Eurotherm®

by Schneider Electric

# Eurotherm PAC
# Raw Comms

User Guide

HA030511/5

August 2017 (Issue 5)

# Contents

# Chapter 1

# Overview

This manual describes the *Raw Communications* feature implemented on LIN based products and software and is designed to be read in conjunction with the related documents listed in "Related Documents" on page 5.

## Prerequisites (User Knowledge Assumptions)

The reader of this document is assumed to have a good working knowledge of LIN blocks and LIN system functionality. This document is not intended as a starting point to understanding LIN in general and more specifically, communication protocols. This manual should be read after or in conjunction with the documents shown in the following section.

### Related Documents

| Document number | Document title |
|---|---|
| HA082375 U003 | LIN Blocks Reference Manual |
| HA084012 U003 | Application & Control Modules Manual |
| HA029280 | Visual Supervisor Handbook |
| HA029587 | Visual Supervisor Tutorial |
| HA028898 | T2550 PAC Handbook |
| HA263001 U055 | LINtools Engineering Studio User Guide |

### Terms

The following terms are used in this manual:

| Term | Meaning |
|---|---|
| LIN | Eurotherm's scalable DCS including elements thereof. |
| Visual Supervisor | Eycon™ 10/20 Visual Supervisor |

# What is Raw Communications?

The RAW_COM Function Block provides the facility for LIN based products to directly control the transmission and reception of messages and protocols over a serial link and can also facilitate multi-node applications if required. It is available for applications where it is necessary to have low level control of the serial communications port to provide the flexibility to construct or analyse messages and protocols exactly as transmitted or received via a serial link.

The block is based on the PC 3000 Raw Comms block but includes the additional ability to execute structured text (ST) Actions. It performs basic functions first and then executes any ST Actions that have been created. The ST is stored in a file and is handled in the same way as for an Action block and cannot access data outside the Raw Comms block. For protocols that are too complex to be handled using the ST inside the block, an SFC should be used to drive the block which consequently provides considerable flexibility and is not limited by the constraints inherent in the block.

The RAW_COM block can be assigned to any serial port if more than one port is available and is designed for use by LIN instruments supporting serial communications, e.g. T2550 PAC and Visual Supervisor. It provides a wide range of low level facilities including:

- Direct access to messages as transmitted or received via the serial link.
- Independent control of message transmission and reception (limited to the same Baud rate).
- Selectable echoing of received characters when required.
- User selectable *Delete* sequence for character deletion when required.
- Can be used in conjunction with SFCs for complex protocol support.
- Additional wide string variable blocks to assist in processing long byte and character sequences.

## Example Applications

The RAW_COM function block can typically be used in the following example applications:

- Communication with devices using non-standard protocols, simple or complex.
- Sending reports to special purpose printers, e.g. Ticket printers.
- Communication with remote Terminals or display devices.
- Communications with Weigh Scales in a single or multi-drop configuration.

# Compatibility

For products that do not support Raw Comms but are caching Raw Comms function blocks from products that support Raw Comms, data within the cached block is available for use and wiring in the expected manner.

## Supported Products

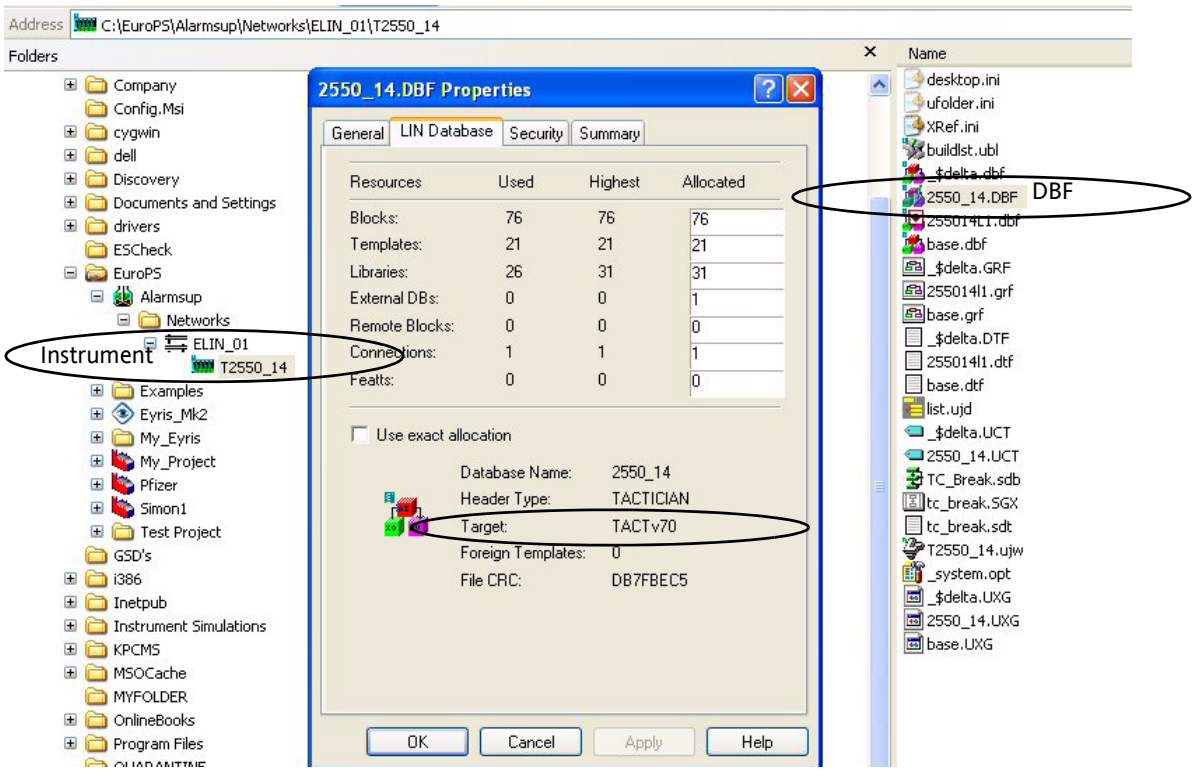
Raw Comms is supported by the following products and software. Upgrading a database to support this function is detailed in "Upgrading an Existing Database" on page 25.

| Product | Version |
|---|---|
| T2550 PAC | Version 7.0 onwards |
| Eycon™ 10/20 Visual Supervisor | Version 5.0 onwards |
| Operations Server\NTSE | Version 4.9 onwards |
| LINtools (Tactician) | Version 4.9 onwards |
| T940X, T800 and T640 | Not Supported |

# Does my Existing Database Support Raw Comms?

This is achieved by checking the 'Instrument Version' that the DBF was created for. Referring to the following figure, right click on the appropriate DBF as circled below and select the **LIN Database** tab to establish the **Target** version. With reference to the section, "Supported Products" on page 7, check that the **Target** version is compatible to support Raw Comms.

If the Instrument is a pre-Raw Comms Version, the DBF does not support Raw Comms and can be upgraded by referring to "Upgrading an Existing Database" on page 25.




# T2550 PAC Redundant Processors Support

The Raw Comms block does not fully support dual redundant operation in terms of a seamless processor changeover. However it is possible to have a Raw Comms block running in a dual redundant system but the following points must be observed.

- The comms ports on a dual redundant system are wired in parallel; consequently transmission is inhibited by the firmware on the secondary processor.
- If a processor changeover occurs when the transmit processing is in the PENDING state, it changes to the ERROR state and any bytes queued for transmission may be lost. Similarly for receive processing, resulting in the possible loss of incoming bytes/characters.

# Application and Control Modules

Six additional blocks as shown below have been added to the Application and Control Module library to assist with processing long byte and character sequences. These blocks can be used as appropriate and are covered in detail in the *Application & Control Modules Manual, HA084012 U003*.

| Block name | Block function |
|---|---|
| BYTESEQ48S | 20 Variables x 48 Bytes |
| BYTESEQ256S | 4 Variables x 256 Bytes |
| BYTESEQ1020 | 1 Variable x 1020 Bytes |
| WIDESTR24S | 20 Variables x 24 Characters |
| WIDESTR128S | 4 Variables x 128 Characters |
| WIDESTR510 | 1 Variable x 510 Characters |

# Raw Comms Licensing

Raw Comms is licensed as part of the Master Communications options for both the T2550 PAC and Visual Supervisor. If the Master Communications option was not purchased at the time of order placement, a licence for the relevant instrument is required. Please contact a Eurotherm office for further details.

