Using a CTI 2500P-ACP1 Application Co-Processor module for RS232 serial data acquisition from an Ohaus IP Series High-Capacity Precision Top-loader bench scale

**Scope:**

This application is an update to a previous Tech Note on using the CTI 2572-TCM2 Serial Interface Adapter for this application. The 2500P-ACP1 is a more powerful and flexible platform which makes this application simpler to develop, test and deploy. NOTE: This application can be run on a 2500P-JACP Application Co-Processor module as well, with a change in the configuration of the PLC interface.
2500P-ACP1 Description:

The 2500P-ACP1 module is a general-purpose auxiliary controller that enhances the capabilities of all CTI 2500 Series® and SIMATIC® 505 PLC systems. This Advanced Function Module includes high-speed processing and multi-protocol communications support to provide existing systems with a significant increase in performance, features, and functionality. The 2500P-ACP1 runs as a PLC coprocessor performing complex logic/math functions, data logging, and communications with external devices. Although the 2500P-ACP1 can operate as a standalone controller, the application generally requires data transfer between a host PLC and the module.

Two different data transfer options are provided:

**PLC I/O (2500P-ACP1 only):**

The 2500P-ACP1 emulates a standard I/O module configured as 32WX / 32WY and/or 32X / 32Y image register data points. This allows the module to work with SIMATIC® 545/555 CPUs in limited applications where a maximum of 32 words of data is transferred to/from the CPU each PLC scan. ***(This is the transfer mode chosen for this application)***

**Data Cache (2500P-ACP1 and 2500P-JACP):**

Proprietary link offering enhanced data throughput to CTI 2500 Series® controllers via a dedicated Ethernet connection. Supports up to 4096 variables mapped to any PLC memory type (including Loop/Alarm variables). The 2500P-ACP1 includes two external 10/100Mb Ethernet ports with automatic detection of network speed, duplex mode, and cable wiring.

**Block Transfer (2500P-JACP only):**

The Block Transfer driver provides a method to transfer large blocks of data between a Janus Application Coprocessor (JACP) module and a SIMATIC® 545/555 PLC or CTI 2500 Series PLC via the I/O backplane. This driver provides significantly greater communications capabilities compared to “PLC I/O” above. This driver supports up to 4096 variables mapped to any PLC memory type (including Loop/Alarm variables).

**Serial Port:**

A serial port (male DB-9) provides an electrical interface for RS-232-C (subset) and RS-422-A connections. All port parameters are set by software configuration. Sending and receiving of messages is controlled by program logic.

**NOTE:**

*In the Appendix at the end of this Tech Note you will find additional details on the ACP1, as well as descriptions of all of the software instructions used and other pertinent information.*
ACP1 solution scope:
The following describes the means by which the 2500P-ACP1 will read the weight value from the scale.

SEND data by simulating the scale output:
This application uses an ACP1 to simulate the known ASCII data stream from the Ohaus scale. This ACP1 module performing the SEND operation will be referred to as ACP1
This WorkBench application is ‘ACP1_Serial_2’.
The ACP1 module is installed in a base with a CTI 2500-C400 PLC

RECEIVE data:
A 2nd ACP1 module receives and decodes the ASCII data stream to read the weight value from the scale and write it to a PLC memory location.

This ACP1 performing the RECEIVE operation will be referred to as ACP2
This WorkBench application is ‘ACP2_Serial’
The ACP2 module is installed in a base with a Simatic 555-1106 PLC

This program uses ‘ACP1 I/O Interface’ to send the following to the PLC:

Value of ‘Decode_done’ bit sent to PLC X2001; this bit is an error indication.
   In the PLC RLL this input is used to stroke a counter to track decoding errors.
   This bit could also be used to signal an HMI to a communications problem.

Value of ‘Scale_wt_intX10’ word sent to PLC WX2065.
   In the PLC SFPGM1 the value in WX2065 is multiplied times 10 and put into V1000. (Real).
   This is the actual scale weight as a floating point number.

Communications wiring:
The field wiring part of this ACP1 module communication is achieved by plugging a standard Null Modem cable into the DB9 serial port on each module.

Ohaus scale configuration:
- Continuous print mode
- 9600 baud
- No parity.
Sending / Encoding and Receiving / Decoding the ASCII data stream

Using an ACP1 to simulate the serial data from the bench scale:

*Number value to be transmitted = -481.2*

**Sent ASCII string from ACP1:**

```
'-$L__481.2_g_____$R$L'
```

*Where:*

1st character = this field represents the polarity of the number value transmitted.

- This field is a Minus Sign `-' indicating a negative number.
- If this field were a positive number this 1st character would have been an Underscore `_'

that ASCII string would then be `'_$L__481.2_g_____$R$L'`

2nd character = `'$L'` this is a ‘Line Feed’.

3rd, 4th, and 5th characters = `_' which are Underscores.

6th, 7th, 8th, 9th and 10th characters = `481.2` is the actual number value transmitted in the format ‘xxx.x’.

11th and 12th characters = `_'g` which are an Underscore and ‘g’ (for grams).

13th – 17th characters = `_' are Underscores.

18th character = `$R$` this is a Carriage Return.

19th character = `$L$` this is a Line Feed.

Using an ACP1 to receive and decode the serial data from the bench scale:

**Received ASCII string at ACP2:**

```
'-$L__481.2_g_____$R$L'
```

```
'$L__481.2_g_____$R$L'
```

**Decoded number value:**

```
rcv_real -481.2
```
ACP1 = SEND

This 1st ACP is being used to send data out the serial port to be received by the 2nd ACP. The screen captures below show the programming requirements to configure the serial port and to send the data out the serial port. The data sent matches the format explained on Page 4. For testing, the only part that you should modify is the actual scale weight embedded in this message.

