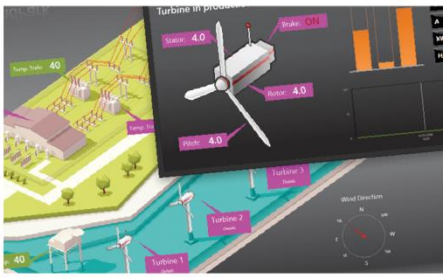
INDUSTRIAL AUTOMATION SOFTWARE

progea

Movicon.next™

Automation Platform.next generation



Movicon NExT

12.0 Scripting

Ver.3.4.268

Table of Contents

1. WWB.NET LANGUAGE 1

1.1. PREFACE 1
1.2. INTRODUCTION 1

2. BASIC SCRIPT IN PROJECT 3

2.1. WINWRAP BASIC LANGUAGE 3
2.2. BASIC SCRIPT AS RESOURCES 4
2.3. BASIC SCRIP IN PROJECT 6
2.4. CODE IN BASIC SCRIPTS AND IN SYMBOLS 6
2.5. BASIC SCRIPT LIBRARIES 6
2.6. SUB, FUNCTION, EVENTS, METODS AND PROPERTIES 8
2.7. SERVER-SIDE SCRIPT 10
2.8. BASIC SCRIPT VARIABLES 12
2.9. WWB.NET™INTO OBJECTS 16
2.10. EVENT MANAGER 17

3. BASIC SCRIPT EDITOR 19

3.1. BASIC SCRIPT EDITOR 19
3.2. BASIC SCRIPT TOOLBAR 20
3.3. BASIC SCRIP DEBUG 21

4. BASIC SCRIPT PROPERTIES 23

1. WWB.NET Language

1.1. Preface

All the information contained in the Platform.NExT documentation is based on the assumption that:

- Windows refers to the trademark registered by Microsoft inc. in Windows 7, Windows 8 and Windows 10 in 64 bit versions
  - Platform.NExT refers to the supervision system developed by Progea and is protected by the international Copyright
- WWB.NET refers to the WinWrap.NET™ component by Polar Engineering integrated in Platform.NExT, compatible with the Microsoft VB.NET language.
  - Ms Access and SQL Server refer to the Microsoft inc. trade mark
  - Any other product or brand mentioned is covered by Copyright on behalf of its owner

Warning: The use of script code in Movicon projects is subject to a LIMITED WARRANTY. The User has complete freedom to manage script when using events and methods of the product and of third party components. However Progea may not be able to completely guarantee the correct behaviour of the product in the event of using script code in properties or specific events or methods. Therefore Users are strongly advised to always refer to the documentation whenever possible and take preventive measures by checking that the code they intend to use functions correctly. Nevertheless Progea provides a LIMITED WARRANTY for those project functionalities that use VB.NET.

1.2. Introduction

Platform.NExT Basic Script Language Programming Guide Contents

The contents of the compatible WWB.NET™ Basic Scripts (Visual Basic.NET) programming contents provide developers with all the information needed to create WWB.NET™ "Basic Script" routines.
This guide comes in two parts. The first part relates to the specific instructions inherent to command, methods and API functions for Platform.NExT. The second part contains the syntaxes of standard compatible WWB.NET™ instructions, and is also a part of the **"WinWrap Basic Language"** guide (Copyright Polar Engineering) incorporated in Platform.NExT™.

2. Basic Script in Project

2.1. WinWrap Basic Language

Platform.NExT integrates a WinWrap Basic software component that allows routines to be edited in Basic language within the application.
Some of the main WinWrap Basic features are listed below:

- creates routines WWB.NET language which is compatible with BASIC code (Beginners All-purpose Symbolic Instruction Code)
- extends instructions sets with customized functions methods
- creates dialog boxes by using the Windows standard controls (buttons, checkboxes, groupboxes, listboxes, option buttons, images and text boxes)
- allows references to the assemblies to be inserted and which have been installed with the 4.5.2 framework or developed by starting from framework.

