The Universal parser plugin
PRINTED MANUAL

Universal parser plugin

Printed: 7/17/2019

Publisher

AGG Software

Production

©2018-2019 AGG Software
http://www.aggsoft.com

Table of Contents

1 Introduction

Based on this module, you can easily create a parser for data analyzing and handling. First of all, the module is intended for handling text data, but it can also process simple binary packets. The Universal parser module allows you to create a more complex parser as compared to ASCII Data Parser module. And it does not require knowledge of programming languages, unlike "Python plugin" or "Proxy plugin" modules.

Creating a parser consists of writing an XML file using a simple text editor, which allows specifying data processing rules. When the module is initiated, it reads the XML file and runs according to the rules.

After installation, in the module folder, you can find several examples of XML files.

2 System requirements

The following requirements must be met for "Universal parser" to be installed:

Operating system: Windows 2000 SP4 and above, including both x86 and x64 workstations and servers. A latest service pack for the corresponding OS is required.
Free disk space: Not less than 5 MB of free disk space is recommended.
Special access requirements: You should log on as a user with Administrator rights in order to install this module.

The main application (core) must be installed, for example, Advanced Serial Data Logger.

Notes for Microsoft Vista and above:

Since our software saves data to the registry and installs to the Program Files folder, the following requirements must be met:

1. You need Administrator rights to run and install our software
2. The shortcut icon of our software will be located on the desktop;
3. Windows Vista will ask for your confirmation to continue the installation.

NOTE: You can configure the user account only once in order not to see the above dialog box any more. Search Google for the solution of this problem.

3 Installing Universal parser

1. Close the main application (for example, Advanced Serial Data Logger) if it is running;
2. Copy the program to your hard drive;
3. Run the module installation file with a double click on the file name in Windows Explorer;
4. Follow the instructions of the installation software. Usually, it is enough just to click the "Next" button several times;

5. Start the main application. The name of the module will appear on the "Modules" tab of the "Settings" window if it is successfully installed.

If the module is compatible with the program, its name and version will be displayed in the module list. You can see examples of installed modules on fig.1-2. Some types of modules require additional configuration. To do it, just select a module from the list and click the "Setup" button next to the list. The configuration of the module is described below.

You can see some types of modules on the "Log file" tab. To configure such a module, you should select it from the "File type" list and click the "Advanced" button.