Configure Port:

*Note: these communications parameters match the specifications for the Ohaus bench scale.*

```plaintext
1 (* MyPort is an instance of 'SerIO' function block *)
2 (* SerIO manages communication through the serial port using user-defined strings *)
3
4 MyPort(true, needtosend FALSE, 'PT=1 BD=9600 DB=7 SB=1 FY=N FC=C IF=RS232', sendstring '=S1__481.2_g_____SRSL');
5 needtosend FALSE := false; (* turn off the sending if it was on from previous scan *)
```

Send Data:

*Note: this data send operation is unsolicited and is cyclic based on the timer function.*

```plaintext
13 (* Send Data *)
14 else
15  if TRUE MyPort.Open TRUE and MyPort.rcv FALSE = FALSE then
16   SendTMR(in := bSend_Start TRUE, pt:= T#1s); (* configure timer *)
17   bSend_Start TRUE := true; (* re-start the timer *)
18   if TRUE (SendTMR.Q FALSE) then
19     if TRUE bSend_Start FALSE := false; (* this must go low to reset TOW() for next use *)
20       needtosend FALSE := true;
21         (*sendstring := any_to_string(senddata);*)
22       end_if;
23     end_if;
24 end_if;
```
ACP2 = RECEIVE

This 2nd ACP is being used to receive data coming into the serial port from the 1st ACP. The screen captures below show the programming requirements to configure the serial port and to receive the data from the serial port. The data received matches the format explained on Page 4. In the following pages this data will be decoded to extract the scale weight itself.

Configure Port:

*Note: these communications parameters match the specifications for the Ohaus bench scale.*

```plaintext
(* MyPort is an instance of 'SerIO' function block *)
(* SerIO manages communication through the serial port using user-defined strings *)

MyPort(true,needtosend FALSE,'PT=1 BD=9600 DB=7 SB=1 PY=N FC=N IF=RS232',sendstring '');

needtosend FALSE := false; (* turn off the sending if it was on from previous scan *)
```

Receive Data:

*Note: this ASCII data is tested to determine if the string length is correct.*

```plaintext
(* Receive Data *)
19 if FALSE MyPort.Open TRUE and MyPort.rcv FALSE then
20 rcvstring := MyPort.DataRcv 
21 rcv_other := rcvstring 
22 (* determine received string length *)
24 rcv_string_length 19 := MLEN (rcvstring ''); (* get string length - should be 19 characters *)
26 if TRUE rcv_string_length 19 = 19 then (* check for correct string length, then decode *)
```
Number value received = -481.2

**ACP2 = RECEIVE**

This 2nd ACP is being used to receive data coming into the serial port from the 1st ACP. This screen captures below show the programming requirements to configure the serial port and to received the data from the serial port. The data received matches the format explained on Page 4. In the following pages this data will be decoded to extract the scale weight itself.

**Analyze and decode the ASCII string if it is a negative number**

*Note: this data packet has two leading characters and two trailing characters all of which aid in determining that this packet is complete and also the polarity of the number value.*

```cpp
/* Decode string pattern for a NEGATIVE value */
if ( (ASCII(rcv_other ' ', 1)) = 45 \ (check if 1st character is '-' ASCII 45) )
    (ASCII(rcv_other ' ', 2)) = 10 \ (check if 2nd character is 'LF' ASCII 10 )
    (ASCII(rcv_other ' ', 18)) = 13 \ (check if 18th character is 'CR' ASCII 13 )
    (ASCII(rcv_other ' ', 19)) = 10 then \ (check if 19th character is 'LF' ASCII 10 )

lead_char_NEG_TRUE := true; \ (* set NEG data good flag *)
/*extract 5 characters starting at 6th position*/
rcv_str '481.2' := (MID(rcv_other ' ', 5,6));
rcv_number '481.2' := Any_to_Real (rcv_str '481.2'); \ (* convert to a number *)
rcv_real '481.2' := -(rcv_number '481.2'); \ (* negate the number *)
// rcvstring := ''; \ (* Clear input buffer *)
rcvstring ' ' := DELETE (rcvstring ' ', 19, 1);
rcv_other ' ' := DELETE (rcv_other ' ', 19, 1);
else
    lead_char_NEG_TRUE := false; \ (* reset NEG data good flag *)
rcv_error NEG ' SL 678.9 g $RSL' := rcvstring ' ';
rcv_length_ERR_NEG 19 := rcv_string_length 19 ;
end_if;
```

**Analyze and decode the ASCII string if it is a positive number**

*Note: this data packet has two leading characters and two trailing characters all of which aid in determining that this packet is complete and also the polarity of the number value.*

```cpp
/* Decode string pattern for a POSITIVE value */
if ( (ASCII(rcv_other ' ', 1)) = 45 \ (check if 1st character is ' ' ASCII 45) )
    (ASCII(rcv_other ' ', 2)) = 10 \ (check if 2nd character is 'LF' ASCII 10 )
    (ASCII(rcv_other ' ', 18)) = 13 \ (check if 18th character is 'CR' ASCII 13 )
    (ASCII(rcv_other ' ', 19)) = 10 then \ (check if 19th character is 'LF' ASCII 10 )

lead_char_POS_FALSE := true; \ (* set POS data good flag *)
/*extract 5 characters starting at 6th position*/
rcv_str '481.2' := (MID(rcv_other ' ', 5,6));
rcv_number '481.2' := Any_to_Real (rcv_str '481.2'); \ (* convert to a number *)
rcv_real '-481.2' := rcv_number '481.2'; \ (* number is positive *)
// rcvstring := ''; \ (* Clear input buffer *)
rcvstring ' ' := DELETE (rcvstring ' ', 19, 1);
rcv_other ' ' := DELETE (rcv_other ' ', 19, 1);
else
    lead_char_POS_FALSE := false; \ (* reset POS data good flag *)
rcv_error POS ' ' := rcvstring ' ';
rcv_length_ERR_POS 19 := rcv_string_length 19 ;
end_if;
```
Determine if there are errors in the ASCII string

`
68 (* check for valid character string based on leading character *)
69 if (lead_char_NEG TRUE = 0) AND (lead_char_POS FALSE = 0) then
70   Data_Error FALSE := true; (* set flag to signal error *)
71   rcv_real -481.2 := 0; (* zero out invalid data *)
72 else
73   Data_Error FALSE := false; (* reset error flag *)
74 end_if;
```

Send scale weight value to PLC memory

`
75 (* send scale weight data to PLC *)
76 Scale_wt_real -4812 := rcv_real -481.2 * 10;
77 Scale_wt_Intx10 -4812 := any_to_int(Scale_wt_real -4812);
```