WWB.NET Support

The WWB.NET is the predefined language which is enabled through the '#Language "WWB.NET" directive inserted for default at the start of a script resource. This permits direct access to the .NET assemblies with VB code. The dialog window used for adding reference objects to be used in the code proposes a list of all the selectable assemblies.

The example code assumes that you have added a reference to the System.Windows.Forms assembly:

```
'#Reference #System.Windows.Forms, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089, processorArchitecture=MSIL

'#Language "WWB.NET"

Dim WithEvents t As System.Windows.Forms.Timer

Sub Main
t = New System.Windows.Forms.Timer
t.Interval = 1000
t.Enabled = True
Wait 1
End Sub

Private Sub t_Tick(ByVal sender As Object, ByVal e As System.EventArgs) Handles
t.Tick
Debug.Print Now
End Sub
```

A series of new instructions to render the code compactible to VB.NET programming has been provided especially for this purpose.

Restrictions:

- The signatures of the routines in the objects cannot be renames such as would be allowed with the .NET programming. Le firme delle routine negli oggetti non possono essere rinominate così come consentirebbe la programmazione .NET
- Not more than one event can be linked to the same procedure using a list of "Handles"
- Variable Script Events cannot be managed in WWB:NET scripts
- In WinWrap Basic version 9, the basic script is single thread and the script code must be executed from the thread that created it. For example, a .NET delegate can not be passed to an object that manages this delegate in another thread. A good example of this would be some of the "System.IO.FileSystemWatcher" class functions that create additional threads for monitoring file modifications. In this case the methods of this class will not support other thread calls. This is also the case for the "System.Net.Sockets" class.
- In WinWrap Basic version 9, the basic script do not support nested types, being a type defined in the ambient of another type. For example, it is not possible to use the "System.Net.WebRequestMethods.Ftp" functions.

Unicode Support

The code editor consents you to insert strings in Unicode format. Therefore you can view the Unicode strings in the basic script dialog window or assign Unicode texts to the project's string variables.
UTF8 or UTF16 Unicode files can be read and/or written by writing one of the two new "vbUTF8BOM" and "vbUTF16BOM" constants, added for this purpose, in the first character of a text file in order to determine its code.

2.2. Basic Script as Resources

Basic Scripts are inserted in the project by inserting it as a new resource. To insert a new Basic Script resource, first select the desired point on the "Scripts" group tree structure in the 'Project Explorer Window" where to put it and then click the "New" button from the contextual menu which opens.
This operation is confirmed by the appearance of the new Basic Script resource in the group or the point selected in the Resource structure along with the opening of its code editor window. At this point you can enter the VBA™ code as described in the paragraphs specifically written for this topic.
The resource can then be assigned a Name by using clicking on it and typing in the name to replace the temporary one.

A Basic Script resource must contain the Main (Sub Main) procedure inside. The instructions contained in this subroutine will be executed when the basic script is launched from the project's logic. At the end of the subroutine, without any programmed loop cycles, the basic is terminated and made ready for the next call.
The Main procedure does not have configurable parameters but a parameter can be associated the moment the Basic Script is called. In the command that calls the Basic Script you should find a parameter that can contain several arguments separated by a separator character. The StartupContext.Parameter() function in the basic script can be used for reading the vlaue of the parameter with which the basic script was

called. The vb.net “split” function can then be used to read each argument by passing the character used as the separator.

After a Basic Script resource has been put into execution for the first time, the resource will remain active after the Sub Main() has terminated. Therefore any variables referred to within the resource will be counted for license purposes. If you wish to terminate the execution of a Basic Script completely, you will need to use "Stop" command available in the Platform.NExT "Command List".

The stop command of a basic script resource unloads only those basic resources which are being run in separate threads from memory. The other basic script resources being run in the same thread are only stopped. As a consequence of this, the "Unloading" event is no longer executed following a stop command for those basic script resources which are not in separate threads. In addition to this the variables used by the basic script always remain in use once the basic script has been run at least once.

