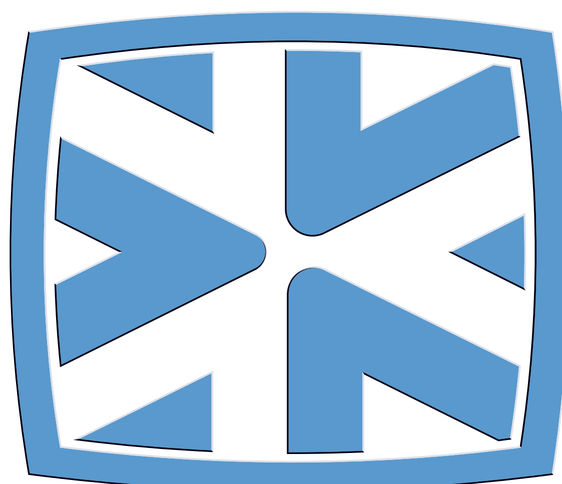
eliwell

Televis Drivers for Third Party Devices
Guide to the Creation of Drivers



CONTENTS

1 Introduction ... 3
2 General notes ... 3
3 Driver Televis Wizard ... 4
3.1 Instructions for the compilation of the Excel sheet containing the MODBUS controllers features. ... 4
3.2 Instructions for the use of Tool MakeDriverWizard ... 8
4 Advanced programming ... 9
4.1 General ... 9
4.2 Communication driver ... 10
4.3 Management of MODBUS/RTU device parameters ... 13
4.4 Script language ... 14
4.5 Structure of the MODBUS drivers for DTM ... 15
4.6 Definition of the parameters of MODBUS devices ... 26
4.7 Example of MODBUS driver ... 27
4.8 Appendix ... 28
5 Observations and limitations ... 29
6 Notes on the use of the driver with TelevisNet ... 29

1 Introduction

This technical note details the procedures for the creation of a communication driver for third party devices that use the MODBUS/RTU serial protocol on Televis networks.

To be able to use the created driver, you will need a system with the following characteristics:

- Eliwell software application compatible with third party devices or application that uses the SoftGate driver
- RS232/RS485 PC Interface converter
- A license for the applications specified above, enabled for the use of third party protocols
- One or more Smart**Adapter** devices connected to one or more MODBUS devices

For information on wiring specifications and on the limits of SmartAdapter devices, see the applicable technical sheet.

2 General notes

The driver for third party controllers is a single file, i.e. a module developed with VB Script, which contains all the items required to monitor analog inputs, digital inputs, statuses and alarms, and to manage the parameters of a specific MODBUS unit.

”Base” MODBUS drivers can be created using the tool that enables to use a pre-compiled Excel file to automatically build a driver with the basic features required for a correct operation.

The resulting driver can then be enhanced by any programmer familiar with Visual Basic Script through the integration of the advanced Softgate features for the Televis protocol (for example global commands and advanced detection of the unit resources).
For information on the advanced programming features for the drivers, see the chapter **Advanced Programming**.

The name of the driver file has a specific format:

EXTPRT_XXXX_YYYY.edr

EXTPRT: **Ext**ernal **Pr**o**t**ocol (that differs from the Telvis code)

XXXX: **First identification code of the unit:**
This code must always be set to **7FFC** (dec. **32764**)

YYYY: **Second identification code of the unit:**
Progressive code or code selected by the driver’s author, which has no effect on the driver itself. This must be an hex number <= 7FFF (32767 decimal). If the unit is an Eliwell unit, it is advisable to use the

following format for this code: low byte for the firmware mask, high and progressive byte to identify the model that corresponds to the mask (for example 01D6 stands for: mask D6 (214 decimal), model 1.

.edr: Driver extension: **e**liwell **dr**iver

This file is saved in the user-specified folder that contains all the *edr drivers that the user intends using.
The path of this folder must be inserted in the DtmCgf.ini configuration file, under <Windows>\System32, in the following location:

```
#
# EXTERNAL DRIVER
#
[EXTPRT]
PATH="C:\Eliwell\EwScript"  (sample path !!!)
```

For additional information, see also the **Add_DtmCfg.ini** file.

3 Driver Televis Wizard

This charter explains how to create a driver using a simplified procedure.
Excel is required both to compile the information sheet and to run the compilation with the **MakeDriverWizard.exe** software tool.

3.1 Instructions for the compilation of the Excel sheet containing the MODBUS controllers features.

The installation CD and the MODBUS support documentation contain the Excel file called **ModbusTlvTemplate.xls**.
This file must be renamed by the user with a more significant name. It is generally advisable to use a name that refers to the unit for which the driver is being created.
This sheet already contains a compilation sample, which will have to be edited by the user accordingly.