Indicate if string length is over running

`
94 (* check if received string length is invalid *)
95 if rcv_string_length 19 > 254 then
96   Decode_done Flag := true; (* flag to clear input buffer *)
97 end_if;
```

---

ROCK SOLID PERFORMANCE. TIMELESS COMPATIBILITY.
Data received at PLC

In this Ladder Logic screen shot you can see where a Special Function Program is being called which is where the weight value is being handled.

Next you can see that X2001 is being used to count (or track) any decoding errors of the ASCII message.
**Data received at PLC**

In the Special Function Program you can see where the ‘Integer x 10’ value from the ACP is being converted into a Real number.

In this Data Window you can see the values of the Scale Weight -
WX2065 shows up as ‘-4812’ which is in the format of ‘Integer x 10’
V1000. shows up as ‘-481.2’ which is in the format of ‘Real’ or ‘Floating Point’
X2001 which is the Boolean ‘Decode_done’ error bit
TCC1 which is the Counter tracking decode errors in ACP2
The ACP I/O interface is how the module reads & writes PLC memory. In this example the I/O interface will be configured for 32 discrete points in, 32 discrete points out, 32 words in, & 32 words out.

2500P-ACP1 I/O Definition
The ACP1 I/O definition is specified by selecting one the following I/O configurations in the CTI 2500P-ACP1 I/O Configuration Wizard provided in CTI Workbench:

- Discrete I/O: 32 inputs / 32 outputs (32X/32Y)
- Word I/O: 32 inputs / 32 outputs (32WX/32WY)
- Mixed I/O: 32 discrete inputs/outputs and 32 word inputs/outputs (32X/32Y/32WX/32WY)

The Mixed I/O interface requires special care when assigning an I/O Address because the Series 505® model allows one “login” address for each module slot. Therefore, the I/O Address assigns the image register positions for both the Discrete I/O and Word I/O values.

In the example, a value of “2001” is designated as the I/O Address. This equates to the following I/O mapping for ACP1 data within the PLC:

- 32 discrete inputs mapped to X2001-X2032
- 32 discrete outputs mapped to Y2033-Y2064
- 32 word inputs mapped to WX2065-WX2096
- 32 word outputs mapped to WY2097-WY2124
2500P-ACP1 I/O Configuration

From within WorkBench you open this configuration wizard and define:
- Module Log-in Configuration—in this case it is 32X / 32Y / 32WX / 32WY
- Module Log-In Address—this is the beginning I/O address of this configuration

Lastly, you populate the rows in each I/O column with the WorkBench symbol that you want to connect to PLC memory
- In this example -
  - Boolean symbol ‘Decode_done’ has been assigned to X2001
  - Integer symbol ‘Scale_wt_Intx10’ has been assigned to WX2065
2.1 Status Indicator LEDs

At the top of the module front panel are three status LEDs. The function of the LEDs is described in the following table.

<table>
<thead>
<tr>
<th>LED</th>
<th>State</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS</td>
<td>Off</td>
<td>Module not operational</td>
</tr>
</tbody>
</table>
|       | Flashing | Module not ready – Operator action required  
|       |       | (Module error, Watchdog timeout, Application program not found, or Host interface failure) |
|       | On    | Module operation is normal                                                  |
| ACTIVE| Off   | Application program stopped                                                |
|       | Flashing | Program loaded but logic is not running (PAUSED state).  
|       |       | I/O interface and communication protocols are active.                     |
|       | On    | Application program is executing (RUN state)                               |
| USER  | Off   |                                                                             |
|       | Flashing | Controlled by application logic                                          |
|       | On    |                                                                             |

2.2 LED Multi-Segment Display

The Multi-Segment Display (MSD) is located below the status LEDs. The MSD is used to display status and error codes. During normal operation the MSD displays the TCP/IP address of the product, one octet at a time. When an error is encountered, the MSD will also display an Error Code. See APPENDIX A: ERROR CODES for a list of error codes and descriptions.

2.3 Reset Button

The Reset Button allows you to initiate a “soft reset” for the 2500P-ACP1 module. This reset is equivalent to cycling power to the module. When the button is depressed (using a pointed object such as a ball point pen, the module is restarted after an orderly shutdown of the application program. This reset action can be disabled by setting module switch (SW3) to CLOSED position. Section 3.1.2 contains information on location and setting for module switches.
2.4 Ethernet Status Indicators

The Ethernet LEDs indicate the state of the TCP/IP interface and whether the module is transmitting and receiving data via the Ethernet as shown in the following table.

<table>
<thead>
<tr>
<th>LED</th>
<th>State</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS (Network Status)</td>
<td>Off</td>
<td>TCP/IP is not operational.</td>
</tr>
<tr>
<td></td>
<td>On-Red</td>
<td>TCP/IP is operational. A device with the same IP address as this 2500P-ACP1 module has been detected.</td>
</tr>
<tr>
<td></td>
<td>On-Green</td>
<td>TCP/IP is connected and operational.</td>
</tr>
<tr>
<td>XMT (Transmit)</td>
<td>Flashing</td>
<td>Ethernet port is transmitting data.</td>
</tr>
<tr>
<td>RCV (Receive)</td>
<td>Flashing</td>
<td>Ethernet port is receiving data.</td>
</tr>
</tbody>
</table>

2.5 Ethernet Ports

The 2500P-ACP1 provides two Ethernet ports capable of operating at 10/100Mb, half or full duplex. The speed and duplex mode are automatically negotiated with the device connected to the port. Each port supports auto-crossover capability, allowing the port to be connected to an external Ethernet switch or directly to a device, such as a laptop or 2500 Series® controller. Both ports are functionally equivalent.

The 2500P-ACP1 incorporates an Ethernet switch which is connected to both Ethernet ports and the microprocessor. The switch allows either port to communicate with the microprocessor. The two ports can be connected or isolated from each other (see ‘Port isolation’ below). Besides providing this connectivity, the switch also provides hardware protection against network broadcast/multicast storms.

It is also possible to enable ‘IP aliasing’ by configuring an Alias IP Address and Alias Subnet Mask. This allows two IP addresses to be associated with the ACP1 module so that each Ethernet port can be connected to a separate sub-network. When ‘IP aliasing’ is enabled, either Ethernet port can be used with either sub-network (i.e. the IP Address and Subnet Mask is not port specific). When using this feature, Port isolation should always be enabled.

Port Isolation can be enabled by setting module switch (SW4) to CLOSED position. This setting blocks forwarding of all Ethernet packets between the two ports and allows the ACP1 to be used with redundant network topologies without creating network loops. Section 3.1.2 contains information on location and setting for module switches.
Each Ethernet port connector contains two embedded LEDs. The **Link LED** indicates whether the Ethernet port is successfully connected to another Ethernet device, such as a network switch. The **Activity (ACT)** LED provides visual indication that Ethernet packets are being received or transmitted via the port. See the following table below for more information.

<table>
<thead>
<tr>
<th>LED</th>
<th>State</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link</td>
<td>Off</td>
<td>Ethernet link is not available.</td>
</tr>
<tr>
<td></td>
<td>On</td>
<td>Ethernet link is available.</td>
</tr>
<tr>
<td>Act (Activity)</td>
<td>Off</td>
<td>No Ethernet frames are being transmitted on the network to which the port is connected.</td>
</tr>
<tr>
<td></td>
<td>Flashing</td>
<td>Ethernet frames are being transmitted on the network to which the port is connected</td>
</tr>
</tbody>
</table>

### 2.7 Serial Port

The male DB9 connector on the front panel provides the serial port interface. Modbus-RTU (Master or Slave) and General ASCII Send/Receive data protocols are supported for the serial port and managed by the application program. All port parameters, including the selected electrical interface (RS-232 or RS-422), are set by software configuration via **CTI Workbench**. The cable used with the external device must connect to the pins used by the selected electrical interface.

#### RS-232 (Subset) Pinout:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>RXD</td>
<td>Receive Data</td>
</tr>
<tr>
<td>3</td>
<td>TXD</td>
<td>Transmit Data</td>
</tr>
<tr>
<td>5</td>
<td>GND</td>
<td>Signal Ground</td>
</tr>
<tr>
<td>7</td>
<td>RTS</td>
<td>Request to Send (optional)</td>
</tr>
<tr>
<td>8</td>
<td>CTS</td>
<td>Clear to Send (optional)</td>
</tr>
</tbody>
</table>

#### RS-422 Pinout:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+RX</td>
<td>Receive Data (+)</td>
</tr>
<tr>
<td>4</td>
<td>+TX</td>
<td>Transmit Data (+)</td>
</tr>
<tr>
<td>6</td>
<td>-RX</td>
<td>Receive Data (-)</td>
</tr>
<tr>
<td>9</td>
<td>-TX</td>
<td>Transmit Data (-)</td>
</tr>
</tbody>
</table>

**NOTE:**

*A serial port connection to CTI Workbench is not supported by the ACP1 module. This interface must be made using TCP/IP connection.*
Null Modem cable pinout:

Serial Port Communications

CTI products such as the 2500P-ACP1 and 2500P-JACP provide a serial port which can be used to communicate with devices support electrical interfaces such as RS-232, RS-422, and RS-485. There are two requirements for communication with the device:

1. The port parameters must be configured to match the requirements of the target device.
2. A communications protocol that is supported by the device must be used.

Configuring the Serial Port Parameters.

The serial port is configured by constructing an ASCII string containing parameter descriptors and associated values as shown in the following table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Descriptor</th>
<th>Valid Values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port ID</td>
<td>PT</td>
<td>Product Dependent</td>
<td>1</td>
</tr>
<tr>
<td>Baud Rate</td>
<td>BD</td>
<td>1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200</td>
<td>9600</td>
</tr>
<tr>
<td>Data Bits</td>
<td>DB</td>
<td>7, 8</td>
<td>8</td>
</tr>
<tr>
<td>Stop Bits</td>
<td>SB</td>
<td>1, 2</td>
<td>1</td>
</tr>
<tr>
<td>Parity</td>
<td>PT</td>
<td>None (N), Even (E), Odd (O)</td>
<td>ASCII (N) / Modbus RTU (E)</td>
</tr>
<tr>
<td>Flow Control</td>
<td>FC</td>
<td>No (N), Yes (Y)</td>
<td>N</td>
</tr>
<tr>
<td>Interface</td>
<td>IF</td>
<td>RS232, RS422 (ACP1 and JACP modules), RS-485 (JACP Module only)</td>
<td>RS232</td>
</tr>
</tbody>
</table>

Usage Rules

- If any parameters are missing or assigned invalid values, the default value for the parameter(s) will be used.
- All characters in the string are case insensitive.
- The string is not order dependent.
- Any extraneous content included in the string will be ignored.

Example String