A Basic Script resource introduced into the project can be put into execution in various ways according to what is required. The following paragraphs illustrate these methodologies.

Execute on command

Execute on command can be easily set by means of using object, menu or resource command lists that provide this possibility.
For example, if you wish to associate a Basic Script routine execution to a button, go to “Common Property Editor”->"Commands"->"Scripts" property and select the Basic resource desired among those proposed on the list referring to the ones introduced, and then define the relevant properties as desired.
The execution on command of a Basic Script can also be established from "Events" resource, whereby the basic routine will be executed upon the occurrence of a determined event and not by a operator command.

The execute script command modes are:

- **Normal**: this is the predefined value, the script is executed in asynchro mode in respect to the call.
- **Synchro**: the script is executed in sychro mode, the call waits until the execution is completed before taking control (the debug is disabled in this mode)
- **Shared**:The script shares the same execution thread of the other scripts that have been started up in the same way. This consents a more limited use of the machine's resources
- **Stop**: the script is unloaded from memory

Execute at startup

A Basic Script routine can automatically be started up at project runtime. In order to use this functionality, you need to access the project's "Startup Script List" property settings.
The Basic Script resource must be selected from those previously inserted in the project using the selection window.

2.3. Basic Scrip in Project

You can use Basic Script routines inside projects in different circumstances and modalities.
It is best to use them in those circumstances where the same operations are not available with other resources or methods: execution/animation properties of symbols, Events and Schedulers for example. The unconditional use of the basic scripts in projects is very handy in the project design time phase but may slow down project execution and consume more of the project's resources.
The basic code can be used in more areas of the project: as resource, directly in the execution properties of drawing objects, as codes associated to the events of objects (alarms, drawings or symbols), in screens.
These functionalities are discussed in more detail in the relating sections.

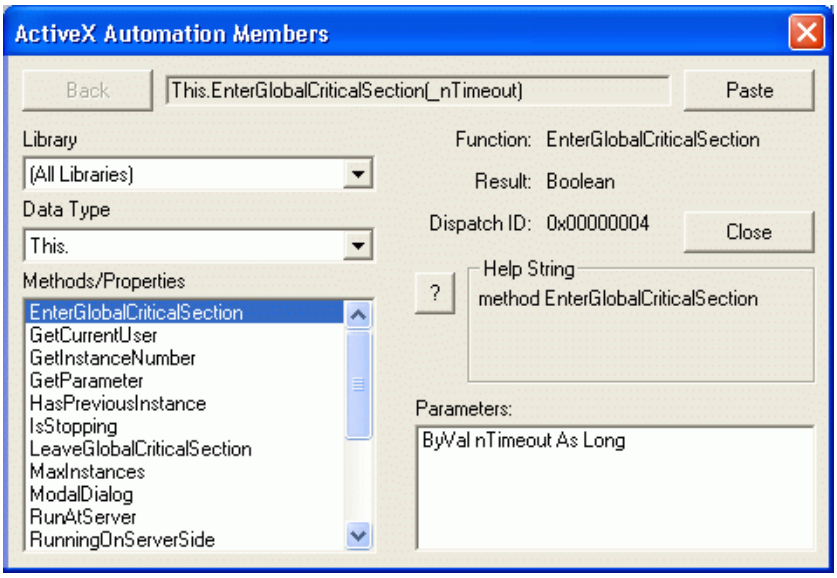
2.4. Code in Basic Scripts and in Symbols

There is a fundamental difference between the functioning of the Basic Script resources and the codes which are inserted inside symbols, screens and alarms. The **"Sub Main"** must exist in Basic Scripts which is executed when called by a basic script. However,the "Sub Main" does not exist in symbols (or screens or alarms), where codes are managed exclusively based on events (SymbolLoading, Click, etc.) which are available in the codes and which were inserted by the programmer. Platform.NExT calls this events only when the symbol (or screen) is active and therefore when the screen is loaded into ram.