The Excel sheet has 4 sections that need to be correctly completed:

1. General section
2. Identify section
3. Resources section
4. Parameters section

3.1.1 General section

The required values are the following:

- User Name: Name of the driver author (Used for descriptive purposes only)
- Date: Date of creation of the driver

(Used for descriptive purposes only)

- Driver file name: Important; see format in paragraph General notes of this document
- RS485: Two values are required for RS485 communications between the SmartAdapter and the MODBUS unit.
- Baud rate: (one of the following numbers)
    - 1200
    - 2400
    - 4800
    - 9600
    - 19200

- Parity:(one of the following letters)
    - N = None
    - E = Even
    - = Odd

- Address for scan Two values to limit the scan range for the unit (max. from 0 to 255).

3.1.2 Identify section

The required data is the following:

- Number of Commands: Number of commands required to identify the unit.

Complete, for each command, one line of the table as follows:

- Command: Byte array of the MODBUS frame in hex format, separated by ';' Example: &H2B;&HE;&H4;&H2
- Condition number: Number of conditions that must be true for the unit to be identified
- #° Condition: Condition expressed by a formula similar to the following: BYTE(12)=&H44 Where BYTE() is the byte array of the reply frame, starting from the address byte that coincides with BYTE(0).

3.1.3 Resources section

Monitored resources are of four types:

- Analog Input

For this type of resource, it is necessary to specify:

- Number of analog inputs

- Number of commands required to read the analog inputs
- Enter the unit of measurement code for each analog input
  (see Table 2 of Appendix)
- Enter the Dec.Pos. code for each analog input
  (see Table 1 of Appendix)

- Digital Input

  - Number of digital inputs
  - Number of commands required to read the digital inputs

- Statuses
  - Number of statuses
  - Number of commands required to read the statuses
  - Enter the appropriate status code for each status

  For information on the status codes, see document "TelevisResourceCoding".

- Alarms
  - Number of alarms
  - Number of commands required to read the alarms
  - Enter the appropriate alarm code for each alarm

  For information on the alarm codes, see document "TelevisResourceCoding".

It is then necessary to complete the table with the commands required to read the resources and decode the read values.
It is necessary to complete a line for each command.
Example: if a single command is sufficient to read analog inputs, digital inputs and statuses, but two commands are required for alarms, it is necessary to specify a total of 5 commands and therefore complete 5 lines of the table.

The fields that have to be inserted below each line are:

Type: Resource type code:
- 1: Analog Input
- 2: Digital Input
- 3: Statuses
- 4: Alarms

Command: Byte array of the MODBUS frame in hex format, separated by ';'
Example: &H3;&H20;&H75;&H0;&H4

Value #: Formula used to decode the reply frame and attain the value related to the read resource:

- The formula for analog inputs is:
  BYTE(3)*256+BYTE(4);SIGNED
  where BYTE() is the array of bytes of the reply frame, starting from the address byte that coincides with BYTE(0), whereas the keyword SIGNED is added when the expected value has a sign.
- The formula for other resources is:
  (BYTE(4) AND &H10) <> 0
  If this condition is true, the resource value is = 1, otherwise = 0.

3.1.4 Parameters section

- Enter the number of expected parameters (cell D52)
- Complete, for each parameter, the lines (starting from 55) as follows:

Code: **Numerical ID of the parameter**
Label: **Parameter label**
Unit Code: **Unit of measurement code (see Table 3 of Appendix)**
Min Type: **1 = The minimum limit is absolute**
2 = The minimum limit refers to another parameter
Min Value **Minimum value of parameter**
Or, index of the parameter that defines it
Max Type: **1 = The maximum limit is absolute**
2 = The maximum limit refers to another parameter
Max Value **Maximum value of parameter**
Or, index of the parameter that defines it
Default Value: Default value of parameter
Type: **Parameter type code (see Table 4 of Appendix)**
Resolution **Resolution code (see Table 5 of Appendix)**

Command Read **Byte list (in hex format) of frames separated by ';'**
Example: &H3;&H18;&H0;&H0;&H1

Command Write **Byte list of frames separated by ';' , with , keywords 'HBYTE', 'LBYTE' if the value to write has the same size of a word, or 'BYTE' if the value corresponds to a byte.**
Example: &H10;&H18;&H0;&H0;&H1;&H2;HBYTE;LBYTE

Decode Read **Read decoding formula, where BYTE() is the array of bytes of the reply frame, starting from the address byte that coincides with BYTE(0).**
Example: BYTE(3)*256+BYTE(4) is the formula that returns the value read by the parameter.

Start Bit: **This is used only if the parameter type is equal to 9 (bit parameter). In this case, it identifies the first byte of the mask, starting from the least significant bit. This value ranges from 0 to 7.**
The default setting is 0.

Number of Bits: **It is related to the previous parameter and is used only if the parameter type is equal to 9 (bit parameter). In this case it returns the number of contiguous bits (1 - 8) used to represent the parameter.**

3.2 *Instructions for the use of Tool MakeDriverWizard*

After completing the Excel file with the **MakeDriverWizard** utility, the application generates an **.edr** driver file that has to be moved to the correct location by the user as specified above.
This program must be installed by the user by means of the Setup.
The program is easy to use: it is sufficient to select the Excel file with the driver that has to be loaded and press “Make” to convert it into an .edr file.
The .edr file is created in the installation folder of MakeDriverWizard, but can be moved by the user to the folder that contains the MODBUS drivers (see General notes paragraph).

