P0435 Rules

Revision History

Version	Date	Author	Description
0.01	31-08-2009	V. Russo	First draft. Built on the previous P0435Rules.
0.02	01-12-2009	V. Russo	Added enough core information to make it “usable”.
0.03	22-02-2010	V. Russo	Added description of cross reference for high and low limit of parameters. Forgotten occurrences of “ParamManager” string were changed into “Params”. Added description of the usage of an empty cell in the EXCLUSION colum of the Client page.
0.04	14-04-2010	V. Russo	The TYPE_CODE column is now optional (content is computed according to DESCRIPTION_CODE and PROGRESSIVE). Added detailed description of the DESCRIPTION_CODE item and added CUS40000 special code. Added description about the meaning of a type code. Added support for IEEE754 values. Added support for address filtering and baud rate/parity enforcement
0.05	18-05-2010	V. Russo	Changed “MODBUS_COMMAND” definition for H04/H16 commands. Added a remark about the use of “PH” category for devices with vector tables. Added description of SEND and CREATEVALUE operators
0.06	07-06-2010	V. Russo	Added flow management to expressions. Added ON_ACQUISITION_START and ON_ACQUISTION_END events to the DataAcquisitionInit page.
0.07	04-07-2010	V. Russo	Added protocol enforcement feature for 3rd party devices. Added file name enforcement feature for 3rd party devices. Added support for SIGN+MANTISSA values. Changed letter for the H04 modbus command
0.08	24-08-2010	V. Russo	Added VIS() operator to ease management of RVD with different keyboards.
0.09	22-10-2010	V. Russo	Updated the rules about parameters. It’s now explained how to encode table of vectors.
0.10	15-11-2010	V. Russo	Added description of “raw” commands to Chapter 2.

1 Contents

This document defines a model for the P04354 document (simply called “P04354” in the resto f the document). The P04354 holds the information needed to acquire data from and interact with a given device using a communication channel (ModBus or Micronet).

1.1 Acronyms and Glossary

CPa	Client Page
DAIPa	DataAcquisitionInit Page
description code	An 8 digit (3 roman capital letters followed by 5 decimal numbers) that is used to identify an untranslated string. See 2.1.2.1.
LPa	Legenda Page
Micronet	Proprietary protocol used by Eliwell devices
Modbus	Standard protocol used by Eliwell and third parties devices
MPa	Models Page
NTLVC	New Televis Compact
P04354	Latest release of the P0435 document
PDPa	ParamsDefaults Page
PPa	Params Page
PPPa	ParamsPermission Page
RTCPa	RTC Page
RVDPa	RVD Page
TLV	Televis
WPa	Write Page
WAPa	WriteArgs Page

2 P04354 Structure

The P04354 holds the information needed to acquire data from and interact with a given device using a communication channel.

Information is organized in pages. Each page contains definitions about a given aspect of the monitoring software (e.g. the parameters or the resources).

2.1 Common columns

Some type of data are required by various pages (e.g. the memory address to poll to retrieved a given parameter or resource). To make the P04354 user friendly (at least a little bit), the columns used for the same purpose share the same name and syntax across the whole document.

2.1.1 Access Coordinates

Access coordinates are references to a memory location of a device and they are widely used within the P04354 document.

Access coordinates are kept into 7 columns:

Column	**Allowed Values**	**Description**
COMPLEMENTED	**TRUE**	The value is fixed point and signed
	FALSE or **-empty cell-**	The value is fixed point and unsigned
	IEEE754	The value is floating point according to IEEE754. The precision (single, double or quadruple) is given by the size of the binary data according to the content of the FILTER cell. Size of the data MUST be compatible with the desired precision (32 bits for single, 64 for double, 128 for quadruple).
	SIGN+MANTISSA	The value is representation is sign + mantissa, the MSb will be used as sign, the rest of the value as mantissa.
MICRONET_COMMAND	**L**	Logical reading/writing (H52/57). The value in **MICRONET_ADDRESS** is the logical area, the value in **MICRONET_SIZE_SUBADDRESS** is the index within the area.
	F	Physical reading/writing (H12/H13). The value in **MICRONET_ADDRESS** is the physical address, the value in **MICRONET_SIZE_SUBADDRESS** is the size of the memory area.
	E	**RFU**
	D	Record reading/writing (H16). The value in **MICRONET_ADDRESS** is the record row, the value in **MICRONET_SIZE_SUBADDRESS** is the index of the record within its row.
	M	DRV_OS or SC2 reading/writing (H7F,H52,H02/H7F,H52,H03). The value in **MICRONET_ADDRESS** is built using the macro-area id as MSB and the area id as LSB, the value in **MICRONET_SIZE_SUBADDRESS** is the index within the logical area.
	C	Custom command. (see next section)
MICRONET_ADDRESS	a number	The meaning changes according to the **MICRONET_COMMAND** value.
MICRONET_SIZE_SUBADDRESS	a number	The meaning changes according to the **MICRONET_COMMAND** value.
MODBUS_COMMAND	**S**	Modbus H03 is used for reading, H06 for writing.
	P	Modbus H03 is used for reading, H10 for writing.
	A	Modbus H04 is used for reading, nothing for writing.
	R	Modbus H01 is used for reading, H05 for writing.
	O	Modbus H01 is used for reading, H0F for writing.
	N	Modbus H02 is used for reading, nothing for writing.
	W	Modbus H04 is used for reading, H16 for writing.
	C	Custom command (see next section)
MODBUS_ADDRESS	a number	The modbus address.
FILTER	an hexadecimal number	The mask to be applied to the value (max 4 bytes).
	REV(hexadecimal number**)**	The bytes are reversed **<u>before</u>** applying the mask to the value

🟩	S[n]	The value is an array of n bytes. The mask applied to the bytes is by default HFF. This must be used for string values and the RVD buffer display.