2.5. Basic Script Libraries

In addition to the basic functions provided by WinWrap Basic, you can use a series of supplementary functions inside the Basic Scripts which belong to the Movicon libraries to enable project interaction. For example these functions can be used for reading and writing Platform.NExt variables in the Address Space, executing page changes, interaction with the Platform.NExT symbol properties, and much more. The Movicon function libraries are called Basic Script Interface Libraries, and each interface has a collection of sets of specified functions for specific project components.
There are a vast number of Platform.NExT basic interfaces and the list of functions in these libraries can be accessed with the "Browse" button from the basic tool bar (which is available after the a basic code, of any component, has been opened):

The window which opens shows the name of the interface in the "Data Type" box and the methods and properties relating to that interface are listed in the "Methods/Properties" list box:



Browse window Basic scripts functions

Another interface can be selected by using the "Data Type" list box.
To get a function's help just select the function and click the question mark if found in the functions' Browse window, or if inside a basic code just select the function and click the F1 key. The various fields in the Browse window mean:

Back

Returns one step back on the libraries hierarchy scale. Some libraries are set with objects which methods and properties are associated to, for instance when displaying the property of an object and clicking on the back button should return you back to the list of objects set in the library.

Paste

Copies the contents of the box at the side, in the point where the cursor is situated in Script's editor window. If the command is disabled indication will be given by telling you it is impossible to copy the contents in the position indicated by the cursor.

Close

Closes the browse window and the Script's editor returns active.

Library

Allows you to select one of the library proposed by the list. When ActiveX/OCX have been added by using the references, these will also be on the list.

Data Type

Allows you to select a data type from the list proposed. The list refers to the library selected in the Library box.

Methods/Properties

Allows you to select a method or a property from the list proposed. The list refers to the data type selected in the Data Type box.

?

Accesses the help of the property or method selected in the box at the side. Some external libraries, not setup by Progea and enabled through the references, do not install their help files.

Parameters

This displays any eventual list of parameters set for the method selected in the Data Type box.

2.6. Sub, Function, Events, Metods and Properties

Routines can be created in basic scripts which are basically portions of code enclosed in a block and come in two types:

- Sub
- Function

The difference between these two types or routines is very slight. Both can be called by parameter passing. The difference of the "Function" compared to the "Sub" is that is can return a value type set by the programmer (Bool, Int, String, etc.).

Example 1:
The Sub Test is called inside the Sub Main of a Basic Script:

```
Sub Main
Call Test()
End Sub

Sub Test()
MsgBox("Test Sub", vbInformation + vbOkOnly,GetProjectTitle)
End Sub
```

Example 2:
The Function Test is called inside the Sub Main of a Script and the key pressed by the user in the MsgBox is put on log:

```
Sub Main
Debug.Print Test()
End Sub

Function Test() As String
If MsgBox("PTest Function",vbInformation + vbOkCancel,GetProjectTitle) = vbOK Then
Test = "OK"
Else
Test = "Cancel"
End If
End Function
```

The routines described above can be called by other routines, creating nested calls between them. There must always be a **"Sub Main"** in the Basic Script Resource which is the routine automatically executed by the Basic Script when run. However, it is the programmer's job to insert the right codes inside this routine and call any other Sub or Function they themselves have created. Once all the instructions obtained in the Sub Main() have been executed, the basic script is terminated and will have to be called again in order to run it another time. It is also possible to insert loops in the Sub Main() in order to keep the Basic Script running.

Uses Clause

Function within other scripts can be referenced within scripts.
This is done by using the #Uses clause in the script within which you wish to reference functions belonging to other script resources.

A syntax example of the #Uses clause:
'#Uses "*General\PublicFunctions"
By using this syntax you will be able to reference the functions contained in the 'PublicFunctions' script resource contained within a folder called 'General'.