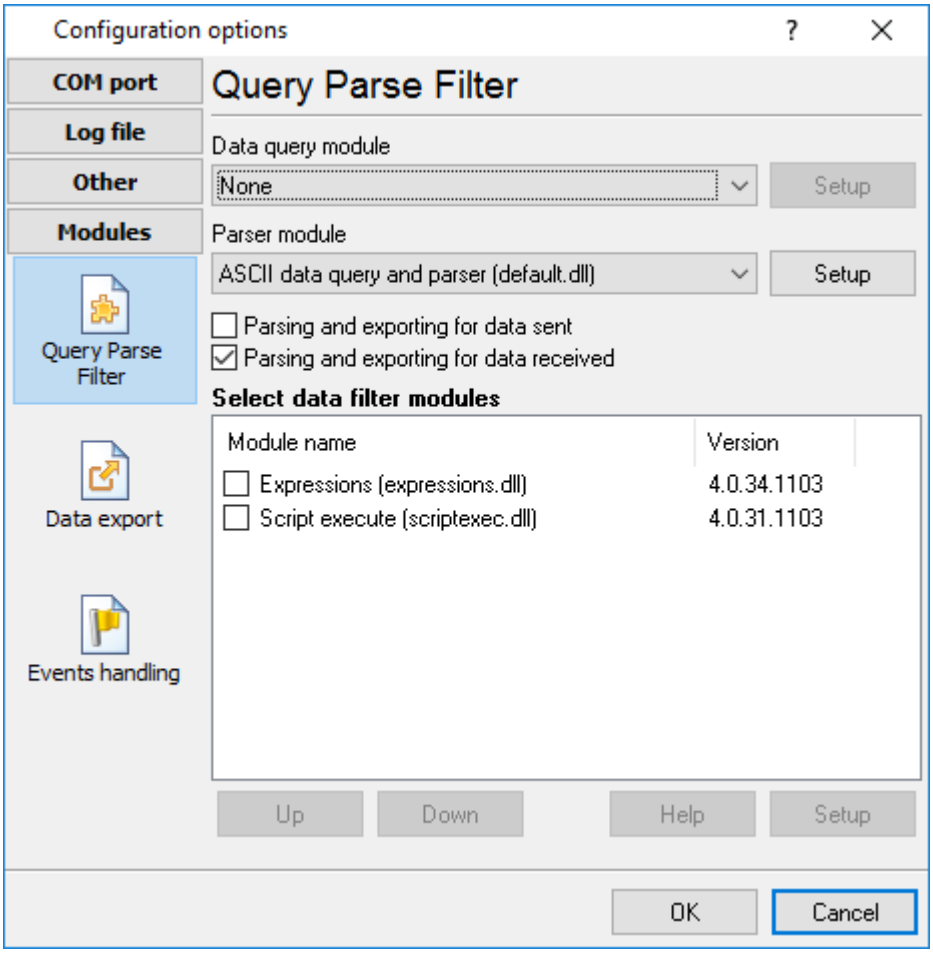

Fig.1. Examples of installed modules

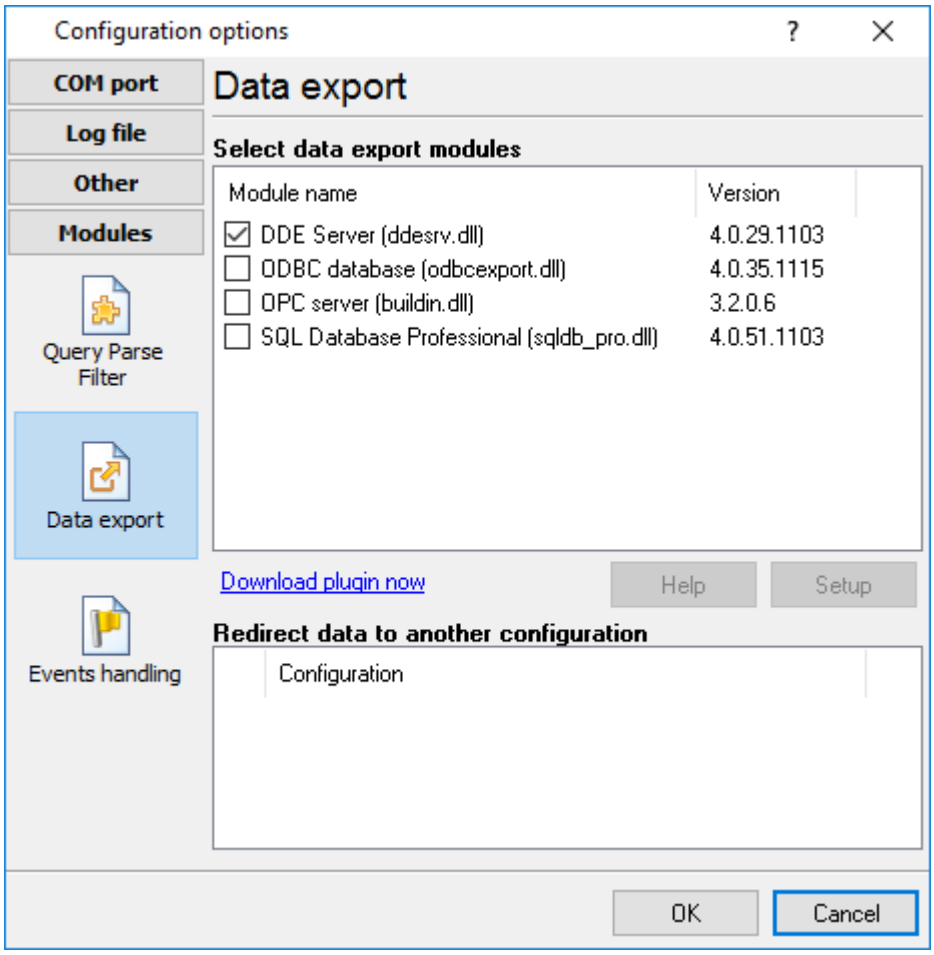

Fig.2. Examples of installed modules

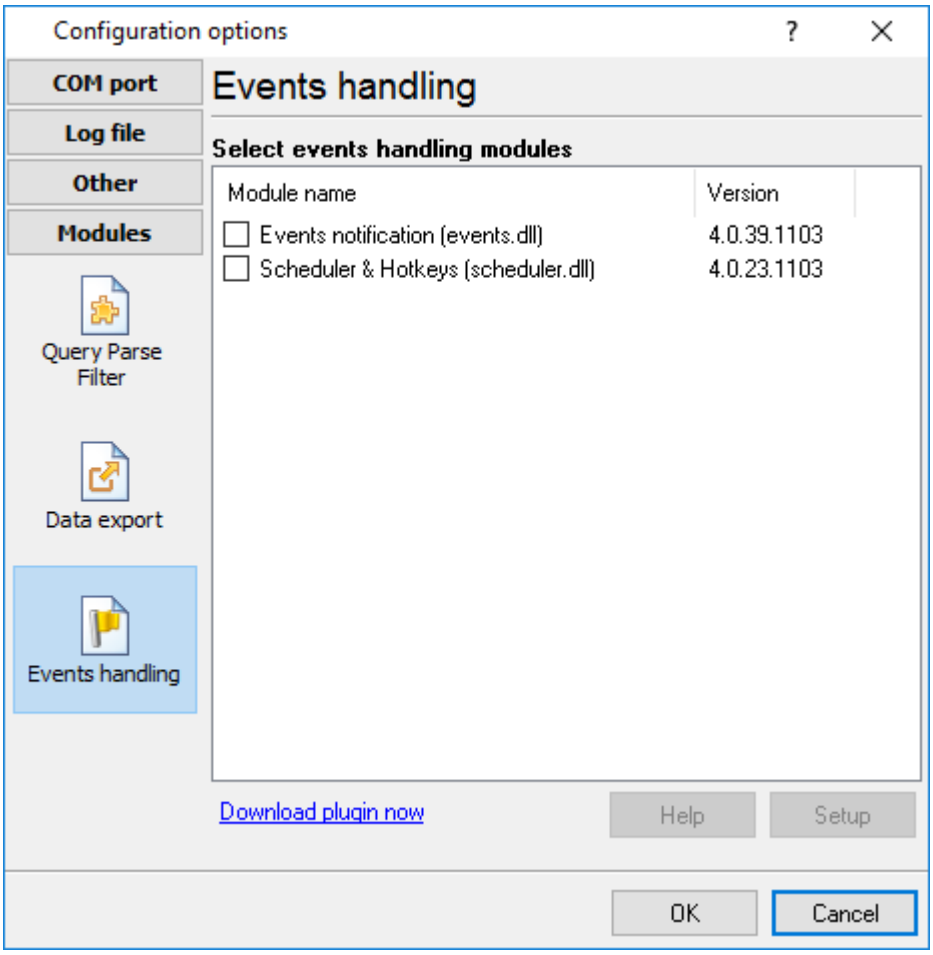

Fig.3. Examples of installed modules

4 Glossary

Main program – the main executable of the application. For example: Advanced Serial Data Logger and asdlog.exe. It allows creating several configuration. Each configuration may use many plugins.

Plug-in - the additional plugin module for the main program. The plugin module extend the functionality of the main program.

Parser – the plugin module that processes the data flow, singling out data packets from it, and then variables from data packets. These variables are used in data export modules after that.

Core - see "Main program".

5 Configuration file

5.1 General information

All configuration files must have the "xml" extension and be located in the "Configs" folder, which must be located next to the file "uniparser.dll."

Example:

```
uniparser
|--Configs
|   |--config.xml
|--uniparser.dll
```

XML files must have standard markup for XML format. In addition, the following general requirements must be met:

1. All logic values ("true" or "false") are presented as numeric values, where:
   - 0 - false.
   - 1 - true.

2. If the character [d] is present in the description of the attribute value, then this value is used by default:
   - 0 [d].
   - value - attribute description ([d] 0).

3. If a default value is not specified, it is either false, 0, or an empty string, depending on the data type.

4. When writing an XML file, you must replace some reserved characters with their valid HTML equivalent in attribute values. When XML file is loaded, substitution will occur automatically:
   - < to <
   - > to >
   - & to &
   - " (quotation mark) to "

5. Attribute values must be enclosed in quotation marks. It is allowed to use both double and single quotes.

Examples:

logtitle="value" - correct attribute value.
logtitle='value' - correct attribute value.
logtitle='value1 "value2"-value1' - correct attribute value, double quotes around value2 will be part of the attribute value, because the attribute values are enclosed in single quotes.
logtitle='value' - incorrect attribute value because the opening quotation mark is single and the closing quotation mark is double.
logtitle=value - incorrect attribute value because no quotation marks are used.
logtitle="value1 & value2" - correct replacement of a reserved character.

5.2 Encoding

The XML 1.0 specification allows you to start an XML document with a version declaration and document encoding.