```
PT=1 BD=19200 DB=8 SB=1 PY=N FC=N IF=RS232
```

Choosing a Communications Protocol

The following communications protocols are available:
- Modbus RTU Master Fieldbus configuration for “Com port.”
- Modbus RTU Slave Function Blocks: MBSLAVERTU, MBSLAVERTUX
- General ASCII: SERIO and SERIO_R to send and receive character string or byte array.

CTI Product Support

CTI 2500P-ACP1
CTI JACP Module
ACP1 Functions and Instructions used in this application

SERIO

Function Block - Manages communications through the serial port

Inputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN : BOOL</td>
<td>Enables communications</td>
</tr>
<tr>
<td>SND : BOOL</td>
<td>TRUE Sends data</td>
</tr>
<tr>
<td>CONF : STRING</td>
<td>Contains Serial Port Parameters</td>
</tr>
<tr>
<td>DATA/SND : STRING</td>
<td>Contains the data to be sent</td>
</tr>
</tbody>
</table>

Outputs

<table>
<thead>
<tr>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPEN : BOOL</td>
<td>TRUE if the communication port is open</td>
</tr>
<tr>
<td>RCV : BOOL</td>
<td>TRUE if data has been received</td>
</tr>
<tr>
<td>ERR : BOOL</td>
<td>TRUE if error detected during sending data</td>
</tr>
<tr>
<td>DATARCV : STRING</td>
<td>Contains received data</td>
</tr>
</tbody>
</table>

Remarks

The RUN input does not include an edge detection. The block tries to open the port on each call when RUN is TRUE (if port is not already open).

When RUN is FALSE the port will be closed (if open).

The CONF input is used for settings when opening the port. See Serial Port Parameters.

The SND input does not include an edge detection. Characters are sent on each call if SND is TRUE and DATA/SND is not empty.

The DATARCV string is erased and replaced with any received data each cycle.

Your application is responsible for storing received character immediately after each call to SERIO block.

The SERIO function block can be used in PC simulation mode.

In that case, the CONF input defines the communication port according to the syntax of the MODE command.

For example:

```
'CON1:9600,3,8,1'
```

ST Language

MySer is a declared instance of SERIO function block.

