The Script execute plugin
PRINTED MANUAL

Script execute plugin

Printed: 3/6/2026

Publisher

AGG Software

Production

©2006-2026 AGG Software
https://www.aggsoft.com

Table of Contents

1 Introduction

The filter module "Script execute" for our data loggers (for example, Advanced Serial Data Logger or Advanced TCP/IP Data Logger) is an interpreter for scripts written in most popular programming languages. Unique feature of this plug-in is ability to use several languages (PascalScript, C++Script, JScript and BasicScript), so you can write scripts using your favorite language. The plug-in doesn't use Microsoft Scripting Host, so it can be used on any computer with Windows.

"Script execute" combines cross-platform scripting, fast code execution, small footprint, rich set of features and a splendid scalability. Make your applications the most flexible and powerful ones with "Script execute".

It is a smart tool easy in use. Supports most standard functions and operators for all languages. Within scripts you can change values of parser variables or/and add new variables to the export. The plug-in allows you to filter data packets and send data to a port.

- Multi-language architecture allows you to use a number of languages (at present moment PascalScript, C++Script, BasicScript, JScript).
- Standard language set: variables, constants, procedures, functions (nested functions allowed) with var/const/default parameters, all the standard operators and statements (including case, try/finally/except, with), types (int, float, bool, char, string, multi-dimensional array, enum, variant), classes (with methods, events, properties, indices and default properties).
- Types compatibility checking.
- Access to any some standard class (for example, TStrings, TFileStream).
- Allows storing values between executions.
- Allows sending string, bytes or data arrays to a port.
- Can be used with multiple ports and in multiple configurations.

2 System requirements

The following requirements must be met for "Script execute" to be installed:

Operating system: Windows 2000 SP4 and above, including both x86 and x64 workstations and servers. The 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.

3 Installing Script execute

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.