# Chapter 2

# RAW_COM Function Block

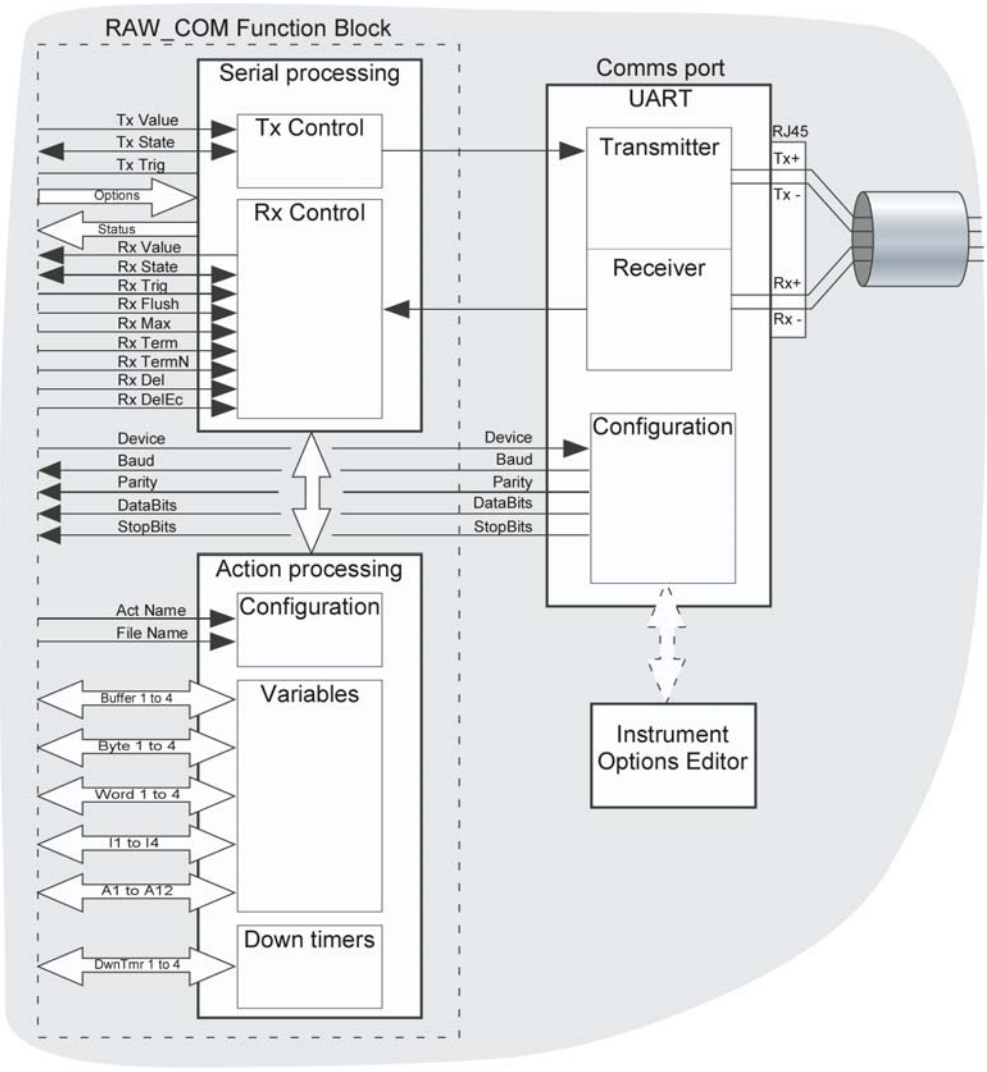
The RAW_COM function block provides low level control of a serial communications port and also has the additional ability to execute structured text (ST) Actions. The block performs basic functions first and then executes any ST Actions that have been created. The ST is stored in a file (.STO) and is dealt with in the same manner as for an Action block, see "Creating a Structured Text (ST) Action" on page 23.

It is designed for use by instruments supporting serial communication ports, i.e. T2550 PAC, Eycon 10/20 Visual Supervisor. For full details refer to the *LIN Blocks Reference Manual, HA082375U003* for detailed information.

**Note:** Note: On Instruments where multiple user tasks are supported (e.g. T2550 PAC) the RAW_COM block can be allocated to run on any of these tasks.

# RAW_COM Functional Diagram

The following figure depicts the functional relationship between the RAW_COM function block and the communication port with its associated UART. The communication port fields (Baud, Parity, for example) are configured using the Instrument Options Editor and are therefore read-only from within the RAW_COM function block. Alarm processing is also included in the block functionality but is not shown in the figure.




# RAW_COM Function Block Fields

Refer to the *LIN Blocks Reference Manual, HA082375U003* for full details.

**Dbase, Block, Type.** See *LIN Blocks Reference Manual HA082375U003* for details of these 'header' fields.

**ActName.** The name (8-characters max) given to any structured Text (ST) Action created for the block as defined in *FileName.STO* (specified below). The file holds the compiled structured text. Refer to "Creating a Structured Text (ST) Action" on page 23.

**FileName.** Specifies the file name (8-characters max) containing the Structured Text (ST) Action specified by the *ActName* parameter above. Refer to "Creating a Structured Text (ST) Action" on page 23.

**Tx_Value.** This is the buffer used to hold the character sequence to be transmitted. The buffer can contain up to 1020 characters.

**Tx_State.** This parameter indicates the current state of the transmitter. Values are **OK**, **PENDING**, **ERROR** and **WRITE**. Transmission can be initiated by setting this parameter to **WRITE**.

**Tx_Trig.** This is provided to allow control of the Raw Comms block by wiring. Setting this parameter to TRUE initiates transmission.

**Options.** Bitfield, setting communications options. All Options default to FALSE.

- **AltTerm.** When TRUE this allows *Rx_Term* below to be treated as a set of alternative single termination bytes any of which identifies the end of a line of input. When FALSE, *Rx_Term* is treated as a sequence of bytes.
- **FlshOnTx.** When TRUE, the receive buffer is always flushed immediately prior to any transmission.
- **DropRefl.** Inhibit character reflection if during transmission from the LIN device's serial port, characters are reflected back into it, for example, 3-wire cabling. When TRUE for all characters transmitted, an equal amount of received characters are ignored.

**Note:** No check is performed to ensure that the ignored characters match the transmitted characters.

- **SlaveTx.** This field is for future use. It is intended to facilitate tri-stating of the serial port transmitter for multi-drop slave applications when set TRUE. Current supported hardware always supports tri-stating mode regardless of setting this field to TRUE or FALSE.
- **Rx_Del.** When TRUE, this enables the automatic processing of delete characters in the input stream, that is, the removal of the delete character itself plus the preceding character (assuming the latter has not already been processed and reached the *Rx_Value* buffer). The byte value to be interpreted as the delete character is defined by the *Rx_Del* field below. If *Options.Echo* is also set, then the delete character is not echoed, instead the optional sequence defined by the *Rx_DelEc* field below is substituted on the condition that a character was actually deleted.
- **Echo.** When TRUE, all received data (excluding loopback) is retransmitted, used for example with a dumb terminal.

**Note:** The echoed data may be conditioned by the *Rx_DelEc* parameter below.

- **LoopBack.** When TRUE, all transmitted data (excluding echo) appears as input. Used for test purposes only.

- **TxMute.** When TRUE, this inhibits transmission of all data (including echo), but has no other effect, that is, the internal transmit processing continues to function as if the data has been transmitted.
- **RxMute.** When TRUE, discards incoming data (does NOT discard loopback data).

**Device.** This identifies the Comms port to which the block refers. For a T2550 PAC it is always RAW1. For Eycon 10/20 Visual Supervisor it may be RAW1 or RAW2, as there are 2 serial ports available.

**Baud.** The Tx/Rx baud rate (independent baud rates cannot be set for Tx and Rx). The value of this read-only field is configured using the Instrument Options Editor, see "Instrument Options Editor" on page 17. Supported baud rates are 1200, 2400, 4800, 9600, 19200 and 38400.

**Parity.** The value of this read only field is configured using the Instrument Options Editor, see "Instrument Options Editor" on page 17. Supported values are NONE, ODD and EVEN.

**DataBits.** The value of this read only field is configured using the Instrument Options Editor, see "Instrument Options Editor" on page 17. It sets the number of bits per character for both receive and transmit. If this is set less than 8 then the most significant (8 - DataBits) bits is ignored when sending and forced to zero when receiving. The number of bits per character supported in this field are 5, 6, 7 or 8.

**Note:** Current hardware does not support 5 or 6 Data Bits.

**StopBits.** The value of this read only field is configured using the Instrument Options Editor, see "Instrument Options Editor" on page 17. It sets the number of stop bits expected by the receiver and sent by the transmitter. The number of stop bits supported are 1 or 2.

**Alarms.**

- **Software.** Asserted, if a sumcheck error in block's RAM data occurs or caching failure.
- **NoAction.** The Structured text (ST)) Action as defined in *ActName* above or *Filename.STO* cannot be found.
- **BadActn.** Set if an ST evaluation error occurs at runtime.
- **BadDev.** The configured device is invalid, e.g. Requires configuration using the *Instrument Options Editor*. For Visual Supervisor only, can also mean a Raw Comms licensing failure.
- **Device.** A low-level communication device failure, Parity error, for example. The status field provides details of the failure.
- **UserAlm1.** Controlled from structured text.
- **UserAlm2.** Controlled from structured text.
- **UserAlm3.** Controlled from structured text.

- **UserAlm4.** Controlled from structured text.
- **Combined.** True if any alarm is active. It adopts the same status message and priority number as the block's highest priority active alarm.

**Status.** Status bits indicate the following error conditions. 0 = no error has occurred.

- **RxChLost.** An internal receive buffer has overflowed causing characters to be lost. This may also be caused by very heavily loaded applications and/or large comms packet sizes, especially if above 512 bytes.)
- **RxParity.** A parity error was detected on a received character.
- **RxOver**. An overrun error was detected on a received character.
- **RxFrame.** A framing error was detected on a received character.
- **RxBreak.** A break condition has been detected on the receive line (not supported on current hardware).
- **RxFrcErr.** *Rx_State* below has been set to **ERROR** to force an error.
- **TxChLost.** An attempt has been made to send a new message before the last transmission has completed.
- **TxFrcErr.** *Tx_State* above has been set to **ERROR** to force an error.

**Rx_Value.** This is the character buffer used to hold the character sequence received from the serial port. The buffer can contain up to 1020 characters. It is volatile.

**Rx_State.** This parameter indicates the current state of the receiver. Values are **OK**, **PENDING**, **ERROR**, **READ** and **FLUSH**. Reception can be initiated by setting this parameter to **READ**. Setting it to **FLUSH** clears any characters that have been received and are waiting to be copied into the *Rx_Value* buffer. Characters already in the *Rx_Value* buffer are not affected.