The columns marked in green are mandatory.
The columns marked in yellow are mandatory but a default will be taken if left empty.
The columns marked in orange are mandatory for a micronet instrument.
The columns marked in blue are mandatory for a MODBUS instrument.

2.1.1.1 Use of Custom Commands

The MICRONET_COMMAND and MODBUS_COMMAND are used to state which is the command to use to read a given memory address.

Most of the commands are related to a memory address of registers. These commands have a well know format and it's enough to provide the value of the address in order to allow the supervisor software to gather the data. Moreover, many of these commands have aggregation rules (e.g. the H03 MODBUS command allows to read sequences of registers with just one memory access), so the supervisor software can optimize the number of required field accesses.

Some other times the commands have no standard structure and a custom request/reply sequence must be provided. It possible to handle these cases as "raw" commands. The structure of the request/reply sequence must be modelled using a simple script language that will be described in the following section.

To manage a field access as a RAW access:

- use the "C" letter in the MICRONET_COMMAND or MODBUS_COMMAND.
- Provide the scripts that model the request/reply in the MICRONET_ADDRESS or the MODBUS_ADDRESS cell. Up to two scripts can be provided and they must be separated by a semicolon. The first one is used for reading the value and the other one for writing it. It's also possible to provide just the reading sequence (which is the most common situation). The next table explains all the allowed cases:

Syntax	Description
Script1;Script2	Script1 is used for reading the value, Script2 for writing it
NONE;Script2	This states that there is no script for reading the value but there is a script for writing it. This is the typical case of a write operation.
Script1	If there's just one script, it is assumed that this is used for reading the value

- The binary size of the data passed/retrieved to/from the stream is computed evaluating the content of the "FILTER" column.

2.1.1.2 Raw Commands script details

The syntax of a single field access is the following

RAW({s(0),s(1),…,s(n)},{r(0),r(1),…,r(m)})

Where
s(i) is a series of descriptors for the stream to be sent.
r(i) is a series of descriptors for the stream to be received. If the received stream does not match this description it must be discarded.

Allowed descriptors are summarized in the following table.

Descriptor	**Description**	**Send**	**Receive**	**Function**
Hxx	constant number	Y	Y	A constant byte. “xx” is a number in hexadecimal notation
MTA	modbus target address	Y	Y	A byte containing the device address in modbus format
MCRC2	Modbus CRC	Y	Y	The CRC16 computed according to the modbus algorithm. This item spreads over 2 bytes.
XI(x)	ignore bytes	N	Y	A series of contiguous bytes that may be ignore while decoding the received bytes. “x” is the number of bytes.
IL	ignore length	N	Y	This means that all the bytes until the next descriptor are ignored while decoding the received bytes, whatever is their number. This can be used just once per stream and

				only if a beginning and an end of stream is described.
XD(x)	data extraction/injection	Y(write)	Y(read)	Extraction or injection of a byte of data. Byte is taken/placed from/to the byte in position “x” of the value.

2.1.1.3 *About the optimization of Raw Commands*

Even if it would be possible to write every micronet or modbus field access using a RAW field access, its use should be avoided when not necessary that is when there’s a standard protocol description available.

When using RAW field access, the system is not able to organize streams to be sent in order to minimize the network traffic. Indeed, the RAW scripts states how a stream must be encoded and decoded, but it does not say anything about how to merge different streams in order to retrieve the same information with a smaller number of network accesses.

Usage of standard protocol descriptions grants an higher level of optimization, because there are rules to optimize them.

2.1.2 Naming

This the information about the naming of the resource.

Naming is kept into 5 columns:

Column	Allowed Values	Description
INDEX	A number	An numeric index for the item. Indexes should start by one “1” and then incremented by one “1” each row.
LABEL	A string	A short description for the item. For items that should be displayed by a supervisor the length of the string should be kept as short as possible because a long string could lead to display issues. The label is also used to build cross references between items. Because of this, the label **must be unique** and it’s not allowed to share the same label between multiple items (e.g. it’s not allowed to use the same label for a parameter and its visibility, a different label must be used).
DESCRIPTION	A string	A friendly description for the item. The purpose of this value is to provide an easy way to identify the item for an human reader. This value should not be used by any supervisor.
DESCRIPTION_CODE	A description code	This value must be used by the supervisor to display the translated description of the item. See 2.1.2.1,2.1.2.2,2.1.2.3,2.1.2.4.
PROGRESSIVE	A number	The number will replace any {0} in the translated description of the item.
	A description code	The string code will be translated and its translation will replace any {0} in the translated description of the item.
	A string	The string will replace any {0} in the translated description of the item. Please note that this string **won’t be translated** so avoid string in Italian language (like “SU”,”GIU”,”ACCESO”) or using Italian notations (like “1°”). If the string must be translated use a string code instead.
GROUPS	String codes separated	If the item is member of a parameter/resource group add its string code to this

	by semicolon a ";"	cell. Multiple groups membership is supported by adding the group string codes separated by a semicolon ";". By default each item is added to the "ALL" (PGR00000) group, so there's no need to mark this membership.
	EMPTY	The item will be added to the "ALL" (PGR00000) group only.