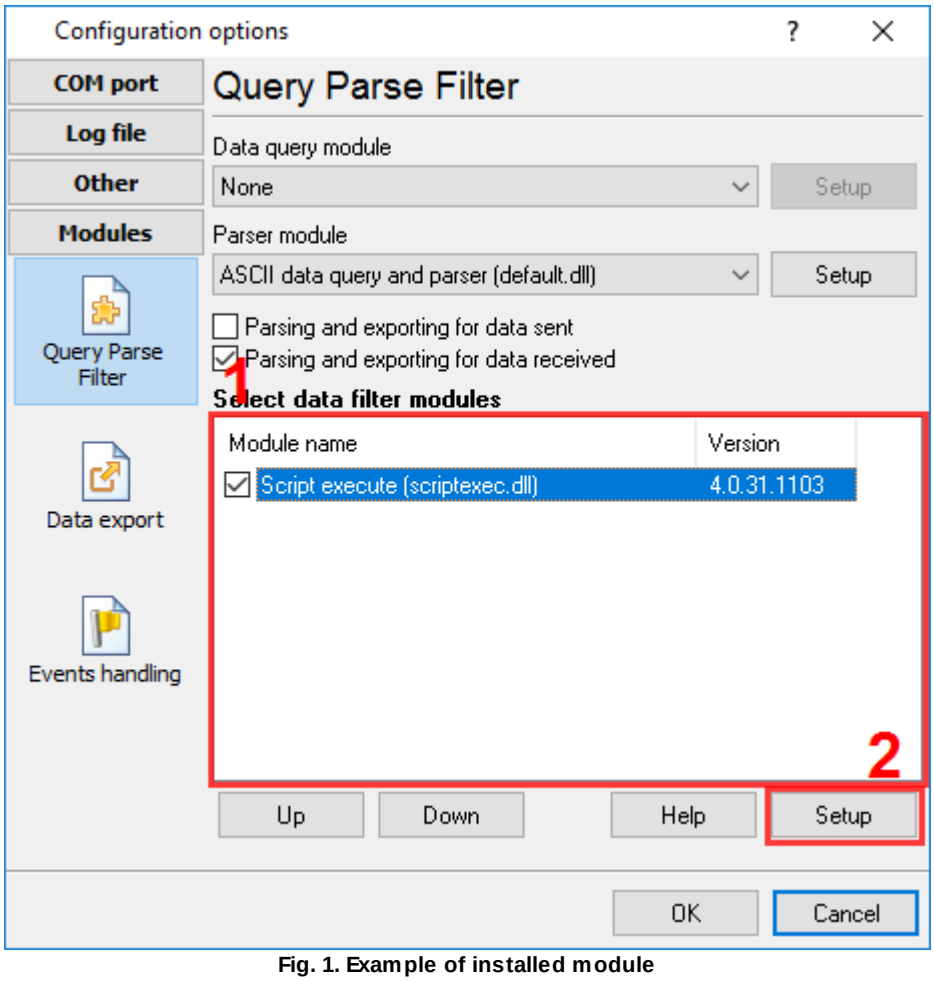


Fig. 1. Example of installed module

4 Glossary

Main program - it is the main executable of the application, for example, Advanced Serial Data Logger and asdlog.exe. It allows you to create several configurations with different settings and use different plugins.

Plugin - it is the additional plugin module for the main program. The plugin module extends the functionality of the main program.

Parser - it is 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."

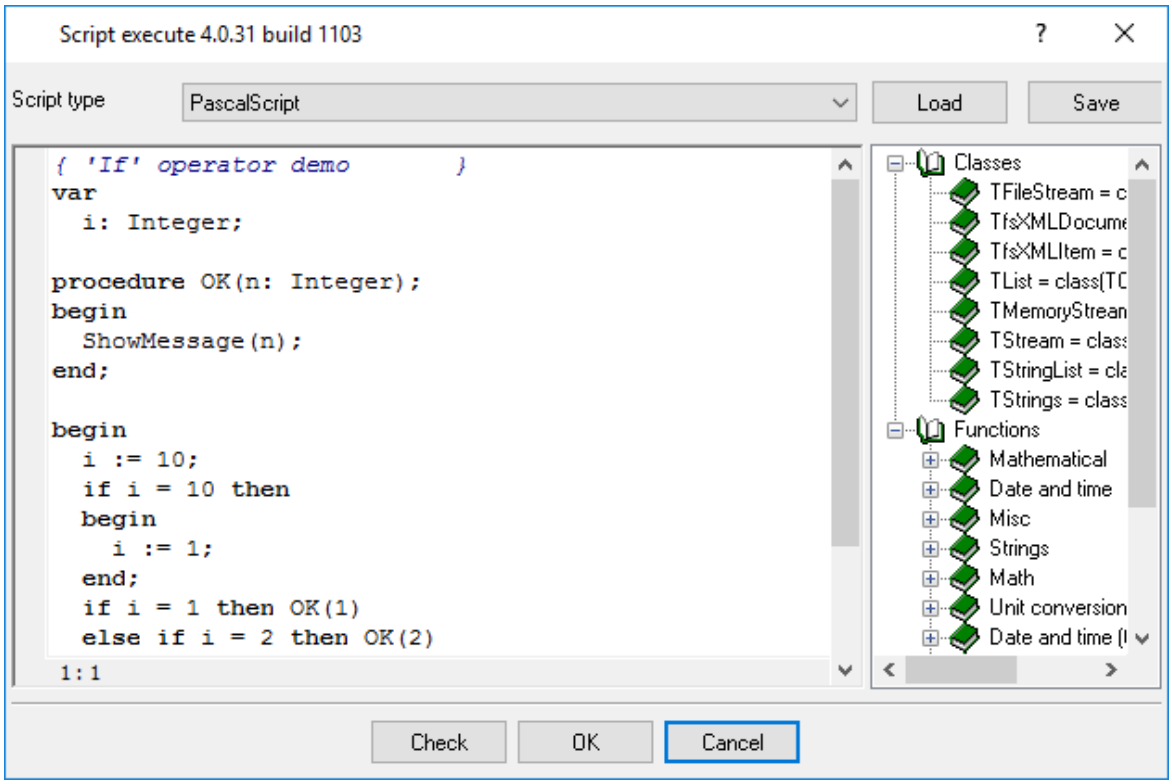
5 Configuration

The module configuration is very simple (pic.1). You may select a script type and specify a script in the editor window. You can add classes or functions from the left tree by double-click.

All supported functions are described in following sections.

When you'll complete editing, you should click the "Check" button and verify your script. Then you can click the "OK" button and the plug-in will apply all your changes.

You can load and save your scripts to a file by clicking "Load" or "Save" buttons.