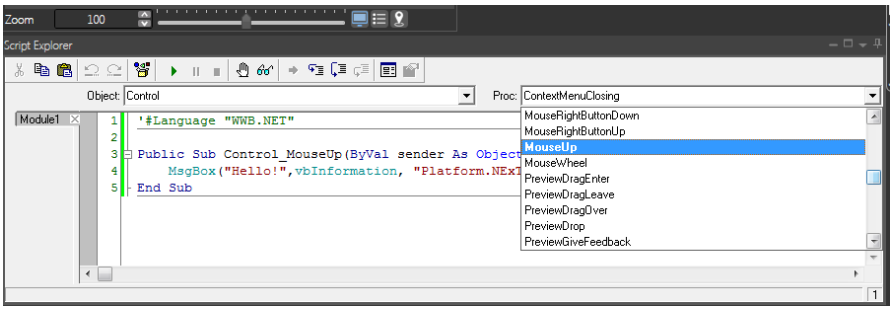
In cases where the called script also resides within a different folder, the syntax in which the #Uses clause will become:

'#Uses "*..\General\PublicFunctions"

where ..\ indicates the root.

Events

There are Sub routines provided by the system (Platform.NExT or any inserted ActiveX component) which are automatically called by the system at the forefront of certain events. These routines, which are actually called "Events", can be inserted in the script and it is up to the programmer to add the desired codes inside them.



Example:
When the following code is inserted inside the a rectangle design's script, a MsgBox with the word "Click" will appear every time the rectangle is clicked on with the mouse:

'#Language "WWB.NET"
Public Sub Control_MouseUp(ByVal sender As Object, ByVal e As System.Windows.Input.MouseButtonEventArgs) Handles Control.MouseUp
MsgBox("Hello!",vbInformation, "Platform.NExT")
End Sub

In this case the Public Sub Control_MouseUp() is an event triggered by the system when performing a Mouse Up on the rectangle, while its internal code must be inserted by the programmer.

A series of events made available by the system can be selected within basic code of screen symbols and objects. These events exclusively concern the component in which the code is being edited being the events in question which regard those triggered by the component. These events are listed in the "Proc" list when the "(Control)" item is selected in the "Object" list as shown in the image above.

Methods and Properties

Those functions that can be inserted from the Platform.NExT basic interface can be exposed as "Methods" and "Properties". The properties identify the characteristics of the object in question, for example its color and title. The method executed a function of that object.

2.7. Server-side Script

The code run on the Server side is executed on the pc in which the Server has been started up (independently from the fact that it has been started up on the client as well). To edit the script code on server side, you will need to enter the code within the Tags, as events generated by the system or as custom methods. The Server side code is loaded upon initialization of the tag in which the script is defined.

When the Tag's code has been set with a breakpoint, a winwrap debug window will open at Server startup even when the method has not yet been invoked.

Server-side Variable Method

Methods for variables can be edited in Movicon Next. This is done by selecting the variable within the Address Space and opting the "Script Explorer" window.
Sub-routines can be created with input parameters.
This method can also be called from the Client after starting up the server side of the project in which the method has been defined by dragging the method onto a client page button for example.

The same result can be achieved by adding a "Call Method" command to the client button and selecting the method to be called in the "OPC UA Browser" card.
The eventual parameters will be displayed in a window after the variable's method has been dragged from the server window onto the button to which the "Call Method" command is to be applied.

As an alternative, by using the "OPC UA Browser" card with the "Data Settings" button you will be able to set the parameter's value.
While associating the method to the button you will be asked to set the method with a value if an input parameter exists.
If you come across an input parameter during the operation in order to associate the method to the button, you will be asked to set a value to associate to the button.
These routines are executed within the server context.

Warning! A syntax will nevertheless be requested with the "Method" prefix added to the name of the routine that has been created.

You can use the "ServerScriptManager.Value" property to read and write the value of the same variable in the method. While the ServerScriptManager.GetTag("NomeVar").value function is used to access another variable.