2.1.2.1 Description codes

A description code is an unique code in the format

xxxyyyyy

composed by 3 ASCII characters (xxx) and 5 numeric characters (yyyyy). Descriptions of items in the P04354 must be provided as descritpion codes instead of explicit description.

The monitoring/supervisor software must keep a lookup table between the description codes and their translations in the desired languages. This allows to display the monitoring/supervisor pages in a given language simply adding the desired lookup table (also known as dictionary file).

E.g. the description code STA00001 will be translated into "Ingresso digitale" if the language is Italian, "Digital Input" if the language is English, "Entrée numérique" if the language is French.

2.1.2.2 Parametric Description Codes

Some description codes are related to strings that contain a variable part, e.g. the analog inputs of a device may be indexed with 1,2,… and their translation may change only for the index. These strings may be handled with a parametric description code which is a still an unique description code in the format

xxx4yyyy

composed by 3 ASCII characters (xxx), the "4" character and 4 numeric characters (yyyy).

The translated items linked with the parametric description code contains the so called placeholder, a "{0}" sequence which must be replaced with the parameter by the monitoring/supervisor software. While the description code is usually provided in the "DESCRIPTION CODE" cell, the parameter is provided in the "PROGRESSIVE" cell.

The parameter may be a number, a string or a (non parametric) description code itself.

E.g. It's possible to add a description for the 4 analog inputs of a device by using the INP40000 description code and the indexes 1,2,3,4. INP40000 in the English dictionary is translated into "Analog input {0}" and then the placeholder is replaced with 1,2,3,4.

Usage of strings and description codes as parameter must be done with care, because it can easily lead to translation errors. So it's recommended to use just numbers as parameter.

2.1.2.3 Reserved Description Codes

Some description codes are reserved and should not be used for descriptions. Those codes are summarized in the following table.

Description Code	Meaning
ALM00300	No Link
ALM00301	Instrument mismatch

2.1.2.4 The custom description code CUS40000

The custom description code CUS40000 is a special description code that is translated in any language with the string "{0}".

This allows to the user to provide language dependent translation for an item without updating the dictionary file. When this code is used in the "DESCRIPTION_CODE" cell, the desired string must be placed in the "PROGRESSIVE" cell.

Those items won't be translated if the monitoring/supervisor software language and the displayed string will be always the one provided in the "PROGRESSIVE" cell. Because of this it's recommended to use this special description code for custom or third parties drivers only and don't use them for standard Eliwell devices.

2.1.3 Representation

This is the information about the representation (how is it displayed by the supervisor) of an item.

Representation is kept into 2 columns:

Column	**Allowed Values**	**Description**
FORMAT	An expression	This expression should return a number x. If the item is not a floating point one, the numeric value of this item will be multiplied by 10^x before being displayed. If the item is a floating point one, this value affects the position of the decimal point only.
	EMPTY	If the item is not a floating point one, the value of this item will be displayed "as is". If the item is a floating point one, the value will be displayed with no fractional part.
MEASUREMENT UNIT	An expression	This expression should return a string code.
	EMPTY	By default the measurement unit "number" (VMU00020) will be taken.

2.2 The Legenda page (LPa)

This page holds information about the identification of the instrument and the P04354 document.

Row	**Allowed Values**	**Description**
Device Timeout	A number	The device communication timeout in milliseconds
Device buffer RX (bytes)	A number	The size of the device RX buffer.
Device buffer TX (bytes)	A number	The size of the device RX buffer.
Author	A string	The name of the P04354 author.
Date last update	A date in the (DD/MM/YYYY) format.	The date of the last update of the P04354 document.
Firmware Mask		

	CUSTOM(number)	If this is the P04354 of a 3rd party device (e.g. an energy meter) or a not standard device (an Eliwell device that for some reason does not reply to the standard Eliwell identification command, e.g. a programmable device like the XT Pro) a fictitious mask number must be assigned. Fictitious mask number must be unique, they will start from 20000 if they are developed outside Eliwell. The keyword **CUSTOM(…)** must be used and the fictitious mask number must be placed between the round brackets. The supervisor should not use the standard Eliwell detection rules with this device, custom detection rules for this kind of device must be provided in the DataAcquisitionInit page (DAIPa).
Firmware Version	A number	The number of the firmware version.
Document version	A number	The number of the current P04354 revision.

2.2.1 Extended Legenda Page (for third party or programmable devices only)

The Legenda page may be extended with 3 additional fields when the P04354 is used to generate a driver for a third party or non standard Eliwell device (e.g. a programmable device).

These fields are optional and a processing tool should not generate an error if they are absent or is they are blank.