Example:

```
<?xml version="1.0" encoding="windows-1251"?>

<?xml version="1.0" encoding="utf-8"?>
```

The module supports all Windows ANSI encodings: Windows 1250, Windows 1251, Windows 1252, etc., and also UTF-8 encoding.

The default encoding is Windows 1252.

5.3 XML file structure

XML files have a tree form structure. Each node of the tree has a name. Typically, nodes of the same level have the same name.

Example:

```
<Config name="MyPlugin [Custom Format 0]" logtitle="MyPlugin">
```

Config - the name of the tree node.

name, **logtitle** - attributes of the tree node.

Each node with a specific name has its own set of attributes. At the same time, attributes with the same name, but belonging to tree nodes with different names, can have different meanings.

5.4 Description of xml tree nodes

5.4.1 Comments

`<!-- ... -->` - Any text between the specified prefix and suffix will be treated as a comment.

5.4.2 Config

The root node of configuration. If this node is missing in the XML file, the file will not be loaded.

Parent nodes: no.

Attributes:

name - the name of the configuration that will be displayed in the program interface, in the list of parser modules.

logtitle - the text specified in this attribute will be added to all messages in the program message log received from this module.

5.4.2.1 Devices

The node contains the list of supported devices. A single configuration file can contain a description of multiple device configurations.

Parent nodes: Config

Attributes:

name - the name of the device list.

caption - the displayed title (reserved, not used in the program interface).

5.4.2.1.1 Includes

Contains a list of configuration tree nodes that can be used multiple times in other configuration locations.

Parent nodes: Devices, Device

Attributes: no

5.4.2.1.1.1 IncludeItem

Contains a group of configuration tree nodes that can be used multiple times in other configuration locations. Convenient to use for repetitive configuration blocks.

Parent nodes: Includes

Attributes:

name - the name of a group of tree nodes. This group of tree nodes is accessed by the specified name, using the element "Include."

Example:

```
<IncludeItem name="decode_gps_fix">
       <Item value="1" name="No fix" />
       <Item value="2" name="2D" />
       <Item value="3" name="3D" />
       <Item value="" name="Unknown" default="1" />
```

```
</IncludeItem>
```

5.4.2.1.2 Include

The node group, the description of which was previously declared using the node "IncludeItem", will be inserted in the specified configuration location.

