application program. |
| Node Number | A unique identifier of a node. |
| Open System Connection (OSI) | A model for network architecture standardised by ISO. |
| Protocol | A protocol is a set of rules and conventions for transmitting information in the network. They govern the content, format, timing, sequencing and error control of messages. |
| RTU | Remote Terminal Unit |
| SCIL | A high level language especially designed for composing customized, process specific supervisory control software. Data transmission from the base system to the CPI application program is done by setting the attributes of the communication system objects or process objects. |
| SPA Device | A SPACOM module connected to SYS600 is seen as a SPA device. |

Table continues on next page

| | |
|---|---|
| System Configuration Tool | Manages the configuration of the base system and the NET communication unit. |
| TCP/IP | Transmission Control Protocol/Internet Protocol, used for data transmission in the LAN network to provide communication with several protocols across the connected networks of computers. |
| Thread | An independent part of the program that is active until it stops itself or it is stopped by the main program. Threads can be used to isolate tasks. Usually, a foreign protocol is implemented as a thread. |

# Section 8 Abbreviations

| Abbreviation | Description |
|---|---|
| ACP | Application Communication Protocol |
| Communication Programming Interface (CPI) and CPI Library | Communication Programming Interface, a protocol environment software that is used for externally implemented protocol converters. The CPI Library is a collection of functions for sending and receiving message to or from the SYS600 base system, as well as functions for packing and unpacking data of messages. |
| ISO | International Standards Organization |
| LAN | Local Area Network |
| OSI | Open System Interconnection |
| RTU | Remote Terminal Unit |
| SCIL | Supervisory Control Implementation Language |
| TCP/IP | Transmission Control Protocol/Internet Protocol |

# Index

**Hitachi ABB Power Grids**
**Grid Automation Products**
PL 688
65101 Vaasa, Finland

https://hitachiabb-powergrids.com/microscadax

Scan this QR code to visit our website



1MRK 511 469-UEN