The variables can then be defined as method type (as data type) in relation to a driver that exposes methods that can be invoked by means of the variable.
Eg:

```
Public Sub Method_Test()'ByVal par1 As String
    Try
        Dim path As String = "c:\temp\MyTest.txt"
        Dim fi As FileInfo = New FileInfo(path)

        If fi.Exists() Then
            fi.Delete()
        End If


the following instruction writes a message in the Log file:

        debug.print "Write text in the .... file"
        'Create the file.
        Dim fs As FileStream = fi.Create()
        Dim info As Byte()
        info= New UTF8Encoding(True).GetBytes("Write text in the ....
file")
        'Add some information to the file.
        fs.Write(info, 0, info.Length)
        fs.Close()
        'Open the stream and read it back.
        Dim sr As StreamReader = fi.OpenText()
        sr.Close()
    Catch ex As System.Exception
        Debug.Print "VarBoolean variable Test method error: " &
ex.Message
    Finally
    End Try
End Sub
```

Server-side variable events

You can also find some events in the variable which can be used by the end user: ServerScriptManager exposes the Init, Terminate and DoEvents events while the AnalogItemState exposes the StateChanged event. For further information please refer to the topic on API Basic Script .

Using Array variables

The elements of array variables can be accessed from within the server's script. The syntax used to do this is:
ServerScriptManager.GetTag("Tag_Array1").Value(0)

However, you must be careful when writing single array elements as the Server will not notify this change to any connected Clients. In order to make sure that the notification takes place properly, you will need to update the whole array variable completely. The best thing to do would be to copy the array in the script's temporary support variable, modify it and then recopy the temporary variable in the server's array variable. For example:

```
Public Sub AnalogItemState_StateChanged(ByVal context As Opc.Ua.ISystemContext,
ByVal node As Opc.Ua.NodeState, ByVal changes As Opc.Ua.NodeStateChangeMasks)
Handles AnalogItemState.StateChanged
   Dim appArray() As Single
   appArray() = ServerScriptManager.GetTag("Tag_Array1").Value
   appArray(0) = 99
   ServerScriptManager.GetTag("Tag_Array1").Value = appArray()
End Sub
```

2.8. Basic Script Variables

PlatformNExt Address Space variables and local Basic Script variables can be used in the Platform.NExT Basic Scripts. The Address Space tags have globalized use the project and the local Basic Script variables are only visible within Basic routines and are destroyed once the Basic Script has terminated. In addition it is also possible to use the Client's “Local” and “System” variables.
The project variables (Platform.NExT Address Space) can be directly used with their name, or by using the "ScriptDocument.GetVariableValue()”, “ScriptDocument .SetVariableValue()” functions with a Script resource, or with the corresponding "Document.GetVariableValue()”, “Document .SetVariableValue()” functions for code edited at screen object level.

The variables internal Basic Scripts have priority over those of the project. Meaning that when using the name of a project variable directly in the basic script code where a variable already exists with the same name, this will be ignored and the already existing one will be set or read. In this case you will have to use the GetVariableValue(), SetVariableValue() basic functions.

Bit type variables, when used directly with their name, are converted into boolean (true or false). This means that a project variable with value "1" will be read as a "true" variable inside the Basic Script and therefore with a numeric value equal to "-1". The "0" value is interpreted as "false" by the Basic Script which will always be a numeric value equal to "0". This mechanism also goes for both the reading and writing of bit variables inside Basic Scripts. The GetVariableValue() and SetVariableValue() basic script functions can always be used to obviate this mechanism. In this way the function's return value will always be the numeric value of the bit, "1" and "0", and not the boolean value.

Example:

When reading the bTest variable declared in the Platform.NExT DataBase as bit and set to "1", the result will be:

```
Sub Main Handles .Main
Debug.Print bProva 'Risultato = -1
Debug.Print ScriptDocument.GetVariableValue("bProva") 'Risultato = 1
End Sub
```

Variables without sign

The WinWrap does not provide the use of variables without sign. Therefore in order to use a Address Space WORD (without sign) type or DWORD (without sign) variable type in the Basic Script it must be converted to avoid causing overflow errors. Overflow error occur when the variable used in the script exceeds the INT type value (ie. 32767 for one Word) because the basic script engine does not manage variables without sign. **To void this problem a UInt32 variable type is always passed to the WinWrap as Long.**

In cases for variables contained inside a folder, for example if there is a "Var1" variable in the "F1" folder, the direct notation from script to refer to this variable would be: "F1_Var1"
Based on the script syntax, it isn't possible to differentiate the variables contained in a folder from those that have names beginning with the same name of the folder.
For example "F1\Var1" and"F1_Var1" would both be referred with the direct "F1_Var1" notation.
Therefore great care must be taken to avoid creating overlaps such as these or otherwise indirect notation should be used to avoid misinterpretation:
ScriptDocument("F1\Var1") for the variable inside the folder and
ScriptDocument("F1_Var1") for the variable in the root.