Parent nodes: Any

Attributes:

name - the name of a group of tree nodes.

Example:

```
<Include name="decode_gps_fix" />
```

5.4.2.1.3 Device

Describes configuration of a single device.

Parent nodes: Devices

Attributes:

name - the name of a device.

caption - the displayed name of the device in the program interface.

5.4.2.1.3.1 SetVar

Declares a global variable that can be used in a device configuration for different purposes (to store temporary values, as a constant). The value of this variable is not stored or loaded and is initialized when the configuration is loaded (initialized).

Parent nodes: Device

Attributes:

name - the name of a global variable.

value - default value of a variable (when loading the configuration).

type - variable type ([d] not specified). If the value of this attribute is "const," then this variable is a constant and cannot be changed at run time.

Example:

```
<SetVar name="UNIQUE_ID" value="" />
```

Contains a list consisting of one or more options that will be available for editing by the user in the module configuration window. These options will be saved when you click "OK" in the window, and loaded when the configuration is initialized.

Parent nodes: Device, Command

Attributes: no

Description of one option that will be available for editing to the user.

Parent nodes: Options

Attributes:

name - the name of an option. When the configuration is initialized, a global variable with this name will be declared, which can then be accessed by name. When this variable is initialized, it will be assigned a user-defined value if the user has previously changed this value or a default value.

caption - the displayed name of an option in the program interface.

hint - a tooltip displayed in the program interface at the input field.

type - the type of the input field that will be used to edit this parameter in the program interface. The following values are possible:

- integer - integer number.
- string - string [d].
- uppercase - a string that can only contain capital letters.
- mixedcase - a string that can contain only letters in any case.
- lowercase - a string that can contain only lowercase letters.
- password - a string, entering the password.
- hex - a string that can contain only letters A-F and numbers 0-9.
- alphanumeric - a string that can contain only letters and numbers.
- validchars - a string that can contain only letters and numbers specified in the attribute "validchars."
- combodropdown - a list of values that allows entering a value manually, specified using the attributes "comboitems" and "combovalues," or using child nodes of the type "ComboItems." In this case, in the configuration settings, the text value (Text) from the drop-down list will be saved.
- combosimple - list of values that allows to input values manually.
- combodropdownlist - list of values without manual input of values.
- combodropdownvalueslist - list of values without manual input of values, however, the selected value "combovalues" or "value" from "ComboItems" will be saved in the configuration settings, which allows you to make a list that is independent of the operating system language.
- color - an integer containing the RGB color code.
- extended - a real number.
- checkbox - input field of the "checkbox" type, which can be in three statuses:
  - 0 - disabled [d].
  - 2 - enabled.

- 4 - intermediate status (a gray checkbox).

default - default value of an option.

min, **max** - only for numeric values. The attribute specifies the minimum or maximum value of this parameter. If the user enters a value outside the specified range during editing, the corresponding message will be displayed.

validchars - the attribute is a string containing allowed characters.

comboitems - a string containing a list of items in the drop-down list specified with the separator "~." For example, comboitem1~comboitem2~comboitem3.

combovalues - a string containing a list of values of elements of the drop-down list specified with the separator "~." For example, value1~value2~value3. The number of values must strictly correspond to the number of elements specified in the comboitems attribute. If this attribute is specified, the module will save and load the value of the element.

checktype - "check box" view.

- 1 - a checkbox with three statuses [d].
- 2 - a checkbox with two statuses.
- 3 - a radio button.

Example:

```
<Option name="DATAFORMAT" caption="Data format" type="combodropdownvalueslist" default="0">
       <ComboItems>
              <Item name="Engineering unit" value="0" />
              <Item name="Perecent of FSR (full scale range)" value="1" />
              <Item name="2s complement hexadecimal" value="2" />
              <Item name="Ohms" value="3" />
       </ComboItems>
</Option>
```

Contains a list of items in the combobox drop-down list.

Parent nodes: Option

Attributes: no

Item

The node contains a description of one item in the drop-down list.

Parent nodes: ComboItems

Attributes:

name - the displayed name of a drop-down list item.

value - the value of a given item in the drop-down list.

5.4.2.1.3.3 Request

Description of a request that the module can periodically send to the device. The interval for sending the request is set by the user. This node may be missing if you do not need to send requests. Contains Item, Items.

Parent nodes: Device

Attributes: no

5.4.2.2 Commands

Contains a list of commands (data packets) that this configuration can handle.

Parent nodes: Device

Attributes: no

5.4.2.2.1 Command

Description of one of the supported commands (data packets). The device can do the following:

- send and receive data packets.
- receive only.
- send only.

Parent nodes: Commands

Attributes:

name - the name of a command.

caption - the command name displayed in the program interface.

5.4.2.2.1.1 Responses

Contains a list of possible responses (one or more) and their format.

Parent nodes: Command

Attributes: no

Description of the response of one type.