Row	**Allowed Values**	**Description**
Network Address Filter	A range in the format "Low-High"	The device will be detected only if its address falls within this range.
	Comma separated specific addresses.	The device will be detected only if its address is one of the provided addresses.
	EMPTY or cell absent	Device will be detected whatever is its address. This is the default condition.
Baud Rate	A number	The baud rate to be used. This applies MODBUS instruments only and only when using a SmartAdapter.
	EMPTY or cell absent	The baud rate it's inherited by the default settings of the monitorig/supervisor software. This is the default condition.
Parity	E, O or N	The parity to be used. This applies MODBUS instruments only and only when using a SmartAdapter.
	EMPTY or cell absent	The parity it's inherited by the default settings of the monitorig/supervisor software. This is the default condition.
Protocol	MODBUS or MICRONET	The driver protocol will be set to MODBUS or MICRONET.
	EMPTY or cell absent	The driver protocol is selected by the compiling software. For a third party driver the default protocol is MODBUS.
Filename	A string ending with the ".bin" extension.	A custom filename for the driver. It must end with the "bin".

2.3 The Models page (MPa)

This page holds information about the device naming. A row must be filled for each model.

Column	**Allowed Values**	**Description**
POLI	A number	The Polycarbonate number for this model
MODEL_NAME	A string	The name of this model. This is the string that will be displayed by the supervisor as device name.
PARAM_MANAGER_NAME	A string	The name used by the param manager map when the protocol is Micronet
PARAM_MANAGER_NAME_MODBUS	A string	The name used by the param manager map when the protocol is Modbus.

2.4 The Params page (PPa)

This page holds information about the parameters of the device. The page is structured in rows, each rows containing the data about a given parameter.

For the

- Access Coordinates (see 2.1.1**).**
- Naming (see 2.1.2).
- Representation (see 2.1.3).

See the previous sections.

There is a restriction, valid only in the PPa, about the FORMAT field.

FORMAT	An expression	This expression should return a number x. The meaning is the same described in section 2.1.3. For the PPa page only there is a restriction about the variety of the expression that can be placed in this cell. Expression must be FIX(x) where x is a number.
	EMPTY	The value of this item will be displayed “as is”.

Besides this information, the PPa requires the following columns.

Column	**Allowed Values**	**Description**
		•
		•
		•
LOW_LIMIT	A number	The raw low limit. Information about the position of the decimal multiplieris not applied to the value (e.g. a -3.2C for a low limit of a set point, which tipically is a parameter with FORMAT equals to FIX(-1), will result in a -32 value in the LOW_LIMIT cell).
	Params(x)	A reference to a parameter that express the low limit.
HIGH_LIMIT	A number	The raw high limit. Information about the position of the decimal multiplieris not applied to the value (e.g. a 18.2C for an high limit of a set point, which tipically is a parameter with FORMAT equals to FIX(-1), will result in a 182 value in the LOW_LIMIT cell).
	Params(x)	A reference to a parameter that express the high limit.
R_W_RW	R	The parameter is read only
	W	The parameter is write only
	RW	It is possible to read and write the parameter

2.4.1 About the uniqueness of the label

Labels in the P04354 must be unique. This need arises from the fact that the label is used as key to make cross references to parameters from wherever in the P04354, using the Params keyword and the label of the parameter.

2.5 The Client page (CPa)

This page holds information about the resources and the network commands of a device. The page is structured in rows, each rows containing the data about a given resource/network command.

For the

- Access Coordinates (applies to Resources only, see 2.1.1**).**
- Naming (applies to all , see 2.1.2).
- Representation (OPTIONS applies to resources only , see 2.1.3).

See the previous sections.

As you will read in the following table, access coordinates are not mandatory here. They are used for resources only and they are mandatory only if no value condition is provided. They are mandatory if there is no value condition or if a reference to them is made somewhere in the P04354 document.

Besides this information, the CPa requires the following columns.

Column	**Applies To**	**Allowed Values**	**Description**
EXISTENCE	All	An expression	If the give expression returns a value > 0, then the resource is visible.
TYPE	All	AI	An analog input resource
		DI	A digital input resource.
		Alarm	Am alarm. The resource value will be cast to e Boolean by the supervisor, true if the value is > 0, false if = 0. It's a good practice to make the resource value 1 if a true is desired, 0 if a false is desired.
		Status	A status
		Net Command	A Network Command.
		RTCCommand	The RTC command (this command follows special rules).
		Support	A support resource. This kind of resource will not be displayed by the supervisor but it's needed because somewhere there's a reference to it.
VALUE_CONDITION	Resources	An expression	If an expression is provided, the supervisor will use the value of the expression instead of the access coordinates when the resource value is computed. Access coordinates can be provided anyway, because they may be used as a reference.
TEST_CONDITION	Resources	An expression or a reference	It tells if the current resource is in error state or not. The supervisor will flag the resource as "error" if the output of the expression/reference is different from 0. This may contain: • A reference to another resource. If a resource Y is the error condition for the resource X, the TEST_CONDITION cell of the X row must contain Client(Y). This means that the resource value is used, so if a value condition is provided for Y, its output is considered and format information is applied.

			• An expression. The most common one is UNSIG(THIS())=FIX(H8000) that can be used for analog resources when the error condition is stated by the (non) value H8000.
WRITE_SEQUENCE	Net Command	A sequence of items defined in the WPa separated by a semicolon ";"	A sequence of items defined in the WPa. When the network command is executed, the write items are excuted in the same sequence reported here. Note that this does not apply to the RTCCommand that follows special rules. In this case the cell must be left empty.
			•

2.6 The Write page (WPa)

This page holds information about the atomic write operations used by network commands and the RTC synchronization command. These items are used as bricks to build a complete a Net Command or a RTC WRITE_SEQUENCE sequence.