```
MySer (RUN, SND, CONF, DATA/SND);
OPEN := MySer.OPEN;
RCV := MySer.RCV;
ERR := MySer.ERR;
DATARCV := MySer.DATARCV;
```

String Operations

Below are the standard operators and functions that manage character strings:

<table>
<thead>
<tr>
<th>Code</th>
<th>Operator / Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>concatenation of strings</td>
</tr>
<tr>
<td>CONCAT</td>
<td>concatenation of strings</td>
</tr>
<tr>
<td>LEN</td>
<td>get string length</td>
</tr>
<tr>
<td>DELETE</td>
<td>delete characters in a string</td>
</tr>
<tr>
<td>INSERT</td>
<td>insert characters in a string</td>
</tr>
<tr>
<td>FIND</td>
<td>find characters in a string</td>
</tr>
<tr>
<td>REPLACE</td>
<td>replace characters in a string</td>
</tr>
<tr>
<td>LEFT</td>
<td>extract a part of a string on the left</td>
</tr>
<tr>
<td>RIGHT</td>
<td>extract a part of a string on the right</td>
</tr>
<tr>
<td>MID</td>
<td>extract a part of a string</td>
</tr>
<tr>
<td>CIRC</td>
<td>build a single character string</td>
</tr>
<tr>
<td>ASC</td>
<td>get the ASCII code of a character within a string</td>
</tr>
<tr>
<td>ATON</td>
<td>converts string to integer using hexadecimal basis</td>
</tr>
<tr>
<td>HTOA</td>
<td>converts integer to string using hexadecimal basis</td>
</tr>
<tr>
<td>CRC16</td>
<td>CRC16 calculation</td>
</tr>
<tr>
<td>ArrayTostring</td>
<td>copies elements of an SINT array to a STRING</td>
</tr>
<tr>
<td>StringToArray</td>
<td>copies characters of a STRING to an SINT array</td>
</tr>
</tbody>
</table>

Other functions are available for managing string tables as resources:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StringTable</td>
<td>Select the active string table resource</td>
</tr>
<tr>
<td>LeadString</td>
<td>Lead a string from the active string table</td>
</tr>
</tbody>
</table>
Constant Expressions

Constant expressions can be used in all languages for assigning a variable with a value. All constant expressions have a well-defined data type according to their semantics. If you program an operation between variables and constant expressions having inconsistent data types, it will lead to syntax errors when the program is compiled. Below are the syntax rules for constant expressions according to possible data types:

**BOOL:** Boolean

There are only two possible Boolean constant expressions. They are reserved keywords TRUE and FALSE.

**SINT:** Small (8 bit) integer

Small integer constant expressions are valid integer values (between -128 and 127) and must be prefixed with SINT. All integer expressions having no prefix are considered as DINTEX (integers).

**USINT / BYTE:** Unsigned 8 bit Integer

Unsigned small integer constant expressions are valid integer values (between 0 and 255) and must be prefixed with USINT. All integer expressions having no prefix are considered as DINTEX (integers).

**INT:** 16 bit Integer

16 bit integer constant expressions are valid integer values (between -32768 and 32767) and must be prefixed with INT. All integer expressions having no prefix are considered as DINTEX (integers).

**UINT / WORD:** Unsigned 16 bit Integer

Unsigned 16 bit integer constant expressions are valid integer values (between 0 and 65535) and must be prefixed with UINT. All integer expressions having no prefix are considered as DINTEX (integers).

**DINT:** 32 bit (default) Integer

32 bit integer constant expressions must be valid numbers between -2147483648 to +2147483647. DINT is the default size for integers: such constant expressions do not need any prefix. You can use 0, 08 or 16 prefixes for specifying a number in respectively binary, octal or hexadecimal basis.

**UDINT / DWORD:** Unsigned 32 bit Integer

Unsigned 32 bit integer constant expressions are valid integer values (between 0 and 4294967295) and must be prefixed with UDINT. All integer expressions having no prefix are considered as DINTEX (integers).

**LINT:** Long (64 bit) Integer

Long integer constant expressions are valid integer values and must be prefixed with LINT. All integer expressions having no prefix are considered as DINTEX (integers).

**REAL:** Single precision Floating Point Value

Real constant expressions must be a valid number, must include a decimal point ("."), if you need to enter a real expression having an integer value, add .0 at the end of the number. You can use E or E separators for specifying the exponent when entering a value using scientific notation. REAL is the default precision for floating point numbers. Such expressions do not need any prefix.

**LREAL:** Double Precision Floating Point Value

Real constant expressions must be a valid number, must include a decimal point ("."), and must be prefixed with LREAL. If you need to enter a real expression having an integer value, add .0 at the end of the number. You can use E or E separators for specifying the exponent when entering a value using scientific notation.
TIME: Time

Time constant expressions can be used to represent durations of less than 24 hours. Expressions must be prefixed by either TIME# or T#:

They are expressed as a number of hours followed by h, a number of minutes followed by m, a number of seconds followed by s, and a number of milliseconds followed by ms.

The order of units (hour, minutes, seconds, milliseconds) must be respected. You cannot insert blank characters within the time expression.

There must be at least one valid time unit letter in the expression. See examples below:

- Declare a variable cyctime with data type TIME. Following examples are valid:
  
  ```
  cyctime := $12s; // Sets cyctime to 12 seconds
  cyctime := T#1m100ms; // Sets cyctime to 1 minute plus 100 milliseconds
  cyctime := t#1h10m6s50ms; // Sets cyctime to 1 hour, 10 minutes, 6 seconds, and 50 milliseconds.
  ```

STRING: Character String

String expressions must be written between single quote marks. The length of the string cannot exceed 255 characters.

You can use the following sequences to represent a special or not printable character within a string:

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS</td>
<td>&quot;S&quot; character</td>
</tr>
<tr>
<td>S</td>
<td>Single quote</td>
</tr>
<tr>
<td>ST</td>
<td>Tab stop (ASCII code 9)</td>
</tr>
<tr>
<td>SR</td>
<td>Carriage return character (ASCII code 13)</td>
</tr>
<tr>
<td>SL</td>
<td>Line feed character (ASCII code 10)</td>
</tr>
<tr>
<td>SN</td>
<td>Carriage return plus line feed characters (ASCII code 13 and 10)</td>
</tr>
<tr>
<td>SP</td>
<td>Page break character (ASCII code 12)</td>
</tr>
<tr>
<td>$xx</td>
<td>Any character (xx is the ASCII code expressed as two hexadecimal digits)</td>
</tr>
</tbody>
</table>

**Example**

Below are some examples of valid constant expressions:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE boolean expression</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE boolean expression</td>
</tr>
<tr>
<td>SINT#127</td>
<td>Short integer</td>
</tr>
<tr>
<td>INT#200</td>
<td>Signed 16-bit integer</td>
</tr>
<tr>
<td>123456</td>
<td>DINT (32-bit) integer</td>
</tr>
<tr>
<td>16#abcd</td>
<td>DINT integer in hexadecimal basis</td>
</tr>
<tr>
<td>LINT#1</td>
<td>Long (64-bit) integer having the value &quot;1&quot;</td>
</tr>
<tr>
<td>0.0</td>
<td>0 expressed as a REAL number</td>
</tr>
<tr>
<td>1.002E3</td>
<td>1002 expressed as a REAL number in scientific notation</td>
</tr>
<tr>
<td>LREAL#1E200</td>
<td>Double precision real number</td>
</tr>
<tr>
<td>T#23h5m9s999ms</td>
<td>Maximum TIME value</td>
</tr>
<tr>
<td>TIME#0s</td>
<td>Null TIME value</td>
</tr>
<tr>
<td>T#1h123ms</td>
<td>TIME value with some units missing</td>
</tr>
<tr>
<td>'hello'</td>
<td>Character string</td>
</tr>
<tr>
<td>'name$Tag'</td>
<td>Character string with two words separated by a tab</td>
</tr>
<tr>
<td>'I'm here'</td>
<td>Character string with a quote inside ('I'm here)</td>
</tr>
<tr>
<td>'x300y'</td>
<td>Character string with two characters separated by a null character (ASCII code 0)</td>
</tr>
<tr>
<td>Sequence</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>$S</td>
<td>&quot;S&quot; character</td>
</tr>
<tr>
<td>$</td>
<td>Single quote</td>
</tr>
<tr>
<td>$T</td>
<td>Tab stop (ASCII code 9)</td>
</tr>
<tr>
<td>$R</td>
<td>Carriage return character (ASCII code 13)</td>
</tr>
<tr>
<td>$L</td>
<td>Line feed character (ASCII code 10)</td>
</tr>
<tr>
<td>$N</td>
<td>Carriage return plus line feed characters (ASCII codes 13 and 10)</td>
</tr>
<tr>
<td>$P</td>
<td>Page break character (ASCII code 12)</td>
</tr>
<tr>
<td>$xx</td>
<td>Any character (xx is the ASCII code expressed as two hexadecimal digits)</td>
</tr>
</tbody>
</table>

**Example**

Below are some examples of valid constant expressions:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE boolean expression</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE boolean expression</td>
</tr>
<tr>
<td>S#INT#127</td>
<td>Short integer</td>
</tr>
<tr>
<td>INT#2000</td>
<td>Signed 16-bit integer</td>
</tr>
<tr>
<td>123456</td>
<td>DINT (32-bit integer)</td>
</tr>
<tr>
<td>16#abcd</td>
<td>DINT integer in hexadecimal basis</td>
</tr>
<tr>
<td>LINT#1</td>
<td>Long (64-bit) integer having the value &quot;1&quot;</td>
</tr>
<tr>
<td>0.0</td>
<td>0 expressed as a REAL number</td>
</tr>
<tr>
<td>1.002E3</td>
<td>1002 expressed as a REAL number in scientific notation</td>
</tr>
<tr>
<td>LREAL#1E-200</td>
<td>Double precision real number</td>
</tr>
<tr>
<td>T#23h59m59s999ms</td>
<td>Maximum TIME value</td>
</tr>
<tr>
<td>TIME#0s</td>
<td>Null TIME value</td>
</tr>
<tr>
<td>T#1h123ms</td>
<td>TIME value with some units missing</td>
</tr>
<tr>
<td>'hello'</td>
<td>Character string</td>
</tr>
<tr>
<td>'name$Tago'</td>
<td>Character string with two words separated by a tab</td>
</tr>
<tr>
<td>'I'm here'</td>
<td>Character string with a quote inside (I'm here)</td>
</tr>
<tr>
<td>'x500y'</td>
<td>Character string with two characters separated by a null character (ASCII code 0)</td>
</tr>
</tbody>
</table>

Below are some examples of typical errors in constant expressions:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoolVar = 1;</td>
<td>0 and 1 cannot be used for Booleans - must use TRUE or FALSE</td>
</tr>
<tr>
<td>1a2b</td>
<td>Base prefix (&quot;16#&quot;) omitted</td>
</tr>
<tr>
<td>1E-200</td>
<td>&quot;LREAL#&quot; prefix omitted for a double precision float</td>
</tr>
<tr>
<td>T#12</td>
<td>Time unit missing</td>
</tr>
<tr>
<td>Tm here'</td>
<td>Quote within a string with &quot;S&quot; mark omitted</td>
</tr>
<tr>
<td>hello</td>
<td>Quotes omitted around a character string</td>
</tr>
</tbody>
</table>
IF THEN ELSE ELSIF END_IF

Statement - Conditional execution of statements

Syntax