Parent nodes: Responses

Attributes:

begin - signature (sign) of the beginning of data packet. Based on the signatures of the beginning and end of a data packet, the module selects data packets from the total data flow. A string or regular expression, depending on the begintype attribute. By default - not specified. Example: begin="+RGA".

begintype - signature type:
- 0 - string of characters [d].
- 1 - regular expression.

beginbuff - ([d] 0) whether or not to store the begin signature of the data packet in the input data buffer. By default, upon detection of the signature in the input data buffer, the module deletes the signature and all data up to the signature from the buffer. However, sometimes, when using regular expressions, it is useful to keep the signature in the buffer for further analysis. For that purpose, you need to set this attribute to 1.

end - ([d] not specified) signature (sign) of the end of a data packet. A string or regular expression, depending on the **endtype** attribute. Example: end="#0D#0A"

endtype - signature type:
- 0 [d] - string of characters. If you want to set a non-printable character as a sign of the end or beginning of a data packet, you can do this by specifying its hexadecimal code. See example: end="#0D#0A". In this example, the sign of data packet end will consist of two ASCII characters, CR and LF and codes 0x0Dh and 0x0Ah, respectively.
- 1 - regular expression.
- 2 - the end of a data packet is not specified, but each packet has a fixed length specified by the **len** attribute.
- 3 - the end of a data packet is determined by **timeout**. Timeout value in milliseconds is set using the timeout attribute. If there is no data within the specified time interval, the data packet will be considered fully received.
- 4 is similar to mode 3, but in this case, packets will be extracted from the data accumulated in the buffer during the timeout using type 0 end signature.
- 5 is similar to mode 3, but in this case packets with type 1 end signature will be extracted from the data accumulated in the buffer during a timeout.
- 6 - data are accumulated during the **timeout** period. Data are then retrieved using all rules from the parent group Responses.

endbuff - ([d] 0) whether or not to store the end signature of the data packet in the input data buffer. Similar to **beginbuff** attribute.

len - the length of a data packet in bytes. By default, this value is not specified, i.e., the length of the data package is not controlled.

override - ([d] 0) if override is "1", and **endtype** is "3", then the signature of this response can be overridden by other Response signatures in the same parent group Responses. If a more suitable signature is found during the timeout, it will be used. For example, it may be a data packet with clearly defined **endtype** and **begintype** equal to 0 or 1.

validatesize - ([d] 0) check buffer size before extracting data. If during the processing of a data packet it turns out that the number of bytes in the packet is less than necessary, and this attribute

is 1, an error message will be generated and the packet will be considered invalid and will not be processed.

timeout - ([d] 500) timeout of packet end signature or a specified amount of bytes (depending on the value of the **endtype** attribute). If during the specified interval (in milliseconds) the packet end signature is not found in the buffer, the contents of the buffer will be deleted.

parse - a string consisting of one or more characters from the series: "b,p,e," and defines the data from which the parser will extract data:
- o b - the data packet begin signature.
- o p - the data packet, that is everything that was received between the start and end signatures of the data packet.
- o e - the data packet end signature.

Example: parse="bp". In this example, the parser will extract data only from the data packet start signature and the data packet body.

The node contains a list of response elements combined into a single group. The attributes specified for this node will be inherited by all child nodes. It is convenient to use groups of response elements to specify elements with several identical attributes (for example, if).

Parent nodes: Response

Attributes: match the attributes of the Item node.

This node also has a special attribute that is not inherited by child nodes:

case - (**[d]** not specified) positive integer. If this attribute is specified, only the first child element of the list, with the condition specified in if attribute returning "true," will be processed.

Description of one response element.

Parent nodes: Responses, Items

Note: if a data packet is handled by a function, it is assumed that it is a part of the data packet specified by **parse** attribute.

Attributes:

type - a type of a response element. One of the following values:
- o **break** - immediate termination of iterating through elements of a response.
- o **goto** - go to the response element, the identifier of which is set by the **expr** attribute. The target response element must have **id** attribute equal to the value **expr**. Response element can be lower or higher in the list. Special attention should be paid to the inadmissibility of infinite cycles.
- o **log** - a message set with the **value** attribute will be added to the program log.

- **push** - the value of a response element with the name specified in the **name** attribute will be saved as a global variable. This feature allows you to store and use values from previous data packets. If the **value** attribute is set, the value will be stored under this name.
- **pop** - a value of a global variable with the name specified by the **name** attribute, that has been saved earlier, will be used as the value of a response element with the same name. If the **value** parameter is specified, then the value with the name specified in the value parameter will be obtained from the list of global variables and stored in a local variable with the name from the name parameter. If the **datatype** parameter is set for **pop** element, then the resulting value will be converted to the specified data type. Otherwise, the data type of the stored value will be used.
- **reset** - a global variable with the name specified in the **name** attribute will be deleted.
- **const** - a fixed set of characters or bytes specified in the **value** attribute will be assigned to a variable with the name of the **name** attribute. It is used in query elements, or for creating response elements with a fixed value. Example: <Item type="const" pos="1" value="*NULL*" />
- **format** - the additional attribute format specifies formatting functions that can be applied to a local variable whose name is specified in the **name** attribute.
- **filter** - one of filter rules specified by using filtertype attribute will be applied. If the **name** attribute is set for the response element, a filter will be applied to the value of the specified local variable. Otherwise, the rule will be applied to the entire data packet.
- **decode** - a search among child elements will be performed. In the case the value attribute of a parent node and a child node match each other, then the value specified by the name attribute of the child node will be assigned to the variable specified in the name attribute. If no matches are found during the search, the value of the child node, to which the logical attribute default="1", will be assigned. Example:

```
<Item type="decode" pos="1" size="1" value="{CALL_TYPE}" name="DIRECTION" export="1">
       <Item value="0" name="INT" />
       <Item value="1" name="OUT" />
       <Item value="2" name="IN" />
       <Item value="" name="INT" default="1" />
</Item>
```