**Rx_Trig.** This is provided to allow control of the Raw Comms block by wiring. Setting this parameter TRUE initiates reception.

**Rx_Max.** This specifies the maximum number of characters held in the internal receive buffer before being passed on to the block's *Rx_Value* buffer. The value should be in the range 1-1020. 1020 is the maximum length of the *Rx_Value* message.

**Note:** Characters are passed to the *Rx_Value* buffer either on the maximum number of characters as defined in *Rx_Max* being reached or a termination sequence of characters has been received as defined in *Rx_Term*.

**Rx_Term.** This specifies a termination sequence of characters that is used to identify the end of a line of input. If *Rx_Term* is left blank the input is read from an internal receive buffer into the *Rx_Value* buffer until *Rx_Max* characters have been received. If *Options.AltTerm* is set, then these characters are treated as alternatives rather than a sequence.

**Rx_TermN.** If *Rx_Term* is not blank, this specifies an additional number of characters to be read after the termination sequence has been received. This is intended to simplify receiving a message which for example has a terminating sequence followed by a CRC or BCC. The limits for this field are 0 - 1020.

**Rx_Del.** Specifies an optional delete character, to be applied to the incoming character stream if *Options.Rx_Del* is set. This would typically be used when communicating with a terminal of some kind.

**Rx_DelEc.** If *Options.Echo* and *Options.Rx_Del* are both set, this specifies the string to be transmitted whenever an actual deletion takes place as a result of the character *Rx_Del* being received. This is typically "$08$20$08", that is, back-space, space, back-space.

**Note:** If a character is not available for deletion, then the defined string in *Rx_DelEc* is not transmitted.

## Function Block Variables

General Purpose Variables accessed by the associated Raw Comms structured text and/or from external SFCs. They can be used as workspace, to hold results and to accept input values.

**Buffer1** to **Buffer4**. These provide access to four character buffers which can be used as workspace by the associated Raw Comms structured text or from an external SFC. Each buffer can contain up to 256 characters.

**Byte1** to **Byte4**. Four bitfields, each of which can be treated as an 8-bit integer or 8 separate Booleans.

**Word1** to **Word4**. Four bitfields, each of which can be treated as a 16-bit integer or 16 separate Booleans.

**I1** to **I4**. Four 32-bit signed integer variables.

**DwnTmr1** to **DwnTmr4**. Four down-timers which indicate seconds as floating point. When non-zero, they count down automatically (updating at every block update) until they reach zero. They can be written to and read at any time.

**A1** to **A12**. Twelve floating point variables.

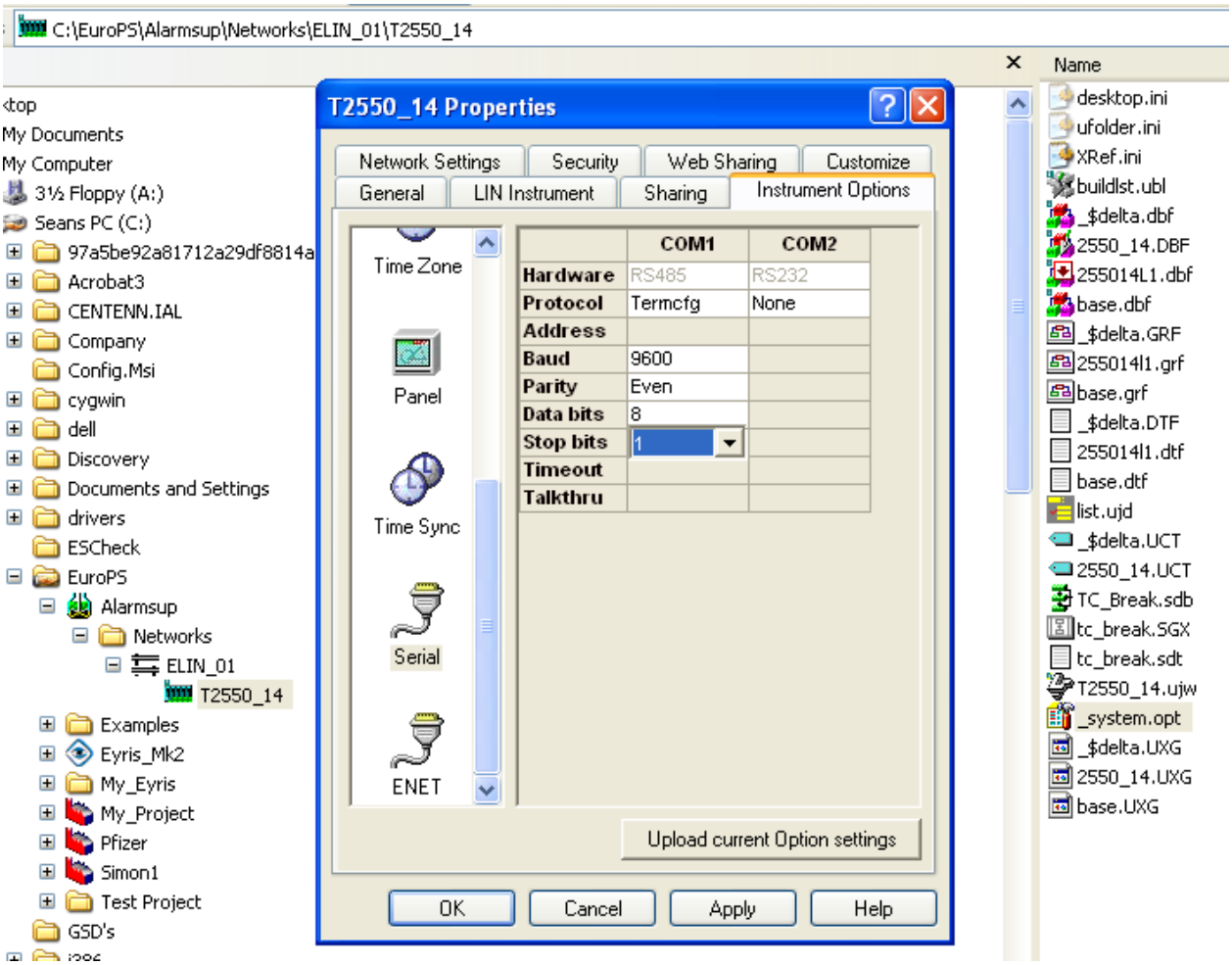
# Instrument Options Editor

The serial communications port is configured using the Instrument Options Editor as previously mentioned in this document (sometimes referred to as the 'CNF' editor). It has an associated file that is created when used and contains the configuration details which can then be edited as appropriate. This file is known as '_system.opt'.

Referring to the following figure, right click on the appropriate Instrument as circled below (e.g. T2550_14) and select the **Properties** item and select the **Instrument Options** tab. Locate the **Serial** icon and select it to edit the serial ports as required.

It is also possible to simply double click on the '_system.opt' file located in the Instrument folder to open it. It also shows the same dialogue as shown in figure 2.3.

If the instrument is currently connected via 'Eurotherm Network', the current option settings can be uploaded by selecting the **Upload current Option Settings** button. Once edited the new settings can be applied by selecting the **Apply** button.



# Byte Sequence Format

The Byte Sequence (ByteSeq) format is covered in greater depth in the LINtools on-line help by pressing the F1 key at any time when using LINtools. The RAW_COM block uses this format for four fields as follows:

- Tx_Value
- Rx_Value
- Rx_DelEc
- Buffer1 to Buffer4

The function of these fields are described in "RAW_COM Function Block Fields" on page 12, but in general terms they cater for long strings and are implemented using Structured Text (ST) either in an SFC or an Action associated with the RAW_COM block, see section 3.5.

**Note:** Byte Sequence fields cannot be wired.

# Structured Text (ST) and Raw Comms

This document does not cover the use or implementation of ST as it is covered in detail in the LINtools on-line help facility and should be used to aid in ST applications, specifically the sections as follows:

- Constants in ST
  - String Constants
  - Wide String and Byte Sequence Conversion
- Operators and Functions in ST
  - Lists the Operators and Functions and where relevant shows worked examples by clicking the hyperlinks

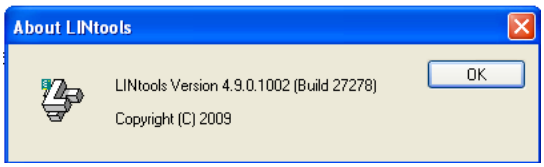
# Chapter 3

# LINtools Applications

This chapter describes the implementation of Raw Communications using the LINtools configurator. It assumes that the user is familiar with LINtools menus and LIN database configuration as described in the *LINtools Engineering Studio User Guide*, *HA263001 U055*. Chapter 2 of this user guide provides reference information associated with the RAW_COM function block.

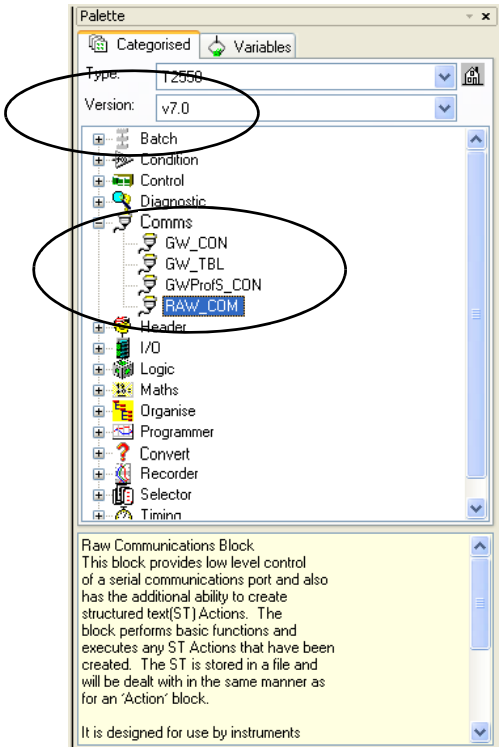
## About LINtools

Ensure that the version of LINtools being used supports Raw Comms. This is achieved by selecting **About LINtools...** in the LINtools **Help** menu and with reference to "Supported Products" on page 7. The following figure shows the **About LINtools** dialogue box stating at least this version of LINtools being used.