4 Advanced programming

To fully understand the information provided below it is necessary to be familiar with programming techniques and with the Visual Basic Scripting programming language.

4.1 General

This specification explains how to create the application that enables you to connect the DAM/DTM communication system to a SmartAdapter" device, using a version compatible with the MODBUS/RTU protocol.
A SmartAdapter is a device generally installed in a standard network of Televis compatible devices. This device is used to support communications with devices that use the MODBUS/RTU protocol, but not the Televis protocol. In this sense, it acts as a "bridge", i.e. to establish a connection between two networks with different physical, electrical and logical characteristics.
The DAM/DTM communication system is designed to manage communications with devices that implement the Televis protocol. Communication with devices is always indirect. The DTM system communicates with an "interface" that manages the physical communications with the devices.
The Televis communication protocol is asymmetrical in terms of interface usage. To be able to send a message to a device connected to the physical network, it is necessary to encapsulate it in a message that can be sent to the interface. The device reply is returned to the DTM in its original format.
However, to be able to connect the SmartAdapter devices to the Televis network you necessarily have to use a double aggregation level.

- The first level encapsulates the MODBUS/RTU message in a Televis message.
- The second level encapsulates the Televis message in a message that can be read by the interface.

For the reasons explained above, reception uses one level of reception only because the operation of the interface is very transparent.
As the MODBUS/RTU protocol is not a native protocol of the DAM/DTM system and the devices usually connected to this network are not traditionally used in Televis environments, it has been necessary to develop a generic configurable driver, which enables the communication with the devices of the MODBUS/RTU network to be implemented by specific definition files. This allows end users of the DAM/DTM system to flexibly interface new MODBUS/RTU devices as needed without having to upgrade the versions of the applications they are using.
The addition of a new device to the DAM/DTM system (or of a class of devices like MODBUS/RTU) requires analyzing the following two aspects:

- The communication driver, i.e. the set of the methods that each driver should integrate for each device that has to be integrated in the Televis system.
- The management of parameters, i.e. the possibility of acquiring and/or editing the operating settings of the device.

The former activity applies mainly to the DTM component that hosts all the drivers of the devices and determines the actual integration of the new class of devices in the Televis system.

The latter activity applies both to the DAM and the DTM components, and controls the access to the internal parameters of the devices of the new class of MODBUS/RTU devices using a method that is similar to the one used for native Televis devices.

4.2 Communication driver

A device can be integrated in a Televis system only if the related communication driver implements a set of methods. Some of these methods may be "blank" (unavailable), i.e. may not be implemented if they are not considered relevant for the basic operation of the Televis system.
The section below lists the methods available for each driver:

- StartRvd The method must return an **unavailable command condition**.
- StopRvd The method must return an **unavailable command condition**.
- ReadDisplay The method must return an **unavailable command condition**.
- PressButton The method must return an **unavailable command condition**.
- ReleaseButton The method must return an **unavailable command condition**.
- KeyboardLock Local keyboard block request. If this mode is not available, the method must return an **unavailable command condition.**
- KeyboardUnLock Local keyboard unlock request. If this mode is not available, the method must return an **unavailable command condition.**
- Power Device start or shutdown request. If this mode is not available, the method must return an unavailable command condition**.**
- Light Lights On/Off request (for devices that control the lighting of equipment or plant areas). Device start or shutdown request. If this mode is not available, the method must return an unavailable command condition**.**
- **DeFrost** Defrost start request. If this mode is not available, the method must return an unavailable command condition**.**
- SyncClock Internal clock synchronization request. If this mode is not available, the method must return an unavailable command condition**.**
- GetInfo ID information request. On compatible Televis devices, this command is implemented in the base class of all devices. In this case (MODBUS/RTU) **this command must be implemented**.
- GetIoConfiguration Specifies the resources (digital inputs, analog inputs, digital outputs, analog outputs, statues and alarms) available on the device. **This command must be implemented**.
- ReadDigitalInputs Digital inputs read request. **This command must be implemented if the device makes at least one digital input (GetIoConfiguration) available**.
- ReadAnalogInputs Analog inputs read request. **This command must be implemented if the devices makes tat least one analog input (GetIoConfiguration) available**.
- ReadStatus Device status read request. **This method must be implemented if the device makes at least one status code available.** Status codes also include the statutes of digital outputs.
- ReadAlarms Alarms read request. **This method must be implemented if the device makes at least one error code available**.

- WriteDigitalOutputs Digital outputs read request. **This method must be implemented if the device makes at least one programmable digital output available.**
- **ReadParameter** Internal parameter read request. **This method must be implemented if internal parameters are not accessed through the standard read command of the Televis protocol. In this case (MODBUS/RTU) this method must be implemented**.
- **WriteParameter** Internal parameter write request. **This method must be implemented if internal parameters are not accessed through the standard write command of the Televis protocol. In this case (MODBUS/RTU) the method must be implemented**.