For the

- Access Coordinates (see 2.1.1).
- Naming (only INDEX and name applies to this, see 2.1.2).

See the previous sections.

Note that the value in the NAME column is used as an identifier for the write item, so the rule is stricter here, NAME value must be unique within the WPa.

Besides this information, the WPa requires the following columns.

Column	Allowed Values	Description
VALUE	A value or e reference to a WriteArg.	The value to be written at the given access coordinate. Values in this page are always unsigned and integer.

2.8 The RTC page (RTCPa)

This page holds the information about the atomic writing operations used by the RTC synchronization command.

For the

- Access Coordinates (see 2.1.1).
- Naming (only INDEX applies to this, for the NAME column see below, see 2.1.2).

See the previous sections.

Besides this information, the RTCPa requires the following columns.

Column	Allowed Values	Description
NAME	InitSequence	This is the sequence of WPa items needed to start the RTC synchronization. Sequence must be provided in the WRITE_SEQUENCE cell. No access coordinates are required fo this row.
	EndSequence	This is the sequence of WPa items needed to start the RTC synchronization. Sequence must be provided in the WRITE_SEQUENCE cell. No access coordinates are required fo this row.
	Second	The coordinate of the second value.
	Minute	The coordinate of the minute value.
	Hour	The coordinate of the hour value.
	DayOfWeek	The coordinate of the day of week value.
	Day	The coordinate of the day of month value.
	Month	The coordinate of the month value.
	Year	The coordinate of the year value (in the two digit notation, e.g. 07 for 2007)
WRITE_SEQUENCE	A sequence of items defined in the WPa separated by a semicolon ";"	A sequence of items defined in the WPa. When the init/end is executed, the write items are executed in the same sequence reported here.

2.10 The ParamsDefaults page(PDPa)

This page holds the information about the default values of parameters for each model. This page is mandatory for new designs, the data stored here are used by the supervisor to handle the parameters of a given instrument (to handle the default values and to compute the visibilities).

The PDPa will be built with the following columns:

Column	Allowed Values	Description
INDEX	A number	Index of parameter
LABEL	A string	Label of parameter (for debug purpose only)
Model code + Protocol identifier	Number or Strings	After the first two columns there are as many columns as the number of different combination of supported model code/policarbonates and protocols. Column header has the following syntax XXXX Where XXXX is the model code/polycarbonate. In the cell below the model code/polycarbonate the identifier of the protocol must be provided and it should be: MICRONET or MODBUS The rest of the column must be filled with the default values of the parameters in the old param manager format, so values are • If numeric: always integers (if the parameter value could have a fractional part it must be multiplied by the proper power of ten. This is true even if the value itself has no fractional part, e.g. 2.0 han no fractional part but it must be written as 20) and signed. • If string: they are simply strings.

Here's an example of PDPa (an extract from the P0435 sheet for mask 140).

INDEX	LABEL	1025	1026	1027	1028	1029	1030

		MICRONET	MICRONET	MICRONET	MICRONET	MICRONET	MICRONET
1	**Set**	320	0	0	-24	0	0
2	**diF**	60	2	2	3	2	2
3	**HSE**	400	99	99	-10	99	99
4	**LSE**	320	-50	-50	-24	-50	-50
5	**OSP**	0	0	0	0	0	0
6	**HC**	1	0	0	0	0	0
7	**Cit**	0	0	0	0	0	0
8	**CAt**	0	0	0	0	0	0
9	**dOd**	0	0	0	0	0	0
10	**dAd**	0	0	0	0	0	0

2.12 DataAcquisitionInit (DAIPa)

This page can be used to enable special of future features of a acquisition module. As default this page **should be left blank**.

Each row of the page contains information about a special/future feature.

Column	**Allowed Values**	**Description**
FUNCTION	IDENTIFICATION	An alternate identification method. Use this when the standard micronet mask and poly command are not supported (e.g. a third party device) or are not enough to identify the device (e.g. an XTPro application).
	PRE_AUTOACQUISITION_DISCOVERY	An optional sequence of operations that may be executed before than retrieving the resources list from an instrument. Global variables may be computed here and then used in the CPa to get simplier expressions (e.g. mask 004)
	PRE_RVD_DISCOVERY	An optional sequence of operations that may be executed before retrieving the information about the RVD. As for the CPa, global variables may be computed here and then used in the RVDPa to get simplier expressions (e.g. mask 004).
	PRE_PARAM_WRITE	A sequence of commands that is executed before each parameter/parameter group writing (e.g. unlock cold parameters)
	POST_PARAM_WRITE	A sequence of commands that is executed after each parameter/parameter group writing (e.g. lock cold parameters).
	ON_AUTOACQUISITION_START	A program that is executed each time the auto acquisition process is started (e.g. turn on the EWTV280).
	ON_AUTOACQUISITION_STOP	A program that is executed each time the auto acquisition process is stopped (e.g. turn off the EWTV280).
PROTOCOL	MICRONET	This feature is valid only if the protocol is micronet.
	MODBUS	This feature is valid only if the protocol is modbus.
	ANY	This feature is valid for both protocols.
0,1,2,3,...,n	An expression	A sequence of expressions, one for each cell, starting from the cell 0 and lasting as many cells as are needed. The default page has a maximum of 11 steps, but this limit can be exceeded, the parser of the cells will/must stop at the first empty cell. Expression must can be built using the explicit field access operators (see 3.2.2) if a PROTOCOL is given (should not be any), even if it's not mandatory and not recommended.