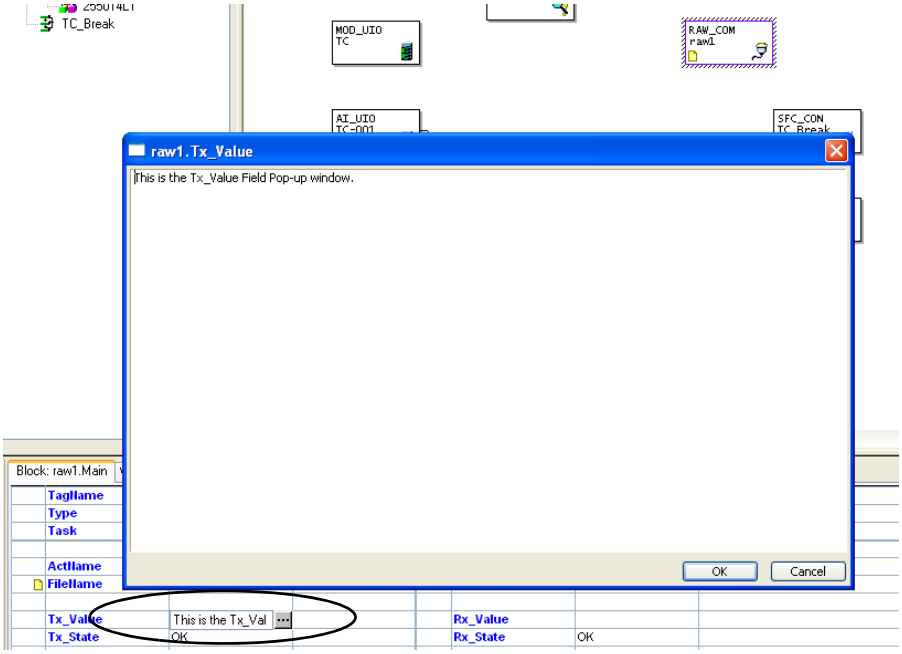
# LINtools Palette

The RAW_COM function block is located in the **Comms** item located in the **Palette** as shown in the following figure. Ensure that the Instrument Version supporting Raw Comms has been selected with reference to "Supported Products" on page 7.



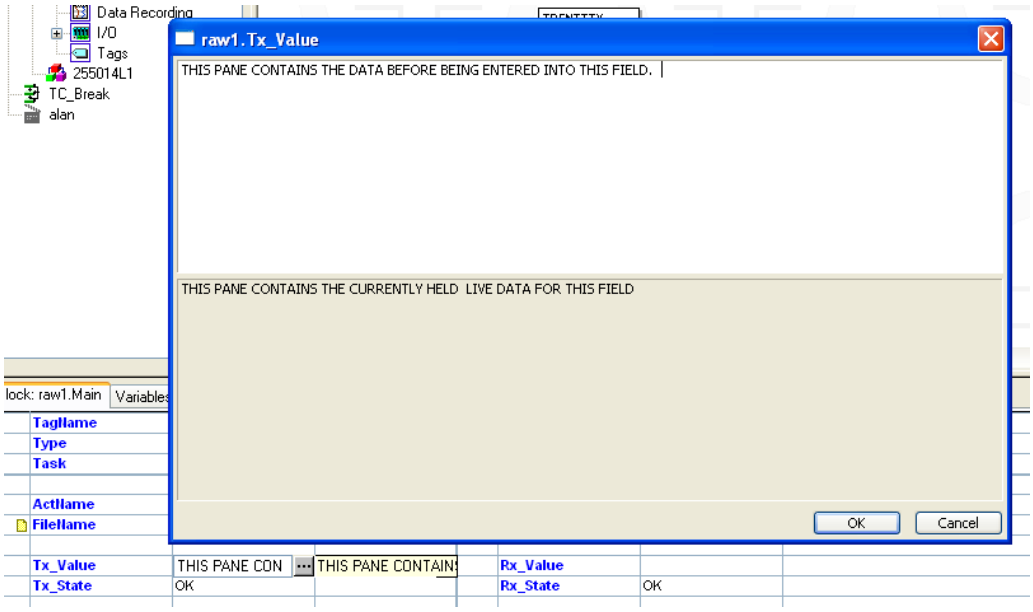
# RAW_COM Function Block extended fields

The Tx_Value parameter is designed to handle up to 1020 characters which consequently is larger than the standard field parameters allows. With reference to the following figure, when editing 'Tx_Value' and clicking within the field an icon appears at the right hand side as shown below. Clicking on the icon opens the Tx_Value Pop-up window to allow entry of large strings as shown below.



# Online Connection

When connected on-line to the RAW_COM block, for large fields (for example, Tx_Value), a dual pane window as activated in "RAW_COM Function Block extended fields" on page 21 is available to enter and view data as shown in the following figure. All other fields operate as expected in the on-line connection mode, and live data is shown in the column adjacent to each field.




# Online Reconfiguration

On instruments that support on-line reconfiguration (the T2550 PAC, for example), it is possible to create, remove or replace a RAW_COM block as required. However, this will only be successful if the serial port protocol has already been configured to **Raw**, via the Instrument Options Editor. For further details, refer to "Instrument Options Editor" on page 17.

If an associated action file has been specified for the created or edited block, when in **TRY** mode this file is read and reloaded. This method can therefore be used to achieve an associated Action reload, which is not possible while under normal running conditions.

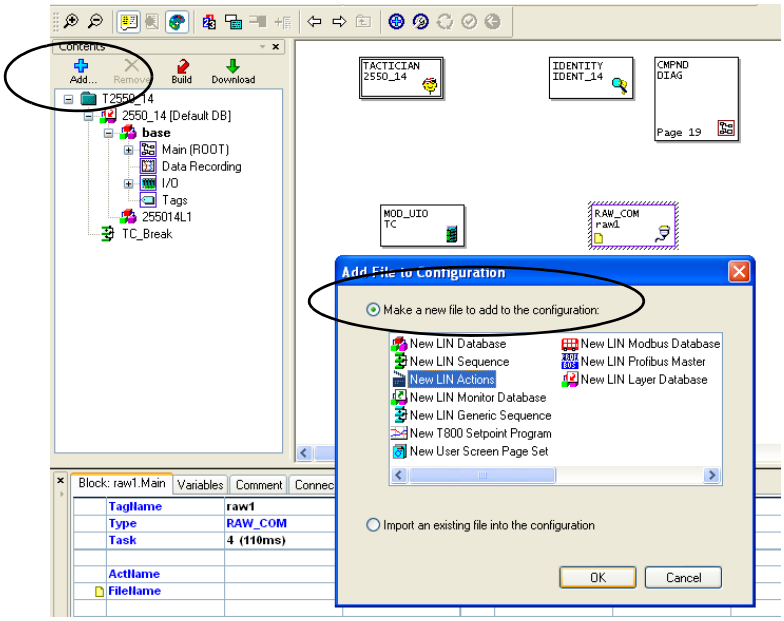
For more information on associated Action files refer to"Creating a Structured Text (ST) Action" on page 23.

# Creating a Structured Text (ST) Action

When developing a protocol using the RAW-COM block, a Structured Text (ST) Action can be used to implement the associated protocol structures. With reference to "RAW_COM Function Block Fields" on page 12, the two fields that are configured with the associated ACTION itself and the filename containing the ACTION are *ActName* and *FileName* respectively. Assuming that a RAW-COM block is loaded in the function block database, the procedure to create an ACTION is described as follows and is based on Example 1 as described in "Example 1: Simple Terminal Application" on page 34.

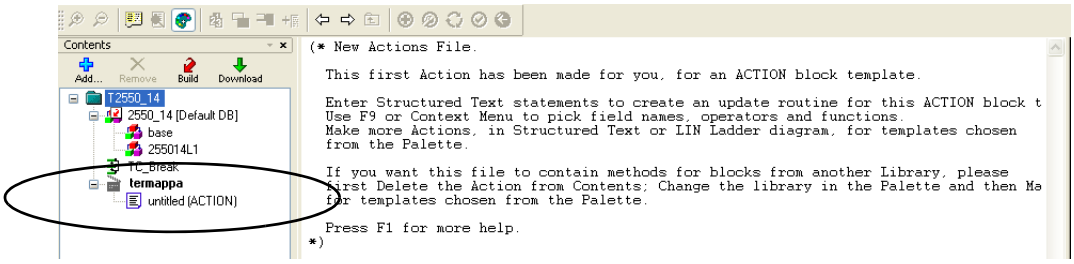
### To create a Structured Text Action

1. Ensure that a relevant version of LINtools supporting Raw Communications is installed, refer to "Supported Products" on page 7.
2. Click on the **Add** button located in the **Contents** pane. With reference to the following figure, the **Add file to Configuration** pop-up window is now available. Select the radio button, M**ake a new file to add to the configuration**, and then select **New LIN Actions** followed by clicking the **OK** button.