Array Variables

In cases where Array variable types are used, each array element can be accessed by specifying its number between rounded brackets by using direct tag access syntax. For example, let's suppose that we have created the 'var_01' array tag type in the server's address space, in order to access the tag's elements we would use the following syntax:

```
Var_01(0) = 1
Var_01(1) = 2
```

However, direct syntaxes can only be used in script on the Client side and not in that of the Server side.
Despite this, it is also possible to access Array elements using script functions with the following syntax:

Script on Client side:

```
ScriptDocument.GetVariableValue("Var_01")(0) = 1
ScriptDocument.GetVariableValue("Var_01")(1) = 2
```

Script on Server side:

```
ServerScriptManager.GetTag("Var_01").Value(0) = 1
ServerScriptManager.GetTag("Var_01").Value(1) = 2
```

However, please note that even though the script functions are used to write an Array element, it will not be notified to the Client which will continue showing the old value. In order for the Server to notify the Client of this change, the entire Array variable will have to be set. Therefore, you will need to use a temporary variable within the

script as in the following example where "Folder1\Tag_ArrayFloat" is the Server's Array variable:

```
'#Language "WWB.NET"
Option Explicit
Sub Main Handles .Main
   Dim myArr() As Single
   Dim nLen As Integer
   Dim iCounter As Integer
   nLen = UBound(Folder1_Tag_ArrayFloat)
   ReDim myArr(nLen)
   myArr = Folder1_Tag_ArrayFloat
   For iCounter = LBound(myArr) To UBound(myArr)
      myArr(iCounter) = myArr(iCounter) + 1
   Next iCounter
   Folder1_Tag_ArrayFloat = myArr
End Sub
```

Structure Variables

When using structure variables directly, the delimiter character to use between the name of the variable and the member is the dot ("."). When using the GetVariableValue() and SetVariableValue() functions instead, the separation character to use between the variable name and the member is the colon (":"). For example, if we define the "TagStruct01" structure type variable with a prototype which has "Member_1" and "Member_2" members, to access the variable the syntax will be:

- Direct access (Client side script only): TagStruct01.Member_1 = 1
- Indirect access (by means of Client side functions): ScriptDocument.SetVariableValue("TagStruct01:Member_1", 1)
- Indirect access (by means of Server side functions): ServerScriptManager.GetTag("TagStruct01:Member_1").Value = 1

Attention! When using the direct syntax, only the simple members defined in the prototype's rool can be accessed and not those members defined in the prototype's folder or membered defined as prototype (nested structures). In this case, it will be necessary to use the script function to access nested members.

For example, if we define a "TagStruct01" variable structure type with a prototype which has a "Member_1" member in the prototype's Folder1, the syntax to access the variable will be:

```
ScriptDocument.SetVariableValue("TagStruct01:Folder1\Member_1", 1)
ServerScriptManager.GetTag("TagStruct01:Folder1\Member_1").Value = 1
```

Similarly, when having defined the "TagStruct01" variable structure type with a prototype that has a "MemberStruct" member that has a "Member_1" member in the nested prototype's Folder1, the syntax to access the variable will be:

```
ScriptDocument.SetVariableValue("TagStruct01: MemberStruct \Folder1\Member_1", 1)
ServerScriptManager.GetTag("TagStruct01: MemberStruct \Folder1\Member_1").Value =
1
```