Pic.1. The configuration window

Hot keys

Key	Action
Cursor arrow	Cursor moving
PgUp, PgDn,	Page Up / Page Down
Ctrl+PgUp	Move to the begin of text
Ctrl+PgDn	Move to the end of text
Home	Move to the begin of line
End	Move to the end of line
Enter	Move to the next line
Delete	Delete symbol at right or selected text
Backspace	Delete symbol at left
Ctrl+Y	Delete current line
Ctrl+Z	Undo last change
Shift+Стрелки курсора	Select the text block
Ctrl+A	Select all text
Ctrl+U	Unindent selected block

Ctrl+I	Indent selected block
Ctrl+C, Ctrl+Insert	Copy to clipboard
Ctrl+V, Shift+Insert	Paste from clipboard
Ctrl+X, Shift+Delete	Cut to clipboard
Ctrl+Shift+<цифра>	Set bookmark
Ctrl+<цифра>	Goto bookmark
Ctrl+F	Search text
F3	Continue search

6 Supported functions

6.1 Missing features

- No type declarations (records, classes) in the script; no records, no pointers, no sets (but you can use 'IN' operator - "a in ['a'..'c','d']"), no shortstrings, no GOTO statement.
- C++Script: no octal constants; no 'break' in the SWITCH operator (SWITCH works like Pascal CASE); '++' and '--' operators are possible only after the variables, that is '++i' is not allowed; '--', '++' and '=' operators do not return a value, that is 'if(i++)' is not allowed; all the identifiers are case-insensitive;
- NULL constant is the Pascal Null - use nil instead of NULL.

6.2 Data types

Internally the plug-in operates with the Variant type and is based on it. Nevertheless, you can use the following predetermined types in your scripts:

```
Byte      | Same as Integer type
Word      |
Integer   |
Longint   |
Cardinal  |

Boolean   | Boolean type

Real      | Same as Extended type
Single    |
Double    |
Extended  |
TDate     |
TTime     |
TDateTime |
```

```
Char      | Char type

String    | String type

Variant   | Same as Variant type
Pointer   |

Array     | Array type
```

C++Script maps some types to standard types:

int, long = Integer
void = Integer
bool = Boolean
float = Extended

JScript has no types, all types are variants. BasicScript may have types (for example, dim i as Integer), or may have no types and even no variable declaration. In this case a variable will have Variant type.

Not all of these types can be assign-compatible. Like in Object Pascal, you can't assign Extended or String to an Integer. Only one type - the Variant - can be assigned to all the types and can get value from any type.

6.3 Mathematical

```
Abs(e: Extended): Extended
ArcTan(X: Extended): Extended
Cos(e: Extended): Extended
Exp(X: Extended): Extended
Frac(X: Extended): Extended
Int(e: Extended): Integer
Ln(X: Extended): Extended
Pi: Extended
Round(e: Extended): Integer
Sin(e: Extended): Extended
Sqrt(e: Extended): Extended
Tan(X: Extended): Extended
Trunc(e: Extended): Integer
```

6.4 String functions

```
Chr(i: Integer): Char
CompareText(s, s1: String): Integer
Copy(s: String; from, count: Integer): String
Delete(var s: String; from, count: Integer)
Insert(s: String; var s2: String; pos: Integer)
```

```
Length(s: Variant): Integer
Lowercase(s: String): String
NameCase(s: String): String
Ord(ch: Char): Integer
Pos(substr, s: String): Integer
SetLength(var S: Variant; L: Integer)
Trim(s: String): String
Uppercase(s: String): String
```

6.5 Date and Time

```
Date: TDateTime
DayOfWeek(aDate: TDateTime): Integer
DaysInMonth(nYear, nMonth: Integer): Integer
DecodeDate(Date: TDateTime; var Year, Month, Day: Word)
DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word)
EncodeDate(Year, Month, Day: Word): TDateTime
EncodeTime(Hour, Min, Sec, MSec: Word): TDateTime
IsLeapYear(Year: Word): Boolean
Now: TDateTime
Time: TDateTime
```

6.6 Miscellaneous functions

Other

```
CreateOleObject(ClassName: String): Variant
Dec(var i: Integer; decr: Integer = 1)
Inc(var i: Integer; incr: Integer = 1)
InputBox(ACaption, APrompt, ADefault: string): string
InputQuery(ACaption, APrompt: string; var Value: string): Boolean
MessageDlg(Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; HelpCtx:
RaiseException(Param: String)
Random: Extended
Randomize
ShowMessage(Msg: Variant)
ValidDate(cDate: String): Boolean
ValidFloat(cFlt: String): Boolean
ValidInt(cInt: String): Boolean
VarArrayCreate(Bounds: Array; Typ: Integer): Variant
VarType(V: Variant): Integer
```