3. A new pop-up window appears prompting for a file name. Enter a meaningful file name limited to 8-characters maximum (termappa in the example below) and click **OK**. The new file is added to the contents pane structure as shown in the following figure.

4 Open the file name by double clicking on it which in turn creates a new **untitled (ACTION)** as shown in the following figure.

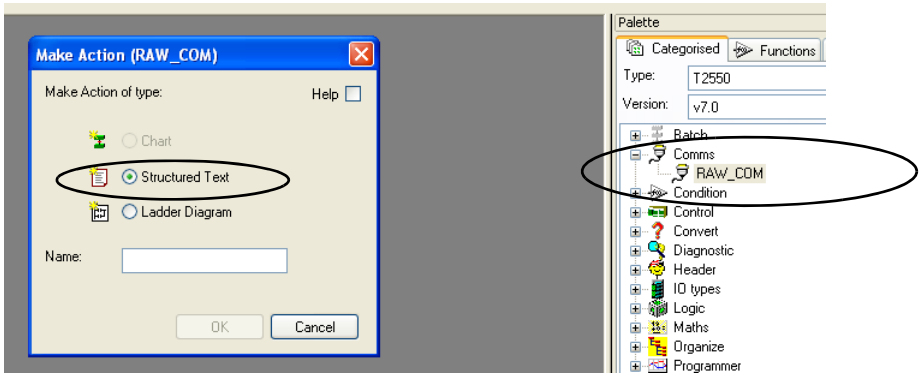



5 Delete the **untitled (ACTION)** item as shown in the above figure by right-clicking on it and select **Delete**.

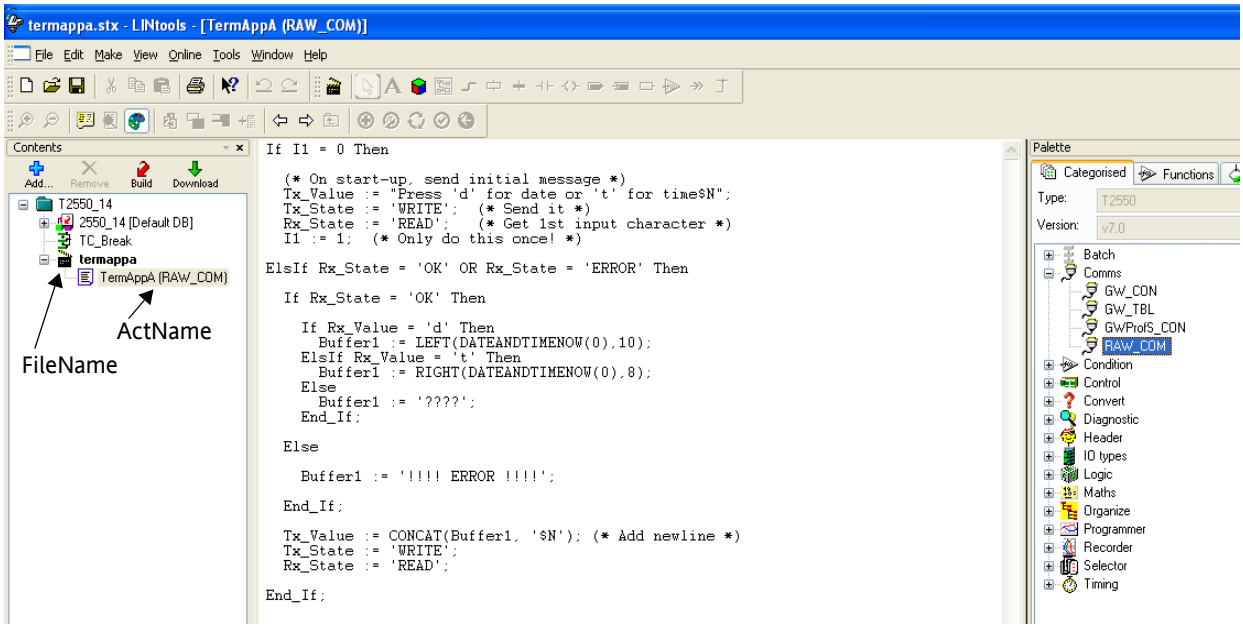
6 If the Palette is not shown, select the **Palette** item from the **View** menu to activate it.

7 Select the **Categorised** tab in the **Palette** and from the drop down **Type** menu, select the appropriate instrument, T2550 PAC, for example. Ensure that the appropriate version is also selected in the **Version** drop down menu to support Raw Comms as shown in "Supported Products" on page 7.

8 Once step 7 is completed the **Palette** contains the Function Block Library headings. Expand the **Comms** heading and double-click the **RAW_COM** item which opens a **Make Action (RAW_COM)** pop-up window as shown in the following figure.



**9** Enter a meaningful Action name limited to 8-characters maximum (TermAppA in the example below), then ensure that the **Structured Text** radio button is highlighted and finally click **OK**. The Action editor is now available for creation of the Action using ST as appropriate. Save the file before closing the editor. A working example of ST Action code as used for Example 1 in "Example 1: Simple Terminal Application" on page 34, is shown in the following figure.



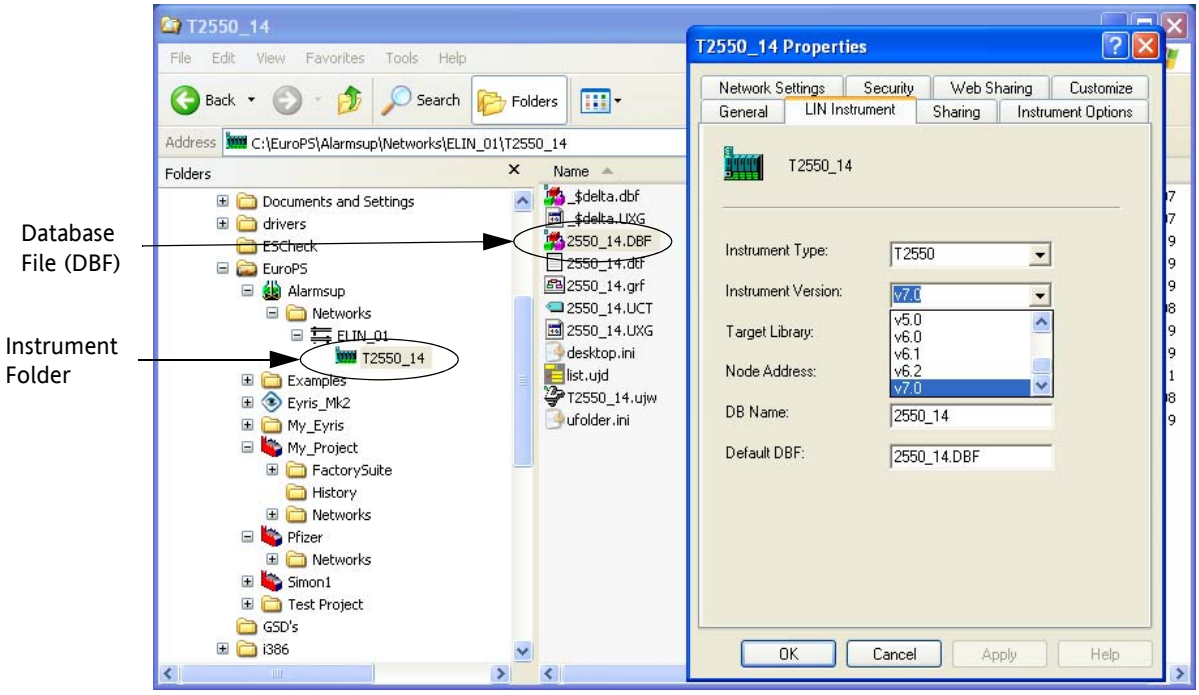

# Upgrading an Existing Database

If a database was created for an instrument pre-Raw Comms and it is now required to implement Raw Comms in that database, an upgrade procedure must be adhered to. This requires the existing database to be loaded into the relevant version of LINtools, the database header block deleted and replaced with the relevant version header block, save, close and re-open the database. This procedure uses an example based on a T2550 PAC described as follows:

### To upgrade an existing database

**1** Ensure that a relevant version of LINtools supporting Raw Comms is installed. For details, refer to "Supported Products" on page 7.

**2** Locate the Instrument folder which contains the database(.DBF) to be upgraded. It is normally be found in the C:\EuroPS\<Project name>\Networks\<Network name>\<Instrument name> folder. An example, using an instrument name of T2550_14 and a database name of 2550_14.DBF is shown in the following figure.




**3** Referring to the above figure, right-click on the instrument folder and select **Properties**, which opens the **LIN Instrument Properties** dialog box. Select the **LIN Instrument** tab and from the pick list in the **Instrument Version** drop down box, select **v7.0** or greater (**v5.0** or greater for Visual Supervisor) and click **OK**.

**4** Open the appropriate .DBF file contained in the instrument folder by double-clicking it, which automatically opens LINtools. It is expected that the associated .GRF file is available which facilitates a graphical representation of the database. If not, select the **Create FBD Layout** item from the LINtools **View** menu.