In the DTM component, each driver is implemented as special C++ class that inherits methods and properties from a base class (generic device) or from a special class (if there are limited differences as compared to other devices).
Each class is uniquely identified by means of a series of properties, generically called "identification data", which contain the following information:

- Address Address of the device on the network.
- **Firmware family or mask** Reference class of the device. This information is generally sufficient to identify the type of instrument.
- Firmware version This information determines the behavior of the driver, depending on the level of firmware implemented in the device. In extreme cases, it may require the development of a new specific driver.
- Model Model of the device within the class (firmware family or mask). This information is required for the RVD and Table features (management of parameters).
- **PCH code** Optional information linked to the parameters map.
- Release date Optional information that indicates the date of release of the firmware implemented in the device.

DTM creates for each device that is physically part of the network an instance of the special class of the device. The class is determined using the "identification data" provided by the program that uses the DTM. A network may comprise several devices of the same type and model. DTM creates, for each of these, a special class.
In ordinary operating conditions, DTM uses a vector of special instances to access all the devices needed for the external program.
The DTM has a specific command (network scan) that enables to locate all the Televis compatible devices that are part of the network. The device can be identified only if it implements the VER command of he Televis protocol. This command enables to integrally rebuild the identification data of the device and therefore to identify the driver class that needs to be used for communication purposes.
After the implementation of the MODBUS/RTU driver, the scan command is modified so that it is able to search both Televis compatible devices and MODBUS/RTU devices if:

- The enabling commands of the DAM/DTM acknowledges the use of the MODBUS/RTU protocol.
- At least one MODBUS/RTU configuration file is present.

The identification of these devices leads to the creation of a specific instance of the MODBUS/RTU driver for each specified configuration file (i.e. for each file present in a directory know to DAM/DTM). The DTM uses the GetInfo method of this instance to locate the identification data of the device and fully identify it.
As explained in the previous chapter, the MODBUS/RTU messages transmitted to the device must be encapsulated in Televis messages. The MODBUS/RTU creates the correct format for the command transmitted to the SmartAdapter, which extracts the MODBUS/RTU message and forwards it to the network.
To guarantee safety and uniqueness, it is essential to observe the following limitations.

- The SmartAdapter address within the Televis network must be fixed and always equivalent to 236 (ECh); there are no limitations as to the number of devices that can be simultaneously connected to a Televis network.
- The addresses of the devices in the MOBUS/RTU subnet should not overlap the addresses of the Televis network. This condition is essential to prevent collisions during transmission, which could occur if several SmartAdapter devices are present. The range used to search the address is specified in the configuration file of the MODBUS/RTU device and is generally limited to the addresses that have already been assigned to Televis devices. This search is performed by DTM.

The class related to the MODBUS devices may receive the file name as parameter of a specific method (if the network scan command is run during the creation of a generic instance) or search for the file starting from the identification data (when a specific instance based on known identification data is created). **If the file doesn't exist or is incorrect, all the driver methods are managed by the DTM component and will return the non implemented command condition.**
The name of the configuration file of the MODBUS/RTU protocol should follow this standard:

EXTPRT_7FFC_<*model code*>.EDR

Where:

- **EXTPRT (EXT**ernal **PR**o**T**ocol: protocol other than Televis**)** fixed string that enables to immediately identify the files required for the MODBUS/RTU protocol.
- **7FFC** **First identification code of the device.**
  This code, which should always be set to the hex value of 7FFC, represents the firmware family or mask of the device in the DAM/DTM configuration. For further information, see the section on the identification data of devices. This implies assuming that all the devices of the MODBUS/RTU class belong to the same family or firmware mask (hex **7FFC**).
- **<*model code*>:** **Second identification code of the device.**
  This code identifies the type of MODBUS/RTU device.
  This code, which can be a progressive number or a code chosen by the author of the driver, has no impact on the driver. This must be an hex number <= 7FFF (32767 decimal). If the Modbus unit is an Eliwell unit, it is advisable to use the following format for this code: low byte for the firmware mask, high and progressive byte to identify the model that corresponds to the mask (for example 01D6 stands for: D6 mask (214 decimal), model 1. This value is inserted in the **Firmware**

Version field of the identification data section; field **Model** and all the remaining fields are set to zero.

The Model Code is not regarded a unique code for the MODBUS/RTU devices at a DTM architecture level. This means that the same device may have different model codes if it is used in different installations.

- **EDR** identifies the unique extension assigned to all the configuration files of the external protocols managed by the DTM system.

DTM searches for the configuration files of the ModBus protocol using the following information that is usually contained in the DTMCFG.INI file:

[EXTPRT]
PATH="<dir-path>"

Where <dir-path> is the full name of the directory that contains the configuration files for the MODBUS protocol.