```
IF <BOOL expression> THEN
  <statements>
ELSIF <BOOL expression> THEN
  <statements>
ELSE
  <statements>
END_IF;
```

Remarks

The IF statement is available in ST only.
The execution of the statements is conditioned by a Boolean expression.

ELSIF and ELSE statements are optional.
Multiple ELSIF statements can be used when desired.

ST Language

```
(* simple condition *)
IF Cond THEN
  Q1 := TRUE;
  END_IF;

(* binary selection *)
IF Cond THEN
  Q1 := TRUE;
  Q2 := FALSE;
ELSE
  Q1 := TRUE;
  END_IF;

(* enumerated conditions *)
IF Cond1 THEN
  Q1 := TRUE;
ELSIF Cond2 THEN
  Q1 := TRUE;
ELSIF Cond3 THEN
  Q1 := TRUE;
ELSE
  Q1 := TRUE;
END_IF;
```

CONCAT

Function - Concatenate strings

Inputs

- **IN_1** : STRING Any string variable or constant expression
- **IN_N** : STRING Any string variable or constant expression

Outputs

- Q : STRING Concatenation of all inputs

Remarks

In PBD or LD language, the block may have up to 16 inputs.
In LD language, the input (EN) enables the operation, and the output (ENO) keeps the same value as the input.
In IL or ST, the function accepts a variable number of inputs (at least 2).
Note that you also can use the "+" operator to concatenate strings.

ST Language