In this example, if the value of the variable "CALL_TYPE" is "1", then string value "OUT" will be assigned to the variable "DIRECTION." If the value of variable "CALL_TYPE" is "10", then string default value "INT" will be assigned to variable "DIRECTION."

- **var** - the value of one local variable will be assigned to another local variable. Example:

```
<Item type="var" pos="1" size="-1" value="{CALL_TYPE}" name="DIRECTION" export="1">
```

In this example, the value of the variable "CALL_TYPE" will be assigned to the variable "DIRECTION."

- **hex** - specifies a string of characters in the form of hexadecimal codes. Example:

```
<Item type="hex" pos="1" size="-1" value="#0D#0A" name="VALUE" export="1">
```

- **fix** - retrieves size bytes from the data packet starting from the pos position. The attribute value can be used.
- **delimit** - retrieves from the data packet the element whose number is set by numeric attribute **ordernum**. It is assumed that a data packet is a string in which one or more values are specified with a delimiter. **Ordernum** starts with "1". **ordernum** can be an arithmetic expression. The attribute value can be used. Examples:

Data packet: 123,321;456;789;012

```
<Item type="delimit" pos="1" size="-1" ordernum="1" delimiter=";" name="FLAG1" datatype="integer" /
```

The result "123,321" will be saved in the variable "FLAG1".

```
<Item type="delimit" pos="1" size="-1" ordernum="2" delimiter="," name="FLAG2" value="FLAG1" datat
```

The result "321" will be saved in the variable "FLAG2".

- **regexp** - retrieves an element from a data packet using a regular expression whose ordinal number is specified by the **ordernum** numeric attribute. The regular expression is specified by the **expr** attribute. **Ordernum** starts with 1. **ordernum** can be an arithmetic expression. In a regular expression, parentheses must be used to extract the required values. If multiple pairs of parentheses are used in an expression, with the **match** attribute, you can specify the number of the selected element (1, 2, etc.). The attribute value can be used. Examples:

CALL_TYPE = 123456I

```
<Item type="regexp" size="-1" name="_DIRECTION" expr="(I|O)" value="{CALL_TYPE}" datatype="string"
```

The result "I" will be saved in the variable "_DIRECTION."

```
<Item type="regexp" size="-1" name="_BEGIN" expr="^(.{5})" datatype="string" />
```

The first five bytes of the data packet will be stored in the "_BEGIN" variable.

- **expr** - calculates the expression specified by the **expr** attribute and places the result in a variable whose name is specified in **name**. Example:

<Item type="expr" size="-1" expr="'IN'+'{DIRECTION}'" name="CALL_TYPE" datatype="string" />

If "DIRECTION" is string ‘12345’, then ‘IN12345’ will be stored in 'CALL_TYPE.'

- **pascal** - executes Pascal script specified as the value of the given node. Example:

```
<Item size="-1" type="pascal"><![CDATA[begin SendByte($06); end.]]></Item>
```

In this example, the script calls a built-in function that sends one byte. You can see the full list of functions supported by the scripting language in the UniParser_ScriptTest utility.exe or in the "Script Execute" module documentation.

- **crc_AAA** - calculates the checksum, where AAA is checksum type. When calculating checksum, the start point and length of a data block is specified using the **pos** and **size** attributes. This response element always returns a numeric value of the checksum. Example:

```
<Item type="crc_crc8-sum" pos="0" size="10" name="CRC" />
```

name - the name of a response element (local variable).

value - ([d] not set) the assignment of an attribute depends on the type of element response. In elements of type **fix**, **expr**, **regexp** it can specify:

- Name of variable. The variable name consists of Latin characters, numbers, and underscore. In this case, the string value of the specified variable is handled. If the variable is not found, an empty string will be used.
- String with wildcard characters. In this case, it substitutes wildcard characters handles the result of the substitution.

pos - ([d] 0) an offset of data of the given response element in a data packet. Data handling starts from zero. Such arithmetic expressions as "{VAR}+1", where {VAR} is a global or local numeric variable, or a previously specified response element named "VAR," can denote size. If you set "-1" as the value of this parameter, then the offset will be calculated automatically using the offset and data size of the previous response elements.

size - ([d] -1) size of response element data. If the data size is specified, the module will extract the specified amount of bytes. If with the given offset it is impossible to get the specified number of bytes due to insufficient length of a data packet, then, depending on the validatesize attribute, either an error message will be displayed, or default value will be used. Arithmetic expressions can denote size.