When you create a new instance of the MODBUS/RTU class for a specific device, you need to pre-process the definitions file associated to the device in order to create in the memory components able to accelerate the interpretation of the commands during the ordinary use of the driver.

The introduction of the MODBUS/RTU driver requires the insertion of two new methods in the DTM component for the management of extended commands.
These two methods are:

- GetExtCommands This method enables to determine the number and description of the extended commands available for the device.
- ExecuteExtCommand This method enables to run one of the extended commands made available by the device.

Extended commands allow the definition of a parameter (DWORD), which is used for the execution of the command. Although these commands are generally dichotomous (i.e. they enable or disable a function), it is also possible to use the content of a command for more complex applications.
The addition of these two commands does not cause compatibility problems with existing drivers. It is in fact sufficient to define these commands as non implemented in the base class of all the devices.

4.3 Management of MODBUS/RTU device parameters

The DAM component manages the read and write requests related to the parameters made available by the models linked to each Televis device. The structure used by the DAM component to manage the parameters is automatically created, for each new device, by a special program that takes into account the specifications of the device and the parameters maps provided in Excel files.
These templates contain a class that defines the model parameters and that links the model to the device class (using the identification data). The class contains a list of all the

parameters available for the device. The following elements are defined for each parameter:

- ID Index that uniquely identifies the parameter within the mode.
- **Tag** (name) String that identifies the standard name of the parameter.
- Um Code that uniquely identifies the unit of measurement associated to the parameters using the DAM/DTM conventions.
- Minimum value interpretation mode Specifies if the minimum value is defined directly or if it has to be acquired from the current value of another parameter.
- Minimum value Specifies the minimum value of the parameter or the index of another parameter with a current value that should be considered equivalent to a minimum value.
- Maximum value interpretation mode Specifies if the maximum value is defined directly or must be acquired from the current value of another parameter.
- Maximum value Specifies the maximum value of the parameter or the index of another parameter with a current value that should be considered equivalent to a maximum value.
- Default value Reference value for the parameter.
- Access mode Mode used to access the parameter for read and write operations. Current modes are equivalent to the typical commands of the Televis protocol.

The model definition, the list of parameters and the characteristics of parameters must be generated using the information in the definitions files for MODBUS/RTU. This operation is possible only if the enabling system used by DAM/DTM supports the MODBUS/RTU protocol. If this is not possible, there is no reason to generate information on the models of device parameters that cannot be managed.
At an operating level, when the DAM system identifies all the interfaces that can be used for communications (i.e. the existing physical networks), it determines also if the MODBUS/RTU support is enabled. If the support is enabled, it creates the parameter templates for the MODBUS/RTU devices that have a definitions file.
As explained above, the method used to access the parameters for read and write operations is managed centrally for Televis devices. In other words, the base class of all the devices contains generic methods that implement the standard read and write methods in the internal areas of the device memories (physical read/write operations, logical operations, etc.). As these methods cannot be used for MODBUS/RTU, it is necessary to use the parameter access methods defined in the special MODBUS/RTU class (ReadParameter / WriteParameter).

4.4 Script language

Windows Script Engine has been selected as environment for the script sections used to define and implement the drivers for MODBUS/RTU devices. WSE is supplied as standard in all Windows systems, starting from Windows 2000. For previous versions (Windows 98/Me, Nt4), it shall be necessary to install Internet Explorer 6.0 that includes WSE.
The WSE environment offers several different formal languages:

- VBScript (native)
- Jscript (native)

- Perl (external)
- Rexx (external)
- Lisp (external)

At present the MODBUS drivers will be implemented using the VBSCRIPT language, which enables single derivers to be preventively tested with Visual Basic 6.0. There are several differences between Visual Basic 6.0 and VBSCRIPT. However, for the purpose of the implementation of MODBUS drivers, the main difference lies in the unification of the type of variable that has to be implemented: VBSCRIPT offers one type of variable only, VARIANT, while Visual Basic 6.0 enables variables to be defined also for other types (int, single, double, etc.).

4.5 Structure of the MODBUS drivers for DTM

The DTM system requires that the file with the MODBUS driver contains a specific number of functions. As these functions are analyzed with the WSE engines, their syntax must be coherent with the specifications of the VBSCRIPT language.
The required functions are the following:

4.5.1 Function gProtocolType().

This function enables to identify the type of logical protocol that must be selected for the SmartAdapter.
The code returned by the function must correspond to the values specified in the following table:

Code	Protocol
01h	MODBUS

4.5.2 Function gAsyncInfo()

This function enables to identify the formatting parameters of the MODBUS network communications.
The code returned by the function must be built using the parameters specified in the following table:

Bit	Description	Acceptable values
bit 0-3	Baud rate	0 = 1200 baud 1 = 2400 baud 2 = 4800 baud 3 = 9600 baud 4 = 19200 baud 5 = 38400 baud (if present) 6 = 57600 baud (if present) 7 = 115200 baud (if present) n = Not admitted
bit 4	Disable	0 = Enable 1= Disable
bit 5-6	Parity	0 = STX odd and Even for the rest 1 = None 2 = Even 3 = Odd

4.5.3 Function gGetNextAddress(ByVal first)

This function enables to identify the next address of the MODBUS network that can be used to locate the device. If the **first** parameter is 1, the function must return the first useful address, otherwise the next address in the sequence.