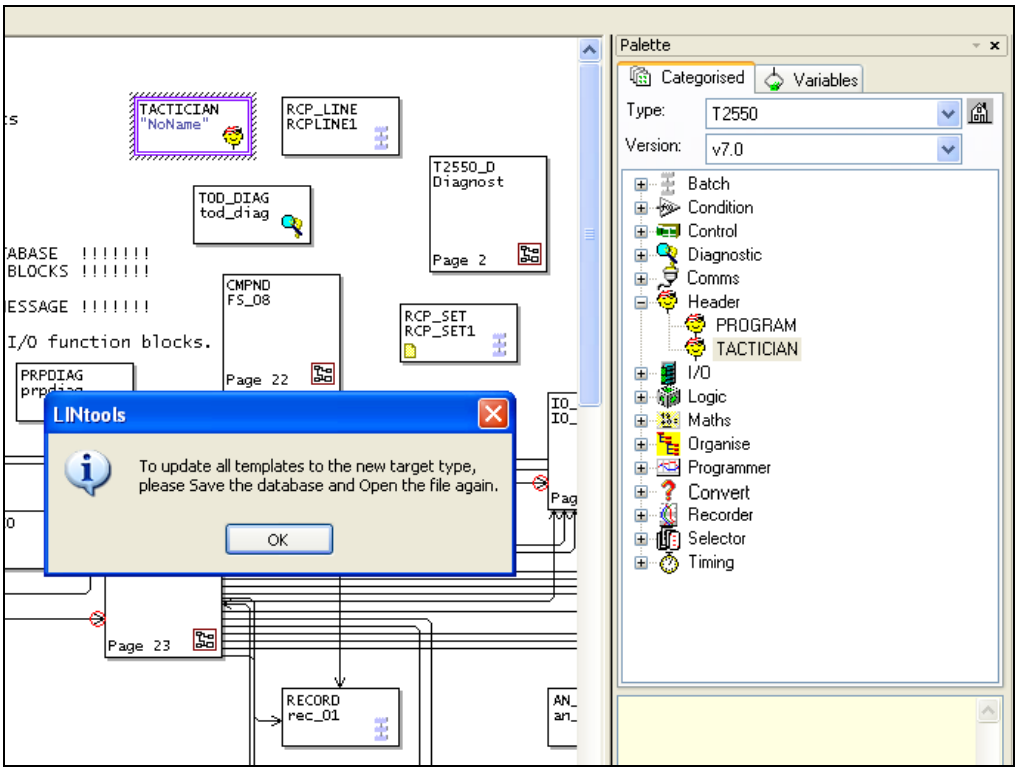
**5** Locate and then open the database Header Block by double-clicking within it. The Header Block details are now shown at the bottom of the LINtools window.

**Note:** Make a record of all connections, comments and parameter settings, e.g. TagName, LIN Name, BrownOut, ColdStrt, etc. Comments and Connections can be found in the relevant tabs.

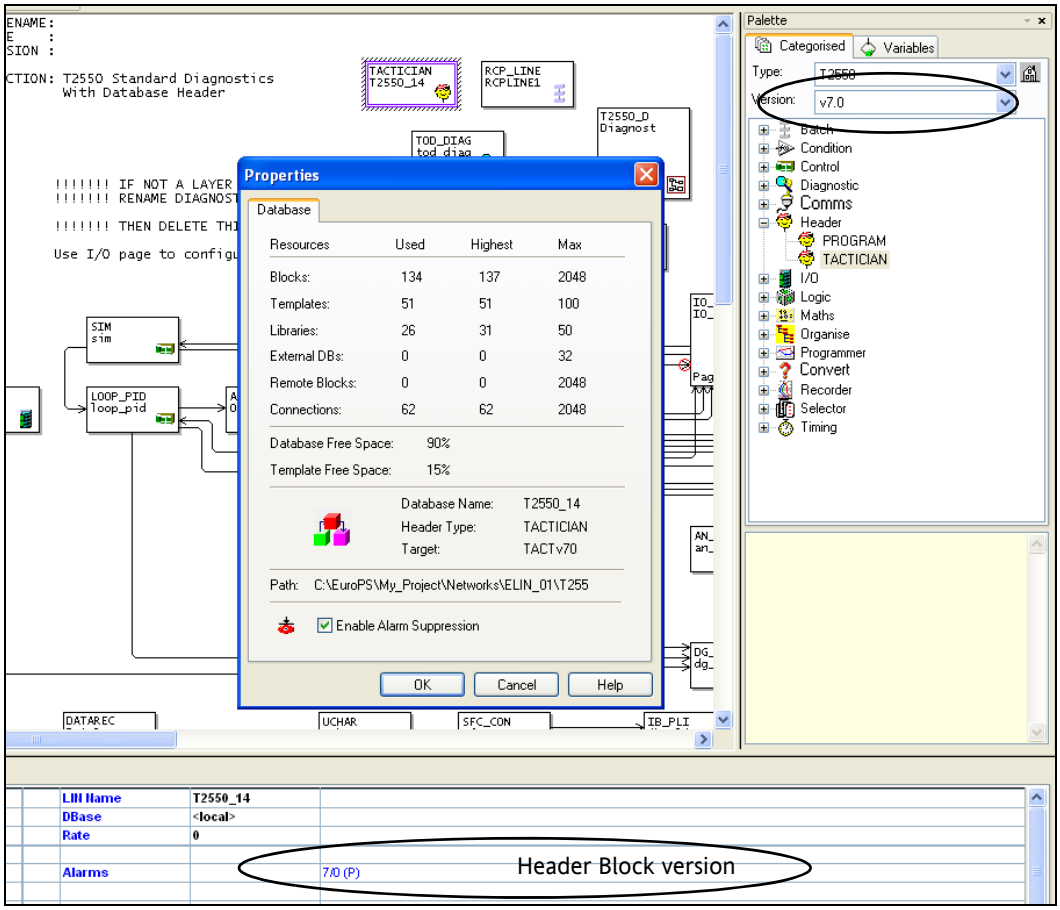
**6** Ensure that the Header Block is still selected, denoted by a blue highlight fringed with diagonal lines, press the **Delete** key to delete the header block.

**7** Referring to the following figure, if not already shown, open the Palette by selecting the **Palette** item from the LINtools **View** menu. Select the appropriate instrument type from the drop down box, then from the **Instrument Version** drop down box select **v7.0** or greater (**v5.0** or greater for Visual Supervisor). From the **Palette** library, select and drag the appropriate Header Block and place where required. Once placed, a LINtools information box appears, reminding the user to save, close and re-open the database. Click **OK** on the information box to close it. Using the information as recorded in step 5, enter the **TagName** and **LIN Name** in the new Header Block. Then save, close and re-open the database.

**Note:** The target Instrument Firmware and Palette version must be matched as close as possible.



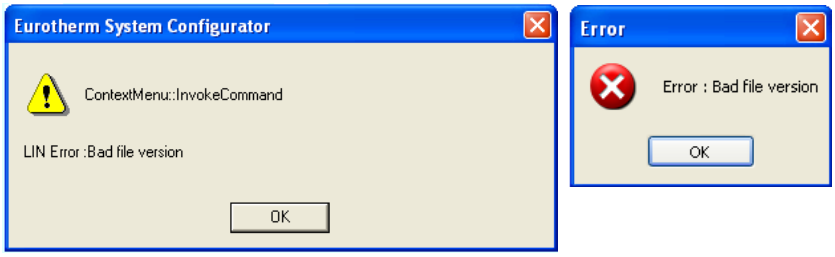
**8** Before fully reconstructing the Header Block, it is recommended to check that the correct version has been applied in the Header Block. Referring to the following figure, open the **Palette** and ensure that the **Version** number shown is **v7.0** or greater (**v5.0** or greater for Visual Supervisor). Open the Header Block by double clicking on it, ensuring that version **7 / 0** or greater (version **5 / 0** or greater for Visual Supervisor) is shown in the field adjacent to the **Alarms** field. Once satisfied that the correct version has been applied, the Header Block can now be fully reconstructed using the information as recorded in step 5.




**9** Ensuring that the target instrument is at the appropriate firmware version by referring to"Supported Products" on page 7, the database can now be edited to include a RAW_COM block and downloaded using the **Build** and **Download** buttons located in the LINtools contents pane.

# Error Messages

When downloading files created in LINtools that support Raw Comms and the firmware version of the target instrument is pre-Raw Comms (refer to "Supported Products" on page 7) typical error messages are given as shown in the following figure.



# Chapter 4

# Configuration and Examples

This chapter describes the setup decisions that need to be made prior to implementing a Raw Comms configuration (whether to use just an associated Action or just an associated SFC or an associated SFC interacting with an associated Action as discussed below). A decision on whether just an associated Action can be used on its own without an associated SFC is determined by the required protocol complexity whilst considering that the maximum size of an associated Action file object code is 1000 bytes.

This chapter also provides worked examples to give an understanding of achieving protocols using an associated Action. Example 3 demonstrates the limit of what can be achieved using just an associated Action.

## Raw Comms Configuration

Raw Comms is configured in two or three parts, determined by the complexity of the required protocol as discussed in the following sub-sections. The RAW_COM block allows for execution of an associated Action which typically would be used for non-complex protocols. For complex protocols an Action can also be used for common routine tasks in association with an SFC. For further information to aid configuration please refer to the *LIN Blocks Reference Manual, HA082375 U003*. For reference to Simple Variable Application Blocks to assist in Raw Comms processing also refer to Chapter 9 of the *Application & Control Modules Manual HA084012 U003*.

## Configuration of the RAW_COM function block fields

The RAW_COM block fields are configured prior to any associated Action or SFC creation. Refer to the chapter, "RAW_COM Function Block" on page 11 for field descriptions and a functional diagram. Also refer to the *LIN Blocks Reference Manual, HA082375 U003.*

**Note:** On Instruments where multiple user tasks are supported (for example, the T2550 PAC) the RAW_COM block can be allocated to run on any of these tasks.

## Associated Structured Text Action

For non-complex protocol applications an associated Structured Text (ST) Action can be used to facilitate the required protocol processing. This requires that two fields are configured in the RAW_COM block. These fields are configured with the associated Action itself and the Filename containing the Action, namely *ActName* and *FileName* respectively. The procedure to create an ACTION is described in "Creating a Structured Text (ST) Action" on page 23. To force the RAW_COM block to load a new associated Action, refer to "Online Reconfiguration" on page 22.