default - ([d] not set) default value for the response element. It will be used in the following cases:
- it is impossible to extract data under the set conditions.
- it is impossible to convert a value to the specified data type.
- it is impossible to apply a formatting function to the value because of a data type mismatch.

format - list of formatting functions separated with a semicolon. With these functions, you can perform simple conversions of data types, and perform some actions with values. Example: format="trim;uppercase".

caption - ([d] not set) displayed name of response element or request element. It is used in the program interface and debugging log files.

datatype - ([d] string) data type of value. One of the following values: string, boolean, float, smallInt, integer, word, dword, date, time, datetime, unknown. If formatting functions are used, the last function must return a result that can be converted to the specified data type.

export - numeric value.
- 0 [d] - the value of the response element can only be used while processing the data packet.
- 1 - this response element will be exported, i.e., the parser will pass it to the main program after processing the data packet.
- 2 - will only be exported if its value is not NULL.
- 3 - exported only if its value is not NULL and not empty (applicable to string values).
- 4 - exported only if its value is not NULL and not false (applicable to boolean values).

Note: it is recommended that you start setting local variables, that will not be exported, with an underscore character. Example: name="_TEMP".

count - ([d] 1) a positive integer or arithmetic expression. Sets the number of elements with identical attributes that only differ in name. If count is greater than one, two special variables are added to the list of local variables when processing the data packet, which can be used to specify the name of the response element or its caption:
- ITEM_COUNT - the counter that is automatically increasing from 1 to count.
- ITEM_COUNT_Z - the counter that is automatically increasing from 0 to count-1.

That makes it easy to set a large number of similar response elements. Example:

```
<Item name="NAME{ITEM_COUNT_Z}" count="{_COUNT1}" caption="Name{ITEM_COUNT_Z}" export="1">
```

In this example, you can specify multiple response elements using a single string. The number of response elements is determined by the local variable "_COUNT1", which must be defined beforehand. The name of response elements will look as follows: NAME0, NAME1, NAME2, etc. The same way you can form the caption of the response element.

filtertype - the type of a filter rule for the response element of type "filter." A filter rule (string or regular expression) is set using the **expr** attribute. If the **name** attribute is specified, then all actions will be performed with the value of the variable with the specified name. Otherwise, all actions will be performed with the data packet.

- 0 - enables formatting functions from the format attribute.
- 1 [d] - search for string match. If a string is found or not found (see **exprtype**), the data packet will be ignored. In this case, searching for remaining response elements will stop. No data will be exported.
- 2 - searching for a match is running with the use of a regular expression.
- 3 - deletes all occurrences of the string specified in **expr**.
- 4 - deletes all characters specified in **expr**.
- 5 - deletes all characters whose code is less than 0x20h.
- 5 - deletes all characters whose code is less than 0x7Fh.
- 7 - deleting with the use of a regular expression.
- 8 - replaces a string with the one specified in **target** attribute.
- 7 - replacement with the use of a regular expression.

expr - expression. The value depends on the type of element.

exprtype - the subtype of a filter **filtertype** 1 and 2:

- 0 [d] - filter is not used.
- 1 - checks if a string is available.
- 2 - checks if a string is missing.

if - ([d] not specified) condition under which the response element will be processed. If this condition returns "false," this response element will not be processed.

Example:

```
... if="LENGTH('{FLAG3}')&lt;=5" />
```

This example controls the length of the string variable FLAG3.

A condition can be set for a group of response elements. If a condition is specified for a group, as well as an individual item of that group, both conditions must be met.

5.4.2.3 Predefine

This node describes default settings for the entire module.

Parent nodes: Config

Attributes:

All attributes below are of "bool" type and have the default value "0". The specified action will be executed only if the value is "1".

tree - dialog window of the module will display the parameters tree.

reqtype - the module will allow you to choose the type of request (at an interval, at a certain time, etc.).

respitems - the module will allow you to add and configure response elements.

dataquery - the module will be visible both in the list of data query modules and in a data handling module.

command - module will allow you to select the type of command that will be sent to the device. It is only required if two or more commands are specified for the device.

The attributes below are to be specified only if the configuration is created for two or more devices.

unique - (bool, [d] 0). If "1", then the module will add a unique prefix to the name of each exported variable for each device that is added to the module configuration.

device - (bool, [d] 0). If "1", then the module will allow adding devices to the configuration.

5.4.2.3.1 QUERYXML

This node contains a predefined configuration for all devices and queries in this XML configuration file.

Parent nodes: Predefine

Attributes: no

Example of predefined configuration:

```
<Predefine tree="1" unique="0" device="0" command="0" reqtype="0" respitems="0" dataquery="
	<QUERYXML>
		<Request caption="My Request" expanded="1">
			<DeviceType>DEF</DeviceType>
			<ReqType>DEF</ReqType>
			<ItemType>B</ItemType>
			<RequestMethod>
				<Timeout>2000</Timeout>
				<MethodPoll check="2">
					<PollInterval>5</PollInterval>
					<PollIntervalUnits>1</PollIntervalUnits>
				</MethodPoll>
			</RequestMethod>
		</Request>
	</QUERYXML>
</Predefine>
```

5.4.2.3.1.1 Request

This node contains predefined settings for one of the queries in the configuration file.