2.12.1 An Alternate identification example, extension of micronet commands.

The Energy XT Pro is a programmable device. This means that its resources and parameters are not known a priori, because they depend on the application that is uploaded.

As a common practice, the application builder stores a value in a MODBUS (or micronet) register to identify the application. So the device and application couple can be identified without shadow of doubt reading the firmware version (provided by the standard Eliwell detection rule) and this value.

Thus, standard identification rules must be integrated with this additional register reading. This can be accomplished using the DAIPa.

In the following table there is an example about the DAIPa of an Energy XT Pro application.

FUNCTION	PROTOCOL	0	1
IDENTIFICATION	MODBUS	SET(MODELCODE)=T(H0311,2,HFFFF)	INT(0,{1150=1},GET(MODELCODE))

This application may be identified by reading the MODBUS register H0311 (se cell 0). If the value in the register is 1150 (cell 1) then the application is the desired application.

When using the IDENTIFICATION function, the return value of the last item of the sequence is used as “existence” condition, so if a 0 is returned the identification is not successful, and it is successful otherwise.

The result of this identification step is then merged with the standard micronet mask identification, that it is still active because a “Firmware mask” number was provided in the LPa, so both conditions must be true to get a successful identification. To completely disable the default micronet mask identification see the next example about the identification of a third party device.

Notice the use of the MODELCODE keyword. This means that the stored value will be used in computation where the model code is required (e.g. the lookup table in the MPa or each time a MOD() is used in an expression).

If required, mask (FAMILYCODE), revision (REVISIONCODE), model (MODELCODE) and polycarbonate (POLYCODE) codes can be overridden in computations using the proper keyword (see the keyword between brackets).

2.12.2 An Alternate identification example, third party devices.

The PM710 is a third party energy meter, that supports the modbus protocol on a RS485 bus. It does not support the standard micronet mask and poly identification, so in the LPa, where the mask number is expected the following must be written:

Firmware Mask	**CUSTOM(10000)**	Mskxx

This means that this devices features a custom/third party identification method. The number between brackets is a fictional mask number assigned to this 3rd party device. Fictional mask numbers created outside Eliwell should start from 20000 and must be unique.

The identification method is expected to be in the DAIPa and if successful, the fictional mask number will be assigned to the device.

FUNCTION	PROTOCOL	0	1
IDENTIFICATION	MODBUS	SET(MODELCODE)=T(7003,2,HFFFF)	INT(0,{1=15165},GET(MODELCODE))

This application may be identified by reading the MODBUS register 7003 (se cell 0). If the value in the register is 15165 (cell 1) then the application is the desired application.

3 Expressions

Expressions are the core of a script language used to describe the dynamic behaviour of a device driver. This allows to:

- Make the number of resources, their measurement unit and their format dynamic in function of parameters or other resources.
- Create virtual resources by combination of other resources.
- Set up a custom detection rule.

3.1 *Concept of program and workspace*

The main usage of an expressions is as “one-shot” expression, it is evaluated and its value returned as the requested value.

In some parts of the P04354 (DAIPa, RVDPa) the concept of program may be used. Expressions can be chained together making a program.

A workspace with the variables used is associated to the program.

Some variables are defined by default such as the model code (accessed by the MOD() operator) or the XD (data extraction variable). The default variables are always accessible, even if the expression is outside a program.

To write a program, expression can be written in sequence and divided by a semicolon “;”.

3.2 *List of Operators*

Operators can be divided into the following categories:

- Reference access operators.
- Explicit Field access operators.
- Value operators.
- Complex operators.
- Logical operators.
- Sub expression operators.

3.2.1 Reference field access operators

The field access operators concerns receiving and sending data to the devices.

Reference field access operators have been introduced with the release of P04354. With the adoption of these operators, expressions are placed at an higher abstraction level and are purged by explicit field access details.

When a reference field access operator is encountered within an expression, the expression solver will use the field access coordinates of the referred even. Field access coordinates of an item for the micronet protocol are the values defined in the following columns:

- MICRONET_COMMAND

- MICRONET_ADDRESS
- MICRONET_SIZE_SUBADDRESS
- FILTER

While for the Modbus protocol are the values defined in the following columns:

- MODBUS_COMMAND
- MODBUS_ADDRESS
- FILTER

Supported operators are:

• THIS	Self reference
• Params	Reference to Parameter Access Coordinates
• Client	Reference to Client Access Coordinates

3.2.1.1 Self Reference – THIS

This is the self reference operator. When this operator is encountered, the access coordinates of the item are used. The syntax is the following:

THIS()

3.2.1.2 Reference to Parameter Access Coordinates – Params

This is the operator for the reference to a parameter. When this operator is encountered, the access coordinates of the referenced parameter are used. The syntax is the following:

Params(label)

Where:

- label is the label identifier of the parameter.

3.2.1.3 Reference to Client Access Coordinates – Client

This is the operator for the reference to a client. When this operator is encountered, the access coordinates of the referenced client are used. The syntax is the following:

Client(label)

Where:

- label is the label identifier of the Client.