4.5.4 Sub gSetNextStep(step)

This procedure enables to define the next step in the sequence of low level messages of the MODBUS network required to complete a macro message of the DTM protocol.

4.5.5 Function gGetTxMessage(index)

This function enables to recover the single bytes of the message that has to be transmitted through the MODBUS network. **index** is the index of the message in the [0..n-1] range, where (n-1) is the maximum length of the message that has to be transmitted.

4.5.6 Sub gPutRxMessage(index, value)

This procedure enables to transfer to VBSCRIPT the content of a message received as reply from the MODBUS network. **index** is the message index in the [0..n-1] range; **value** is the byte value received in the **index** position.

4.5.7 Sub gPutAddress(address)

This procedure enables to define the address (**address**) of the device that will be used to interact with the MODBUS network. All commands generated by the MODBUS driver refer to this address.

4.5.8 Function gGetAddress()

This function enables to recover the current address of the MODBUS device.

4.5.9 Function gEncodeStartRvd()

This function enables to generate the messages required to configure the RVD mode of the device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.10 Function gDecodeStartRvd()

This function enables to analyze the reply of the device to the previous message created by function gEncodeStartRvd(). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.11 Function gEncodeStopRvd()

This function enables to generate the messages required to disable the RVD mode of the device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.12 Function gDecodeStopRvd()

This function enables to analyze the reply of the device to the previous message created by function gEncodeStopRvd(). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.13 Function gEncodeReadDisplay()

This function enables to generate the messages required to read the display of the currently selected MODBUS device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.14 Function gDecodeReadDisplay()

This function enables to analyze the reply of the device to the previous message created by function gEncodeReadDisplay (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.15 Function gEncodePressButton(key)

This function enables to generate the messages required to simulate the selection of a key on the local keyboard of the currently selected MODBUS device. The code of the key used for the simulation (key) is specified in the standard commands of Televis devices.
The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.16 Function gDecodePressButton()

This function enables to analyze the reply of the device to the previous message created by function gEncodePressButton (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.17 Function gEncodeReleaseButton(key)

This function enables to generate the messages required to simulate the release of a key on the local keyboard of the currently selected MODBUS device. The code of the key used for the simulation (key) is specified in the standard commands of Televis devices.
The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.18 Function gDecodeReleaseButton()

This function enables to analyze the reply of the device to the previous message created by function gEncodeReleaseButton (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.19 Function gEncodeKeyboardLock()

This function enables to generate the messages required to lock the local keyboard of the currently selected MODBUS device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.20 Function gDecodeKeyboardLock()

This function enables to analyze the reply of the device to the previous message created by function gEncodeKeyboardLock (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.21 Function gEncodeKeyboardUnLock()

This function enables to generate the messages required to unlock the local keyboard of the currently selected MODBUS device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.22 Function gDecodeKeyboardUnLock()

This function enables to analyze the reply of the device to the previous message created by function gEncodeKeyboardUnLock (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.23 Function gEncodePower(active)

This function enables to generate the messages required to start (active = 1) or stop (active = 0) the currently selected MODBUS device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.

- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.24 Function gDecodePower()

This function enables to analyze the reply of the device to the previous message created by function gEncodePower (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.25 Function gEncodeLight(active)

This function enables to generate the messages required to start (active = 1) or stop (active = 0) the lights (generally of linked cells) of the current selected MODBUS device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.26 Function gDecodeLight()

This function enables to analyze the reply of the device to the previous message created by function gEncodeLight(). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.27 Function gEncodeDeFrost()

This function enables to generate the message required to run the defrost cycle on the currently selected MODBUS device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.28 Function gDecodeDeFrost()

This function enables to analyze the reply of the device to the previous message created by function gEncodeDeFrost (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.29 Function gEncodeSyncClock(sync_date)

This function enables to generate the messages required to configure the date/time on the currently selected MODBUS device; sync_date is the date in the DateTime format (format with floating point, in which the integer section represents the days while the fractional section represents the fractions of the day, starting from 31/12/1899). The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.30 Function gEncodeSyncClock ()

This function enables to analyze the reply of the device to the previous message created by function gEncodeSyncClock (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.31 Function gEncodeGetInfo()

This function enables to generate the messages required to identify the device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.32 Function gDecodeGetInfo()

This function enables to analyze the reply of the device to the previous message created by function gEncodeGetInfo (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.

- 0 : The reply has been correctly managed.