```
Q := CONCAT ('AB', 'CD', 'E');
(* now Q is "ABCDE" *)
```

FBD Language

```
  Q
```

ROCK SOLID PERFORMANCE. TIMELESS COMPATIBILITY.
**MLEN**

Function - Get the number of characters in a string

**Inputs**

IN : STRING  Character string

**Outputs**

NBC : DINT  Number of characters currently in the string (0 if string is empty)

**Remarks**

In LD language, the input (EN) enables the operation, and the output (ENO) keeps the same value as the input. In IL, the first input (IN: STRING) must be loaded on the stack before calling the function.

**ST Language**

```plaintext
| NBC := MLEN (IN) ;
```

---

**ASCII**

Function - Get the ASCII code of a character within a string

**Inputs**

IN : STRING  Input string
POS : DINT  Position of the character within the string
(the first valid position is 1).

**Outputs**

CODE : DINT  ASCII code of the selected character.
(or 0 if position is invalid)

**Remarks**

In LD language, the input (EN) enables the operation, and the output (ENO) keeps the same value as the input. In IL language, the first parameter (IN) must be loaded on the stack before calling the function. The other input is the operand of the function.

**ST Language**

```plaintext
| CODE := ASCII (IN, POS) ;
```

---

**MID**

Function - Extract characters of a string starting at any position within the string

**Inputs**

IN : STRING  Character string
NBC : DINT  Number of characters to extract
POS : DINT  Position of the first character to extract (first character of IN is at position 1).

**Outputs**

Q : STRING  String containing the first NBC characters of IN.

**Remarks**

The first valid position in the string is 1.
The number of characters extracted is limited to smallest value of: IN string length, Q string length, or (POS + NBC - 1).
In LD language, the input (EN) enables the operation, and the output (ENO) keeps the same value as the input.
In IL, the first input (IN: STRING) must be loaded on the stack before calling the function. Other argument operands of the function, separated by commas.

**ST Language**