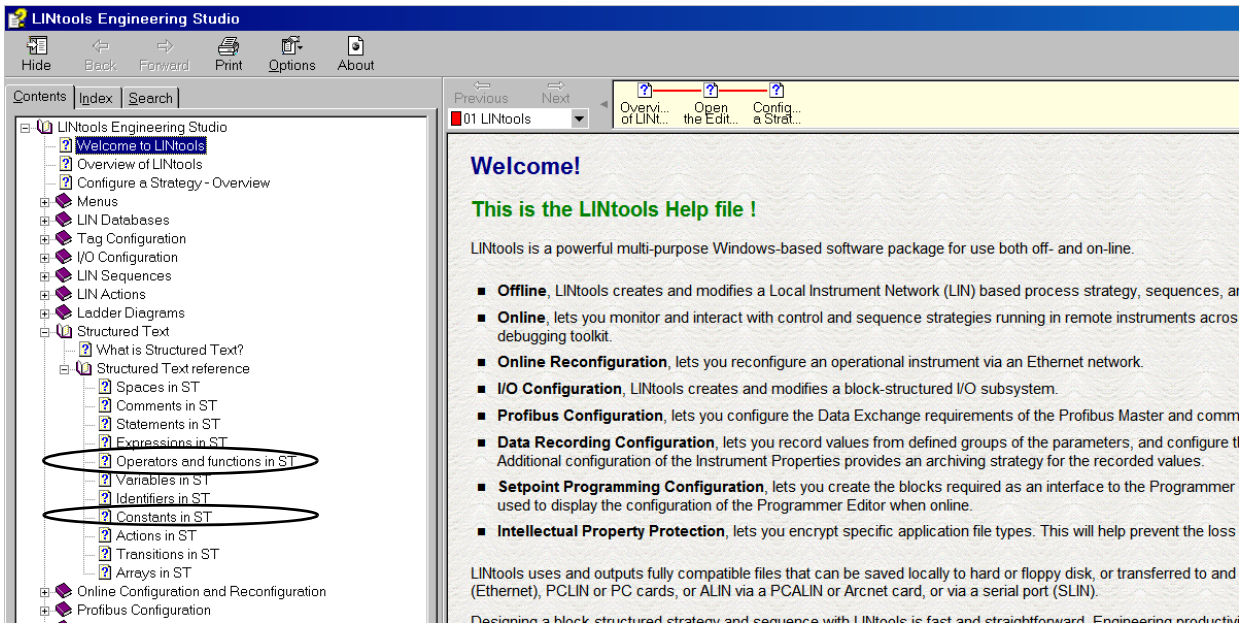
## Associated SFC

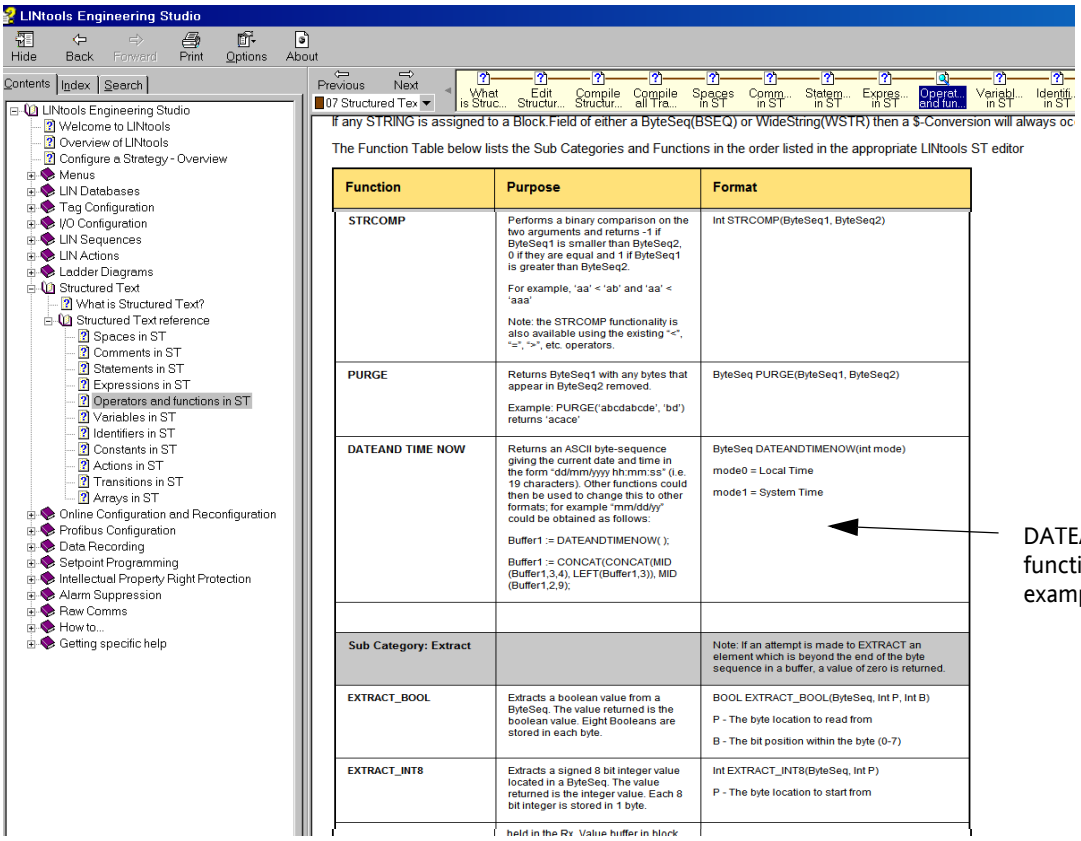
For more complex protocols an SFC is required to manage the overall protocol state processing requirements. However in this case it may be possible to process common routine tasks using an associated Action which interacts with an SFC. The SFC manages the overall protocol state processing tasks.

An associated SFC interacting with the RAW_COM block is considered just like any other SFC and the process for creating and implementing the SFC is no different.

# Further Information and Help

The LINtools on-line help provides information on creation of SFCs, Structured Text (ST), Actions and general help. Specifically, for further information relating to ST usage available to assist in Raw Comms processing, it is strongly recommended to see the following help topics, 'Operators and functions in ST' and 'Constants in ST' as shown in following two figures.



# Examples

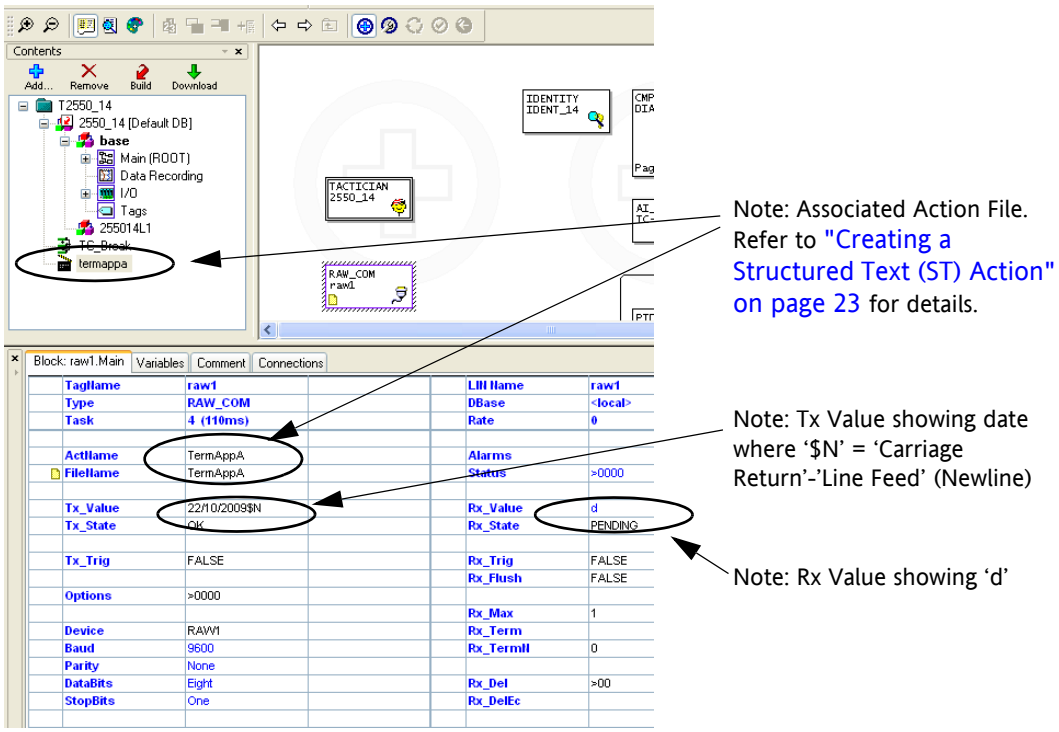
The following sections provide examples of the configuration of Raw Comms.

## Example 1: Simple Terminal Application

The first example is an extremely basic, but is a complete application communicating with a dumb terminal such as HyperTerminal on Windows®. Its function is simply to print out the current date or time when the 'd' or 't' key is pressed respectively.

Referring to the following figure, the database contains two function blocks as follows:

- An appropriate header block for the target instrument, e.g. a TACTICIAN block for a T2550 PAC, or an 'Eycon-20' block for a Visual Supervisor.
- A RAW_COM block. All fields should be left with their default values, except that 'ActName' and 'FileName' are both set to 'TermAppA'. Refer to section 2.2 for field descriptions.