4.5.33 Function gEncodeGetIoConfiguration()

This function enables to generate the messages required to define the resources of the device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.34 Function gDecodeGetIoConfiguration()

This function enables to analyze the reply of the device to the previous message created by function gEncodeGetIoConfiguration (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.35 Function gEncodeReadDigitalInputs()

This function enables to generate the messages required to read the digital inputs of the device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.36 Function gEncodeReadDigitalInputs()

This function enables to analyze the reply of the device to the previous message created by function gEncodeGetIoConfiguration (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.37 Function gEncodeReadAnalogInputs()

This function enables to generate the messages required to read the analog inputs of the device. The function must return one of the following values:

- -2 : Internal driver error

- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.38 Function gDecodeReadAnalogInputs()

This function enables to analyze the reply of the device to the previous message created by function gEncodeReadAnalogInputs (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.39 Function gEncodeReadAlarms()

This function enables to generate the messages required to read the alarms of the device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.40 Function gDecodeReadAlarms()

This function enables to analyze the reply of the device to the previous message created by function gEncodeReadAlarms (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.41 Function gEncodeReadStates()

This function enables to generate the messages required to read the statuses of the device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.42 Function gDecodeReadStates()

This function enables to analyze the reply of the device to the previous message created by function gEncodeReadStates (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.43 Function gEncodeWriteDigitalOutputs(index, value)

This function enables to generate the messages required to write the **index** digital output (in the [0..n-1]) range with the **value** (0/1) value of the MODBUS device. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.44 Function gDecodeWriteDigitalOutputs()

This function enables to analyze the reply of the device to the previous message created by function gEncodeReadStates (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.45 Function gEncodeReadParameter(index, ByRef value)

This function enables to generate the messages required to read the **index** parameter of the device. If the operation is successfully completed, the **value** variable acquires the value of the read parameter. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.46 Function gDecodeReadParameter()

This function enables to analyze the reply of the device to the previous message created by function gEncodeReadParameter (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.

- 0 : The reply has been correctly managed.

4.5.47 Function gEncodeWriteParameter(index, value)

This function enables to generate the message required to write the **index** parameter of the device; **value** is the value that has to be written. The function must return one of the following values:

- -2 : Internal driver error
- -1 : Indicates that the command has not been implemented for the driver.
- 0 : Indicates that the generation of all the messages required to run the command has been completed.
- >0 : Specifies the length of the next message that has to be transmitted to the MODBUS device.

4.5.48 Function gDecodeWriteParameter()

This function enables to analyze the reply of the device to the previous message created by function gEncodeWriteParameter (). The function must return one of the following values:

- -1 : The returned reply contains a CRC error.
- 0 : The reply has been correctly managed.

4.5.49 Function gIdentified()

This function returns 0 if the MODBUS driver has not been identified by the driver; otherwise it returns 1.

4.5.50 Sub gGetNumOfResources(ByRef n_di, ByRef n_ai, ByRef n_al, ByRef n_st)

This procedure enables to recover the configuration of the device in terms of available resources (number of digital and analog inputs, number of alarms and number of statuses).

4.5.51 Sub gGetAiInfo(index, ByRef um, ByRef mul)

This procedure enables to recover the information related to the **index** analog input of the device; **um** is the ode of the unit of measurement defined in the DTM system; **mul** is the code of the multiplier that has to be used to standardize the read value in accordance with the definitions of the DTM system.

4.5.52 Function gGetDi(index)

This function enables to recover the value of the **index** digital input. The driver must save the value of digital inputs every time the device is explicitly read.

4.5.53 Function gGetAi(index)

This function enables to recover the value of the **index** analog input. The driver must save the value of analog inputs every time the device is explicitly read.

4.5.54 Function gGetAlarmCode(index)

This function enables to recover the **index** alarm code.

4.5.55 gGetAlarm

This function enables to recover the **index** alarm value. The driver must save the alarms value every time the device is explicitly read.

4.5.56 gGetStateCode

This function enables to recover the **index** status code.

4.5.57 gGetState

This function enables to recover the **index** status value. The driver must save the value of statuses every time the device is explicitly read.

4.5.58 gGetDisplayBuffer

This function enables to recover the content of the buffer that represents the device display map. The driver must save the buffer every time the device is explicitly read.

A MODBUS driver may contain utilities, which should preferably not be public (i.e. should not be accessible through the DTM Manager).
If one of the base functions of the driver is not present or the MODBUS driver contains a syntax error, this driver is not used for communication purposes.

4.6 Definition of the parameters of MODBUS devices

The parameters used for MODBUS devices are defined in the final area of the driver. The parameter manager of the DAM, for MODBUS devices, searches for specific keywords in the driver to be able to dynamically build matrices of parameters that are equivalent to those normally used by Televis devices.
The driver area used to define the parameters must meet the following requirements:

- The model name should be on a separate line with initial pattern:
  ‘DAMDTM_MODEL:
  Ex:
  'DAMDTM_MODEL:FC_BASICOM_MODBUS
  Where FC_BASICOM_MODBUS is the model name.

- The lines that define the parameters should start with:
  'DAMDTM_PAR
  This pattern should not be present in any part of the driver.