```plaintext
| Q := MID (IN, NBC, POS) ;
```
ANY_TO_REAL

Operator - Converts the input into a single-precision Real value

Inputs
IN : ANY  Input value

Outputs
Q : REAL  Value converted to 32-bit Real number

Remarks
For BOOL input data types, the output is 0.0 or 1.0.
For any INTEGER input data type, the output is a REAL number with the same value.
For TIME input data types, the result is the number of milliseconds.
For STRING inputs, the output is the number represented by the string, or 0.0 if the string does not represent a valid number.
In LD language, the operation executes only if the input (EN) is TRUE. The output (ENO) keeps the same value as the input.
In IL language, the ANY_TO_REAL function converts the value pushed on the stack.

STLanguage
Q := ANY_TO_REAL (IN);  

NEG -

Operator - Performs an integer negation of the input

Inputs
IN : DINT  Integer value

Outputs
Q : DINT  Integer negation of the input

Truth table (examples)

<table>
<thead>
<tr>
<th>IN</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>-123</td>
<td>123</td>
</tr>
</tbody>
</table>

Remarks
In FBD and LD language, the block NEG can be used.
In LD language, the operation executes only if the input (EN) is TRUE. The output (ENO) keeps the same value as the input.
This feature is not available in IL language.
In ST language, "-" can be followed by a complex Boolean expression between parenthesis.

STLanguage
Q := -IN;
Q := - (IN1 + IN2);

DELETE

Function - Delete characters in a string

Inputs
IN : STRING  Character string
NCH : DINT  Number of characters to be deleted
POS : DINT  Position of the first deleted character (first character position is 1)

Outputs
Q : STRING  Modified string.

Remarks
The first valid character position is 1.
In LD language, the input (EN) enables the operation, and the output (ENO) keeps the same value as the input.
In IL, the first input (IN : STRING) must be loaded on the stack before calling the function. Other arguments are operands of the function, separated by commas.

STLanguage
Q := DELETE (IN, NCH, POS);
## ASCII conversion chart

### Decimal - Binary - Octal - Hex – ASCII Conversion Chart

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Octal</th>
<th>Hex</th>
<th>ASCII</th>
<th>Decimal</th>
<th>Binary</th>
<th>Octal</th>
<th>Hex</th>
<th>ASCII</th>
<th>Decimal</th>
<th>Binary</th>
<th>Octal</th>
<th>Hex</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000000</td>
<td>000</td>
<td>00</td>
<td>NUL</td>
<td>32</td>
<td>00100000</td>
<td>040</td>
<td>20</td>
<td>SP</td>
<td>64</td>
<td>01000000</td>
<td>100</td>
<td>40</td>
<td>@</td>
</tr>
<tr>
<td>1</td>
<td>000001</td>
<td>001</td>
<td>01</td>
<td>SOH</td>
<td>33</td>
<td>00100001</td>
<td>041</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>01000001</td>
<td>101</td>
<td>41</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>000010</td>
<td>002</td>
<td>02</td>
<td>STX</td>
<td>34</td>
<td>00100010</td>
<td>042</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>01000010</td>
<td>102</td>
<td>42</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>000011</td>
<td>003</td>
<td>03</td>
<td>ETX</td>
<td>35</td>
<td>00100011</td>
<td>043</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>01000011</td>
<td>103</td>
<td>43</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>000100</td>
<td>004</td>
<td>04</td>
<td>EOT</td>
<td>36</td>
<td>00100100</td>
<td>044</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>01000100</td>
<td>104</td>
<td>44</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>000101</td>
<td>005</td>
<td>05</td>
<td>ENQ</td>
<td>37</td>
<td>00100101</td>
<td>045</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>01000101</td>
<td>105</td>
<td>45</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>000110</td>
<td>006</td>
<td>06</td>
<td>ACK</td>
<td>38</td>
<td>00100110</td>
<td>046</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>01000110</td>
<td>106</td>
<td>46</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>000111</td>
<td>007</td>
<td>07</td>
<td>BEL</td>
<td>39</td>
<td>00100111</td>
<td>047</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>01000111</td>
<td>107</td>
<td>47</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>001000</td>
<td>008</td>
<td>08</td>
<td>BS</td>
<td>40</td>
<td>00101000</td>
<td>050</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>01010000</td>
<td>110</td>
<td>48</td>
<td>H</td>
</tr>
<tr>
<td>9</td>
<td>001001</td>
<td>009</td>
<td>09</td>
<td>HT</td>
<td>41</td>
<td>00101001</td>
<td>051</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>01010001</td>
<td>111</td>
<td>49</td>
<td>I</td>
</tr>
<tr>
<td>10</td>
<td>001010</td>
<td>010</td>
<td>10</td>
<td>LF</td>
<td>42</td>
<td>00101010</td>
<td>052</td>
<td>2A</td>
<td>*</td>
<td>74</td>
<td>01010100</td>
<td>112</td>
<td>4A</td>
<td>J</td>
</tr>
<tr>
<td>11</td>
<td>001011</td>
<td>011</td>
<td>11</td>
<td>VT</td>
<td>43</td>
<td>00101011</td>
<td>053</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>01010110</td>
<td>113</td>
<td>4B</td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>001100</td>
<td>012</td>
<td>12</td>
<td>FF</td>
<td>44</td>
<td>00110000</td>
<td>054</td>
<td>2C</td>
<td>.</td>
<td>76</td>
<td>01011000</td>
<td>114</td>
<td>4C</td>
<td>L</td>
</tr>
<tr>
<td>13</td>
<td>001101</td>
<td>013</td>
<td>13</td>
<td>CR</td>
<td>45</td>
<td>00110001</td>
<td>055</td>
<td>2D</td>
<td>-</td>
<td>77</td>
<td>01011010</td>
<td>115</td>
<td>4D</td>
<td>M</td>
</tr>
<tr>
<td>14</td>
<td>001110</td>
<td>014</td>
<td>14</td>
<td>SO</td>
<td>46</td>
<td>00110010</td>
<td>056</td>
<td>2E</td>
<td>_</td>
<td>78</td>
<td>01011011</td>
<td>116</td>
<td>4E</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>001111</td>
<td>015</td>
<td>15</td>
<td>SI</td>
<td>47</td>
<td>00110011</td>
<td>057</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>01011101</td>
<td>117</td>
<td>4F</td>
<td>O</td>
</tr>
<tr>
<td>16</td>
<td>010000</td>
<td>020</td>
<td>20</td>
<td>DLE</td>
<td>48</td>
<td>00110100</td>
<td>060</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>01100000</td>
<td>120</td>
<td>60</td>
<td>P</td>
</tr>
<tr>
<td>17</td>
<td>010001</td>
<td>021</td>
<td>21</td>
<td>DC1</td>
<td>49</td>
<td>00110101</td>
<td>061</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>01100001</td>
<td>121</td>
<td>61</td>
<td>Q</td>
</tr>
<tr>
<td>18</td>
<td>010010</td>
<td>022</td>
<td>22</td>
<td>DC2</td>
<td>50</td>
<td>00110110</td>
<td>062</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>01100010</td>
<td>122</td>
<td>62</td>
<td>R</td>
</tr>
<tr>
<td>19</td>
<td>010011</td>
<td>023</td>
<td>23</td>
<td>DC3</td>
<td>51</td>
<td>00110111</td>
<td>063</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>01100011</td>
<td>123</td>
<td>63</td>
<td>S</td>
</tr>
<tr>
<td>20</td>
<td>010100</td>
<td>024</td>
<td>24</td>
<td>DC4</td>
<td>52</td>
<td>00111000</td>
<td>064</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>01101000</td>
<td>124</td>
<td>64</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>010101</td>
<td>025</td>
<td>25</td>
<td>NAK</td>
<td>53</td>
<td>00111001</td>
<td>065</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>01101001</td>
<td>125</td>
<td>65</td>
<td>U</td>
</tr>
<tr>
<td>22</td>
<td>010110</td>
<td>026</td>
<td>26</td>
<td>SYN</td>
<td>54</td>
<td>00111010</td>
<td>066</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>01101010</td>
<td>126</td>
<td>66</td>
<td>V</td>
</tr>
<tr>
<td>23</td>
<td>010111</td>
<td>027</td>
<td>27</td>
<td>ETB</td>
<td>55</td>
<td>00111011</td>
<td>067</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>01101101</td>
<td>127</td>
<td>67</td>
<td>W</td>
</tr>
<tr>
<td>24</td>
<td>011000</td>
<td>030</td>
<td>30</td>
<td>CAN</td>
<td>56</td>
<td>00111100</td>
<td>070</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>01110000</td>
<td>130</td>
<td>58</td>
<td>X</td>
</tr>
<tr>
<td>25</td>
<td>011001</td>
<td>031</td>
<td>31</td>
<td>EM</td>
<td>57</td>
<td>00111101</td>
<td>071</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>01110001</td>
<td>131</td>
<td>59</td>
<td>Y</td>
</tr>
<tr>
<td>26</td>
<td>011010</td>
<td>032</td>
<td>32</td>
<td>SUB</td>
<td>58</td>
<td>00111101</td>
<td>072</td>
<td>3A</td>
<td>;</td>
<td>90</td>
<td>01110010</td>
<td>132</td>
<td>5A</td>
<td>Z</td>
</tr>
<tr>
<td>27</td>
<td>011011</td>
<td>033</td>
<td>33</td>
<td>ESC</td>
<td>59</td>
<td>00111101</td>
<td>073</td>
<td>3B</td>
<td>`</td>
<td>91</td>
<td>01110011</td>
<td>133</td>
<td>5B</td>
<td>[</td>
</tr>
<tr>
<td>28</td>
<td>011100</td>
<td>034</td>
<td>34</td>
<td>F5</td>
<td>60</td>
<td>00111101</td>
<td>074</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>01110100</td>
<td>134</td>
<td>5C</td>
<td>]</td>
</tr>
<tr>
<td>29</td>
<td>011101</td>
<td>035</td>
<td>35</td>
<td>GS</td>
<td>61</td>
<td>00111101</td>
<td>075</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>01110101</td>
<td>135</td>
<td>5D</td>
<td>]</td>
</tr>
<tr>
<td>30</td>
<td>011110</td>
<td>036</td>
<td>36</td>
<td>RS</td>
<td>62</td>
<td>00111110</td>
<td>076</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>01111000</td>
<td>136</td>
<td>5E</td>
<td>^</td>
</tr>
<tr>
<td>31</td>
<td>011111</td>
<td>037</td>
<td>37</td>
<td>US</td>
<td>63</td>
<td>00111111</td>
<td>077</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>01111011</td>
<td>137</td>
<td>5F</td>
<td>_</td>
</tr>
</tbody>
</table>