3.2.2 Explicit Field access operators

The field access operators concerns receiving and sending datas to the devices. They differ from the reference field operators because the access coordinates are explicitly declared as arguments of the operator.

Due to this reason, **they are deprecated** in P04354. They are reported in this document as an help to extract information from older documents and for special uses in the P04354, such as the 2.12 DataAcquisitionInit (DAIPa).

The following operators are supported:

• F	Physical read
• L	Logical read
• D	Logical DRV_OS read
• T	Modbus 0x03 read

3.2.2.1 Physical read – F

The physical read operator has the following syntax:

F(addressHx,byteCount,bitmask)

Where:

- addressHx is the hexadecimal memory address.
- byteCount is the decimal number of bytes to be read.
- Bitmask is the hexadecimal data mask.

An example of physical read is F(H0012,2,HFFFF).

3.2.2.2 Logical Read

The logical read operator has the following syntax:

L(area,offset,bitmask)

Where:

- Area is the hexadecimal index of the area.
- Offset is the hexadecimal offset in the area.
- Bitmask is the hexadecimal data mask.

An example of logical read is L(H12,H00,HFFFF).

3.2.2.3 Logical DRV_OS read

The logical DRV_OS read operator has the following syntax:

D(address,byteCount,bitmask)

Where:

- Address is a six digit hexadecimal number, built by the concatenation of area, multiarea and element index.
- byteCount is the length of the data, which is by default 2 (string reading is not supported).

- Bitmask is the hexadecimal data mask.

3.2.2.4 *Modbus 0x03 read – T*

The modbus 0x03 read operator has the following syntax:

T(addressHx,byteCount,bitmask)

Where:

- addressHx is the hexadecimal address of the register.
- byteCount is the size of the read data in bytes (since modbus registers are 16 bits words, this is an even number).
- Bitmask is the hexadecimal data mask.

An example of modbus read is T(H0012,2,HFFFF).

3.2.2.8 *About the data format*

Here are explained additional rules used in the interpretation of the data within the operators:

- An hexadecimal number must be written in the format Hx, where x is an hexadecimal number written using upper cases only.
- A mask value must be an hexadecimal number and contain only contiguous bits. Mask with “holes” (a zero bit between two one bits) are not allowed.

3.2.3 Access Modifiers

These operators may be applied to an access coordinate (both reference or explicit) to modify how a value is managed by the expressions solver. Supported operators are:

• SIG	Force sign modifier.
• UNSIG	Force unsign modifier.
• REV	Little endiand/Big endian modifier.

For these operators the syntax is

SIG(A) or UNSIG(A) or REV(A)

Where A is an access coordinate. The resulting expression is still an access coordinate (so another access modifier may be applied to it, creating a chain of access modifiers).

3.2.3.1 *Force Sign – SIG*

The value obtained from the given access coordinate is considered as a signed value (even if the original access coordinate is unsigned).

3.2.3.2 *Force Unsign – UNSIG*

The value obtained from the given access coordinate is considered as unsigned (even if the original access coordinate is signed).

3.2.3.3 Reverse endianness – REV

The bytes obtained from the given access coordinate will be reversed. It can be used to change a value from big endian to little endian or vice versa.

3.2.3.4 Chains of SIG and REV

SIG and REV are modifiers that can be applied together to a reading operator. SIG(REV(…)) states that the bytes must be reversed and the value is signed.

An example of access modifier use is SIG(REV(THIS())).

NOTE: To avoid misunderstanding about the correct evaluation of the SIG and REV operands (it makes difference if they are applied before or after the mask operation), they can be used only with a "full mask", that is a mask with all bits set to one. It's not possible to modify the SIG and REV syntax to a much clear version since it's widely used in actual P0435 documents.

3.2.4 Value operators

Value operators are used to put a value inside an expression. The following operators are supported:

• FIX	Constant value.
• CONST	Constant value (alias).
• SET	Sets a variable in the workspace.
• GET	Retrieve a variable from the workspace.
• CREATEVALUE	Creates a value merging a set of access coordinates
• MOD	Retrieve the model code from the workspace.

3.2.4.1 Constant Value – FIX

FIX puts a constant value inside an expression. The correct syntax is

FIX(value)

Where:

- Value can be a number or a string.

An example of fix usage are FIX(31) and FIX(VMU00001).

NOTE: Use of a string as a constant value is allowed only during the evaluation of a measurement unit. Use of constant strings within a complex expression leads to impredictable results and must be avoided.

3.2.4.2 Constant Value – CONST

This is an alias for the FIX operand. CONST puts a constant value inside an expression. The correct syntax is

CONST(value)

3.2.4.3 Set variable – SET

Sets a variable with the given name in the workspace. The syntax is

SET(variable_name)=expression

Where:

- Variable_name is the name of a variable in the workspace.

When “expression” is evaluated, its value is stored in the workspace memory as a variable with the “variable_name” identifier. If no variable with this name exist it is created, otherwise the existing variable is overwritten. After that, the value may be retrieved with the GET operator.

An example of Set usage is SET(MEASUREMENT_UNIT)=Params(dro).

3.2.4.4 Retrieve variable – GET

Gets the variable with the given name from the workspace. The syntax is

GET(variable_name)

Where:

- Variable_name is the name of a variable in the workspace.

When the GET is executed the value of “variable_name” is retrieved from the workspace. If no value with the given identifier exists then the expression is considered faulty an it’s not evaluated.