- The pattern should be followed by this information:

  - Numerical ID codes of the parameter
  - Parameter label
  - Code of the parameter unit of measurement (see Appendix, Table 3)
  - Indication of whether the lower limit is absolute (=1) or refers to another parameter (=2)
  - Minimum value of the parameter (or index of the parameter that defines it)
  - Indication of whether the upper limit is absolute (=1) or refers to another parameter (=2)
  - Maximum value of the parameter (or index of the parameter that defines it)
  - Default value of parameter

- Type of parameter (see Appendix, Table 4)
- Value multiplier (see Appendix, Table 5)
- Start bits   This is used only if the parameter type is 9 (bit parameter). In this case, it identifies the first byte of the mask, starting from the least significant bit. Values can range from 0 to 7. The default value is 0.
- Bit number   It is linked to the previous parameter and used only if the parameter type is 9 (bit parameter). In this case it returns the number of contiguous bits (1 - 8) used to represent the parameter.

Ex:

'DAMDTM_PAR:001,SP_C,0,1,100,1,500,150,2,-1,0,0
'DAMDTM_PAR:002,SP_H,0,1,100,1,500,150,2,-1,0,0

4.7 Example of MODBUS driver

The **EXTPRT_7FFC_01D6.EDR** file, available on the setup CD, contains an example of base driver, which can be used to communicate with Eliwell Fan Coil Basicom devices in MODBUS mode.

4.8 Appendix

Table 1: Multiplier codes for analog inputs

Warning: the read value is divided by **10^DECPOS_#**

DECPOS_1 = 0
DECPOS_10 = 1
DECPOS_100 = 2
DECPOS_1000 = 3
DECPOS_10000 = 4
DECPOS_INVALID = -1

Table 2: Codes of units of measurement for analog inputs

CELS = 0
FAR = 1
BAR = 2
RH = 3
VOID = 4
PA = 5
BIN = 6
PSI = 7
VOLT = 8
AMP = 9
HZ = 10
HOURS = 11
KWA = 12
KWR = 13
COS = 14
KWHA = 15
KWHR = 16
MINUTES = 17
SECONDS = 18
INVALID = -1

Table 3: Codes of units of measurement for parameters

CELS = 0
FAR = 1
BAR = 2
RH = 3
VOID = 4
PA = 5
BIN = 6
PSI = 7
VOLT = 8
AMP = 9
HZ = 10
PERC (%) = 11
SECONDS = 12
MINUTES = 13
HOURS = 14
CELS10 = 15
SECONDS10 = 16
BAR10 = 17
NVALID = -1

Table 4: Acceptable types for parameters

CHAR = 0

BYTE	**= 1**
SHORT	**= 2**
WORD	**= 3**
NOT IMPLEMENTED LONG	**= 4**
NOT IMPLEMENTED DWORD	**= 5**
NOT IMPLEMENTED FLOAT	**= 6**
NOT IMPLEMENTED DOUBLE	**= 7**
NOT IMPLEMENTED STRING	**= 8**
BITS	**= 9**

Table 5: Resolution codes for parameters

Warning: the read value is divided by **10^(-#)**

READ VALUE = 0
READ VALUE DIVIDED BY 10 = -1
READ VALUE DIVIDED BY 10 0 = -2
READ VALUE DIVIDED BY 10000 = -3

5 Observations and limitations

A – Analog inputs are signed sorted (-32768, +32767):
Larger values are not managed

B –The simplified driver does not implement:

- Global commands
- RVD
- Automatic detection of the configuration of the unit's resources

C –If the installation comprises one or more SmartAdapter modules, it is not possible to use the 14.12 Televis address for devices.

D –The addresses of MODBUS devices of subnets connected to SmartAdapter modules cannot overlap those used for devices with Televis protocol. This includes also reserved addresses.

6 Notes on the use of the driver with TelevisNet

A – The InstrumentDictionary table contains only one record for the external driver units. This record refers to the generic units family, i.e. *MicroFamily* **= 7FFC (32764 decimal)** and *ModelCode* = 0.

B - The specific name of the unit is displayed in the Driver Description field of the Excel file. However, as this is the unit description referred to the table view, the driver must contain at least one parameter.

C - Custom codes for alarms and statuses.
Users may create custom codes for alarms and statuses by entering the codes and related descriptions in the following files, which are located in the Database subfolder under the Televis**Net** installation folder.

- File UserStateCodes.ini - Codes from 1501 to 2000
- File UserAlarmCodes.ini - Codes from 501 to 999

Descriptions have to be entered in the language selected by the user. From the *Control Panel – Program* page of Telvis**Net**, users can transfer these codes and descriptions to the database of the program.

Eliwell & Controlli s.r.l.
Via dell'Industria, 15 Zona Industriale Paludi
32010 Pieve d'Alpago (BL) ITALY
Telephone +39 0437 986111
Facsimile +39 0437 989066
Internet http://www.eliwell.it

Technical Customer Support:
Email: techsuppeliwell@invensys.com
Telephone +39 0437 986300

Invensys Controls Europe
Part of the Invensys Group