Parent nodes: QUERYXML

Attributes:

caption - ([d] not specified) Caption or description of query. It will be visible in the module settings window.

expanded - (bool, [d] 0). If "1", then a node with this query/response will be expanded when the configuration window is opened.

Данный узел содержит предопределенные настройки для одного из запросов в конфигурационном файле.

Value of this node contains the name of the device with predefined settings. The value must match the **name** attribute of the node Device.

Parent nodes: QUERYXML => Request

Attributes: no

Value of this node contains the name of the query with predefined settings. The value must match the **name** attribute of the node Command.

Parent nodes: QUERYXML => Request

Attributes: no

This node value sets the run mode. Possible values:

- D - Disabled. Requests are not sent, and responses are not processed.
- B - Requests are sent and responses are not processed.
- R - Requests are not sent, but responses are processed.
- S - Requests are sent, but responses are not processed.

Parent nodes: QUERYXML => Request

Attributes: no

Group of parameters for sending a request (at an interval, at a certain time, etc.).

Parent nodes: QUERYXML => Request

Attributes: no

Response timeout in milliseconds.

Parent nodes: QUERYXML => Request => RequestMethod

Attributes: no

Option to send a query is only available once when opening the data source.

Parent nodes: QUERYXML => Request => RequestMethod

Attributes:

check - numeric attribute. If the value is 2, then this type of request is active.

Option to send a request at a specific time interval. See attributes in MethodOnce.

Poll interval value. Positive numeric value.

Parent nodes: QUERYXML => Request => RequestMethod => MethodPoll

Attributes: no

Value of poll interval unit:

- 0 - milliseconds.
- 1 - seconds.
- 2 - minutes.
- 3 - hours.

Parent nodes: QUERYXML => Request => RequestMethod => MethodPoll

Attributes: no

6 Configuration debugging

In order to debug configurations, the module can create a debug log file. For that purpose, you need to create an empty uniparser.dbg file next to the uniparser file.dll and restart the program. When DLL is loaded, the DBG file is checked. And if it exists, logging mode is enabled.

7 List of formatting functions

utc2local, local2utc - converts UTC time to local time and vice versa.

dec2hex, hex2dec - converts decimal value to hexadecimal and vice versa.

num2date - converts numeric value into TDateTime data type.

time2sec - returns the number of seconds from the beginning of the day.

sec2datetime, msec2datetime - converts seconds, milliseconds to TDateTime data type.

sec2datetime2, msec2datetime2 - converts the number of seconds, milliseconds from the beginning of the era (January 1, 1970) to TDateTime data type.

be2le - converts date in Big Endian format to Little Endian format.

bytes2uint64 - converts eight bytes into Int64.

bytes2int64 - converts eight bytes into Int64.

bytes2dword - converts four bytes to DWORD type.

bytes2word - converts two bytes to WORD type.

bytes2byte - converts a byte to Byte.

bytes2longint - converts four bytes into Longint type.

bytes2smallint - converts two bytes into Smallint.

bytes2shortint - converts a byte to Shortint type.

bytes2signle - converts four bytes to Single.

bytes2double - converts eight bytes to Double.

getbit(X) - checks if the specified bit is set as a number or not (count starts from zero).

leadzero - adds zeros to the beginning of the string until its size is equal to the size specified in the size attribute.

leadspace - the same, but with a space character added.

hi - function returns the high-order byte from a numeric value.

lo - function returns the low-order byte from a numeric value.

trim - trims spaces on the left and right of a string value.

trunc - converts a real number to an integer, discarding the fractional part.

trimleft - trims space characters to the left of a string value.

trimright - trims space characters to the right of a string value.

uppercase - converts a string value to uppercase.

utf8decode, utf8encode - converts a string value to UTF-8 encoding.

lowercase - converts a string value to lowercase.

delete(X, Y) - deletes Y characters in position X in a string value. X, Y - integers.

substr(X, Y) - returns Y characters in X position of a string value. X, Y - integers.

div(X) - divides a numeric value by X, where X is an integer or a real number.

mul(X) - multiplies a numeric value by X, where X is an integer or a real number.

sub(X) - subtracts X from a numeric value, where X is an integer or a real number.

add(X) - adds X to a numeric value, where X is an integer or a real number.

If the function name starts with a special character "@," then the built-in expression interpreter is used to execute this function. When this function is called, the current value of the response element is passed as a function parameter, and the result of the function execution becomes the new value of the response element.

8 List of checksum types

Below is the list of supported checksums:

crc8-sum
crc8
crc8-darc
crc8-i-code
crc8-itu
crc8-maxim
crc8-rohc
crc8-wcdma

crc16-sum
crc16
crc16-buypass
crc16-dds-110
crc16-dect
crc16-dnp
crc16-en-13757
crc16-genibus
crc16-maxim
crc16-mcrf4xx
crc16-riello
crc16-t10-dif
crc16-teledisk
crc16-usb
crc16-ccitt-1D0F
crc16-ccitt-FFFF
x25
xmodem
modbus
kermit

crc24
crc24-flexray-a
crc24-flexray-b

crc32
crc32-bzip2
crc32c
crc32d
crc32-mpeg
posix
crc32q
jamcrc
xfer