An example of Get usage is
INT(VMU00000,{0=VMU00000,1=VMU00001},GET(MEASUREMENT_UNIT)).

3.2.4.5 Create Value – CREATEVALUE

Builds a value extracting bits from the device using a set of access coordinates and appending the result together. The syntax is

CREATEVALUE(coords1, coords2,…,coordsn)

Where

- Coordsx is an access coordinate.

Bits are appended in a little endian way, where the bits extracted using coords1 are the less significant bits, while the bits extracted using coordsn are the most significant bits.

3.2.4.6 Model code – MOD

Retrieves information about the instrument model from the workspace. The value is retrieved a stored during the identification step. If no custom identification rule is provided the standard Eliwell model code is returned, otherwise the value set as MODEL_CODE in the custom detection rule is returned.

The syntax is

MOD()

NOTE: The data stored in the workspace are evaluated at the instrument identification step, no field access in involved when using these operators.

3.2.5 Complex operators

Complex operators perform complex operations on the operand. Complex operators are

INT	The lookup operator
LOOKUP	The lookup operator (alias)

3.2.5.1 Look-up operator – INT

The look-up operator performs a look-up operation. The syntax is

INT(defaultValue,{key1=value1,…,keyn=valuen},expression)

3.2.5.1.1.1.1 Where

- defaultValue is the default value to be returned if no one of the keys matches with the evaluated expression. Please note that the expression must have a value anyway. If the expression has no value (e.g. a communication error), no value is returned.
- Keyx=valuex is a set of look-up values. If the evaluate expression is equal to one of the key values, then the associated value is returned as result of the lookup operation.
- Expression is an expression with a return value.

An example of INT usage are INT(0,{17=1},MOD()) and !INT({0,-4,4,-5,5} , SIG(REV(F(H101C, 2, HFFFF)))).

3.2.5.2 Look-up operator – LOOKUP

This is an alias for the INT operand. The LOOKUP operator performs a look-up operation. The syntax is

LOOKUP(defaultValue,{key1=value1,…,keyn=valuen},expression)

3.2.6 Logical operators

These are logical operation on operands. Supported operators are:

And	“&”
Or	“|”
Xor	“^”
Equals	“=”
Greater than	“>”

• Less than	“<”
• Not	“!”
• Negate	“~”

3.2.7 Arithmetical operators

These are the simplest arithmetical operators. Supported operators are:

• Addition	“+”
• Subtraction	“-“
• Multiplication	“*”
• Division	“/”

3.2.8 Arithmetical Cosiderations

A few words must be spent on how the values are handled when evaluating an expression.

3.2.8.1 About the representation of numbers

Numbers within expressions are always integers (signed or unsigned). Size may vary, but only numbers up to 32 bits are allowed, expressions handling numbers with size bigger than 32 bits may lead to unpredictable results.

Unless otherwise specified, a number is a 32 bits signed integer.

Wherever it is needed, numbers will be extended to 32 bits.

Fractional numbers are not allowed. It’s still possible to represent a fractional number to an user, by handling its FORMAT (see 2.1.3), but all the calculations involving this number before being shown to the user are done on integers.

This choice was done to prevent misunderstandings and ambiguity with the use of expressions, such as misplacement of the fractional point or wrong assumptions about the size of the operands.

3.2.8.2 About the output of an arithmetical or a logical operation

The output of an arithmetical or logical operation is always a 32 bits signed integer.

For a logical operator the output will be a 32 bits signed one if the operation is TRUE and a 32 bits signed zero if the operation is FALSE.

3.2.8.3 About the “Equals” operator

The equals operator will work only of operands of the same type (numbers with number, booleans with booleans, strings with strings and time with time). Comparison between items of different nature will always return a FALSE.

Comparisons are done by value, regardless of the binary representation of the value. So:

- Size does not matter when comparing numbers. E.g. H00000001 (size is 32 bits) is equal to H01 (size is 32 bits).

- Sign does matter when comparing numbers, so HFFFF signed is NOT equal to HFFFF unsigned.
- Binary representation does not matter when comparing Booleans. H02 is equal to H01 if the oprands are Boolean.
- String representation does not matter. Strings will be converted to UNICODE (according to the proper conversion rule) and then compared.
- String codes won't be translated.
- Times will be converted to a full time object before being compared. This means that when evaluating times the missing data (e.g the year) will be filled with the current time data.

With very few exceptions, comparisons in expressions are about comparisons between numbers, so keep well in mind the first two points.

3.2.8.4 About the "Greater Than" and "Lesser Than" operators

As the Equals operator (see 3.2.8.3) the Greater Than and Lesser Than operators works only on operands of the same type. Comparison between items of different nature will always return a FALSE and in this case they have a less intuitive meaning.

Comparison between values is done by value and does not apply to strings.

3.2.8.5 About the division by zero

A division by zero is managed as an expression error. If a division by zero occurs, then the whole expression is judged as "not evaluable". This can lead to malfunctioning such as undetectable instruments or no link errors. So division operator must be use with care.

3.2.9 Sub expression operator

It's possible to define a sub expression writing it within round brackets "(" and ")". Sub expression usage has some limitations. A sub expression can't be used

- As argument of a field access operator.
- As default value of a lookup operation.
- As member of a key,value pair in a lookup operator.