### Associated Action

There is a single action file, TermAppA.STX, containing a single action, also named TermAppA. For details of how to create this, refer to "Creating a Structured Text (ST) Action" on page 23. The contents of the action are as follows:

```
If I1 = 0 Then

  (* On start-up, send initial message *)
```

```
  Tx_Value := "Press 'd' for date or 't' for time$N";
  Tx_State := 'WRITE';  (* Send it *)
  Rx_State := 'READ';   (* Get 1st input character *)
  I1 := 1;  (* Only do this once! *)

ElsIf Rx_State = 'OK' OR Rx_State = 'ERROR' Then

  If Rx_State = 'OK' Then

    If Rx_Value = 'd' Then
      Buffer1 := LEFT(DATEANDTIMENOW(0),10);
    ElsIf Rx_Value = 't' Then
      Buffer1 := RIGHT(DATEANDTIMENOW(0),8);
    Else
      Buffer1 := '????';
    End_If;

  Else
    Buffer1 := '!!!! ERROR !!!!';
  End_If;

  Tx_Value := CONCAT(Buffer1, '$N'); (* Add newline *)
  Tx_State := 'WRITE';
  Rx_State := 'READ';
End_If;
```

Before attempting to run this application, the first (or only) serial port must have been set up via the Instrument Options Editor. Refer to "Instrument Options Editor" on page 17 for further details. The comms settings (baud, parity, etc) must match with the terminal being used. If the terminal is not RS-485, then a suitable converter (KD485 converting to RS-232, for example) is also required. Also ensure that the target instrument being used is licensed for Raw Comms (licensing is the same as for Modbus Master).

## Testing

With reference to the following figure, on start-up, the following message should appear on the terminal:

```
Press 'd' for date or 't' for time
```

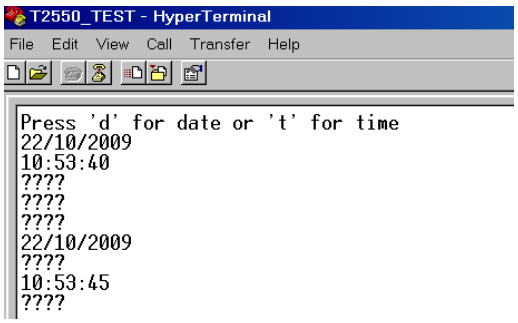
If the 'd' key is pressed then a date, such as

```
22/01/2011
```

should be displayed. Similarly for the 't' key a time, such as

```
10:53:40
```

should appear. Any other character causes '????' to be displayed.




# Example 2: ASCII Modbus

This is a simple example used to communicate with an instrument via the ASCII variant of the Modbus protocol. Its function is to repeatedly read the *PV* value in the instrument which is then written to the *A1* field of the RAW_COM block. The fields *I1-I3* are used to count transmit errors, receive errors and timeouts respectively. *I4* counts message length errors while *Word1* counts checksum errors. Refer to "RAW_COM Function Block Fields" on page 12 for field descriptions.

The RAW_COM block is set up with the following non-default values:

- Tx_Value = ':010347000001B4$N' – this is a request to read PV.
- Rx_Max = 16 – this is longer than the expected response
- Rx_Term = '$N' – this is the expected termination sequence (CR-LF) of the response.
- Options.FlshOnTx = TRUE – this ensures that any erroneous data is always cleared before each transaction.

## Associated Action

For details of how to create an Action, refer to "Creating a Structured Text (ST) Action" on page 23. Only a very limited check on the response validity has been performed in the example ST code below. The contents of the action are as follows:

```
If (Tx_State = 'OK') AND (Rx_State = 'OK')  Then
  If (Byte1 = 1 (* indicating successful read *) ) Then
    (* Check response is correct length *)
    If (LEN(Rx_Value) = 15) Then
      (* Verify checksum *)
      Byte4 := 0;
      For Byte3:= 2 To 10 By 2 Do
        Byte4 := Byte4 - SCAN('X', MID(Rx_Value, 2, Byte3));
```

```
      End_For;
      If Byte4 = SCAN('X', MID(Rx_Value, 2, 12)) Then
        (* Value is expressed in hex, in units of tenths *)
        A1:=SCAN('X',MID(Rx_Value,4,8))/10;
      Else
        Word1 := Word1 + 1; (* Word1 counts checksum errors *)
      End_If;
    Else
      I4 := I4 + 1; (* I4 counts length errors *)
    End_If;
  End_If;
  (* Do next read immediately *)
  Rx_State := 'READ';
  Tx_State := 'WRITE';
  Byte1 := 1; (* Initialise to 'success' *)
  DwnTmr1 := 5;  (* Initialise 5 sec timeout *)
ElsIf (Tx_State = 'ERROR') Then
  Byte1 := 0;
  I1 := I1 + 1;  (* I1 counts Tx errors *)
  Tx_State := 'OK';
ElsIf (Rx_State = 'ERROR') Then
  Byte1 := 0;
  I2 := I2 + 1;  (* I2 counts Rx errors *)
  Rx_State := 'OK';
ElsIf (DwnTmr1 = 0) Then (* Timeout *)
  Byte1 := 0;
  Tx_State := 'OK';
  Rx_State := 'OK';
  I3 := I3 + 1;  (* I3 counts timeouts *)
End_If;
```

## Example 3: Bisync Protocol

The following example has been used to communicate with a legacy Eurotherm S6360 process controller via the EI bisync (binary) protocol. The *SP* value is written from the field *A1*, and the *OP* and *ER* values are read into the *A2* and *A3* fields respectively. These three transactions are cycled through using the *Byte* fields to keep track of which is in progress.

Specific points worth noting for this example:

- This is a binary protocol, hence the frequent appearance of base 16 numbers (for example,16#7F), and '$' escape sequences in literal strings.
- The data encoding is quite intricate, requiring the use of *shift* and *replace* functions.
- When reading, *Rx_TermN* is set to 1 as the protocol expects one extra byte (the checksum) following the terminating ETX.
- When writing SP, the BCC function is used to calculate the required checksum.
- The instrument number has been hard coded as 0 in this example.

**Note:** The object code maximum size of an associated Action is 1000 bytes.

This example represents about the extreme of what is practically achievable using only an associated action file (especially as it is close to the 1000 byte object code limit). For anything more complicated, an SFC would be a more manageable choice.

## Associated Action

For details of how to create an Action, refer to "Creating a Structured Text (ST) Action" on page 23. The contents of the action are as follows:

```
If Rx_State = 'OK' OR Rx_State = 'ERROR' Then


  If Rx_State = 'ERROR' Then
    Byte4.Bit2 := 1;  (* Error flag *)
  Else
    Byte4.Bit2 := 0;
    If Byte3 = 2 Then
      (* Decode OP, first checking response looks valid *)
      If EQUAL(LEFT(Rx_Value, 2), '$02$89') Then
        Word2 := SHL16(EXTRACT_UINT8(Rx_Value,3), 14) +
                 SHL16(EXTRACT_UINT8(Rx_Value,4) AND 16#7F, 7) +
                 (EXTRACT_UINT8(Rx_Value,5) AND 16#7F);
        A2 := Word2 / 100.0;
      End_If;
    ElsIf Byte3 = 3 Then
      (* Decode ER, first checking response looks valid *)
      If EQUAL(LEFT(Rx_Value, 2), '$02$A3') Then
        Word3 := SHL16(EXTRACT_UINT8(Rx_Value,3), 14) +
```

```
                    SHL16(EXTRACT_UINT8(Rx_Value,4) AND 16#7F, 7) +
                     (EXTRACT_UINT8(Rx_Value,5) AND 16#7F);
          A3 := Word3 / 10.0;
        End_If;
      End_If;
    End_If;

Byte3 := Byte3 + 1;  (* Move on to next request *)
  If Byte3 > 3 Then
    Byte3 := 1;
  End_If;
If Byte3 = 1 Then
    (* Set SP *)
    Buffer1 := '$04$80$80$02$92$84$00$00$03$00';
    I1 := A1 * 10;
    Word1 := I1;
    Buffer1 := REPLACE_UINT8(Buffer1,6, 16#84 OR
  SHR16(Word1,14));
    Buffer1 := REPLACE_UINT8(Buffer1,7, 16#80 OR SHR16(Word1,7)
  );
    Buffer1 := REPLACE_UINT8(Buffer1,8, 16#80 OR Word1);
    Byte1   := BCC(MID(Buffer1,5,5),0,0);
    Tx_Value := REPLACE_UINT8(Buffer1,10,16#80 OR Byte1);
    Rx_Max := 1;
  ElsIf Byte3 = 2 Then
    (* Read OP *)
    Tx_Value := '$04$80$89$89$05';
    Rx_Max := 20;
    Rx_Term := '$03'; (* ETX *)
    Rx_TermN := 1; (* BCC *)
  Else
    (* Read ER *)
    Tx_Value := '$04$80$A3$A3$05';
    Rx_Max := 20;
    Rx_Term := '$03'; (* ETX *)
    Rx_TermN := 1; (* BCC *)
  End_If;
```

```
    Rx_State := 'READ';
    Tx_State := 'WRITE';
    DwnTmr1 := 10;

  ElsIf DwnTmr1 = 0 Then
    Rx_State := 'ERROR';
  End_If;
```

# Index

## P



## R



## S

## T



## U



## W

Scan for local contents

Eurotherm Ltd
Faraday Close
Durrington
Worthing
West Sussex
BN13 3PL
Phone: +44 (0) 1903 268500
www.eurotherm.co.uk



HA030511 Issue 5 (CN35714)