Conversion

```
DateTimeToStr(e: Extended): String
DateToStr(e: Extended): String
FloatToStr(e: Extended): String
IntToStr(i: Integer): String
```

```
StrToDate(s: String): Extended
StrToDateTime(s: String): Extended
StrToFloat(s: String): Extended
StrToInt(s: String): Integer
StrToTime(s: String): Extended
TimeToStr(e: Extended): String
VarToStr(v: Variant): String
```

Formatting

```
Format(Fmt: String; Args: array): String
FormatDateTime(Fmt: String; DateTime: TDateTime): String
FormatFloat(Fmt: String; Value: Extended): String
FormatMaskText(EditMask: string; Value: string): string
```

6.7 Special functions

```
procedure SetVariable(Name: String; Value: Variant)
```

The procedure changes or adds a variable in a data packet. Later this variable can be used in a data export plug-in.

```
function GetVariable(Name: String):Variant
```

The procedure retrieves a value of a variable in a data packet. If the variable doesn't exist in the data packet, then the function returns Null.

```
procedure PushVariable(Name: String; Value: Variant)
```

The procedure stores a named value in an internal storage. Later you can retrieve this value and use in your script. If the value exists in the storage, then the procedure will overwrite it. The internal storage is empty on starting and will be cleared if the module configuration is changed.

```
function PopVariable(Name: String):Variant
```

The procedure retrieves a named value, previously stored by PushVariable. If the variable had not been stored before, then the function returns Null.

```
procedure DeleteVariable(Name: String)
```

The procedure deletes a variable in a data packet.

```
procedure DiscardDataPacket
```

Call this procedure if you don't want to export a current data packet. The module will not pass this data packet down to a data export module.

```
function IsVariableDefined(Name: String):boolean
```

With help of this function you can verify that a variable exists in a data packet.

```
function IsVariableStored(Name: String):boolean
```

With help of this function you can verify that a named value has been stored before.

```
function SendData(Data: String):boolean
```

This function allows you to send a data array defined as a string to a port. For example Data = #01+'Test'+#02

```
function SendString(Data: String):boolean
```

This function allows you to send a string value. Before sending, the data logger will replace all characters like #0D with a characters with a corresponding hexadecimal code.

```
function SendByte(Data: Byte):boolean
```

This function allows you to send a single byte.

```
function SendDataToDataSource(DataSource, Data: String):boolean
```

Example:
`SendDataToDataSource('COM3', 'Test')`

The function is identical to `SendString`, but you may specify a name of a target data source. The value for the DataSource parameter you may get from the drop-down list in the main window.

```
procedure DbgLog(Msg: String; LogLevel: Byte = 0)
```

The procedure sends a message to a message log. The LogLevel parameter can be: 0 - error, 1 - warning, 2 - information.

```
procedure SendEvent(Name: String)
```

The procedure generates an internal program event with the specified identifier.

```
procedure SendEventEx(Name: String; Args: array of const)
```

The procedure generates an internal program event with the specified identifier and several arguments.

Example: `SendEventEx('EVENT-ID', ['ValueName1', 0, 'ValueName2', 'Test'])`

```
procedure RedirectData(DataSource: String)
```

The procedure redirects data of the current data packet to another configuration. See also SendDataToDataSource above for an example.

```
function GetVariableCount:integer
```

The function returns the number of variables in the current data packet. You may use this one and next two function to iterate through all variables in the current data packet.

```
function GetVariableByIndex(AIndex: integer):Variant
```

The function returns the variable value by its index in the data packet.

```
function GetVariableNameByIndex(AIndex: integer):string
```

The function returns the variable name by its index in the data packet.

```
procedure NewRow()
```

The procedure creates a new list of variables for a new data packet. All subsequent calls of all special functions will operate with the new data packet.

```
procedure NewLogFile(DataSource: String)
```

This function starts a new log file in the DataSource configuration in the logger. The logger adds a sequential number to the current log file name. See also SendDataToDataSource above.

Example:

```
NewLogFile('192.168.1.3:2000')
```