Variables inside Folders

In cases accessing variables contained inside folders, the "_" character must be used as a folder and variable separator or folder and folder separator. When using the GetVariableValue() and SetVariableValue() functions to access variables in folders the "\" character must be used as the separator. For example, when accessing the "Var1" variable inside the "Folder1" Folder, the following syntax must be used:

- Direct access:
  Folder1_Var1 = 1

- Indirect Access (by use of functions)
  ScriptDocument.SetVariableValue("Folder1_\Var1", 1)

  Or with more folder levels:

- Direct access:
  Folder1_Folder2_Folder3_Var1 = 1

- Indirect access (by use of functions)
  ScriptDocument.SetVariableValue("Folder1\Folder2\Folder3\Var1", 1)

When accessing tags directly you need to make sure that you don't define the names of folders and variables that might result as homonyms. For example, If you define "Motor1" variable inside the "Machine1" folder in the root of the address space, and a second variable directly in the address space's root as "Machine1_ Motor1", the script intellisense will have problems distinguishing them from each other because the syntax you would have to use to access them both would be the same: "Machine1_ Motor1". In situations like this, it would be better to use the GetVariableValue() and SetVariableValue() in order to make distinctions between the two tags.

TimeStamp and Quality

Variable TimeStamps and Qualities can be read using script by using direct "VarName_Quality" and "VarName_TimeStamp" notation.
Avoid creating variables with the same name of existing variables with the "_quality" and "_timestamp" suffixes as it may be difficult to distinguish whether the quality of the variable or the variable with that name is to be referred to. The "GetVariableQuality" and "GetVariableTimeStamp" functions can also be used to refer to the variable's quality and timestamp as an alternative to distinguish the differences as described above.

Client system and Local variables

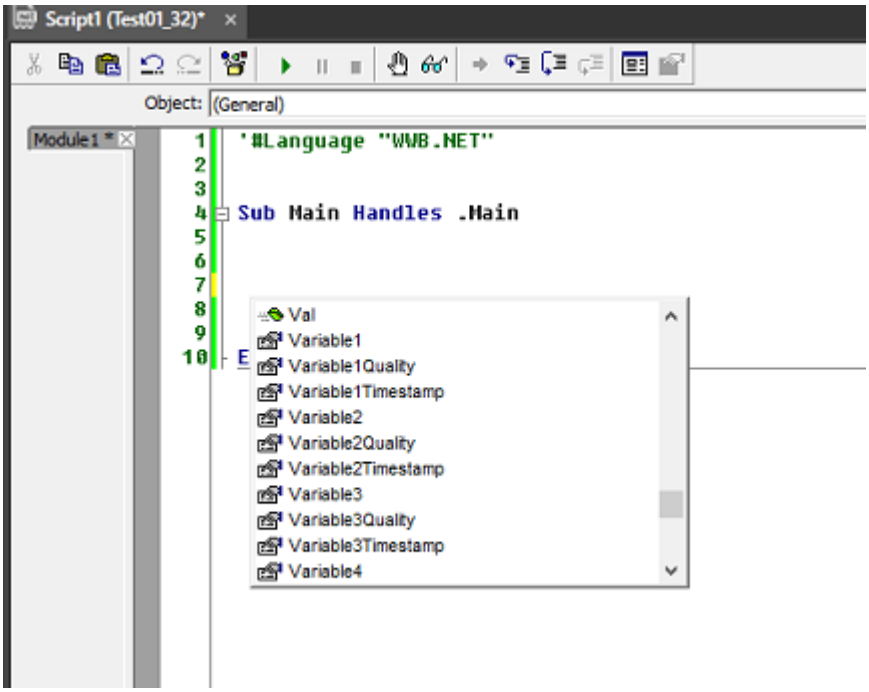
The Client's local and system Tags can be accessed by script code. To access local tags digit "TemporaryVariables" followed by a dot to open the list of selectable members. To access system tags digit "SystemVariables" followed by a dot to open the list of selectable members. For example:

- TemporaryVariables.LocalTag1
- SystemVariables.CurrentUser

Intellisense in Scripts

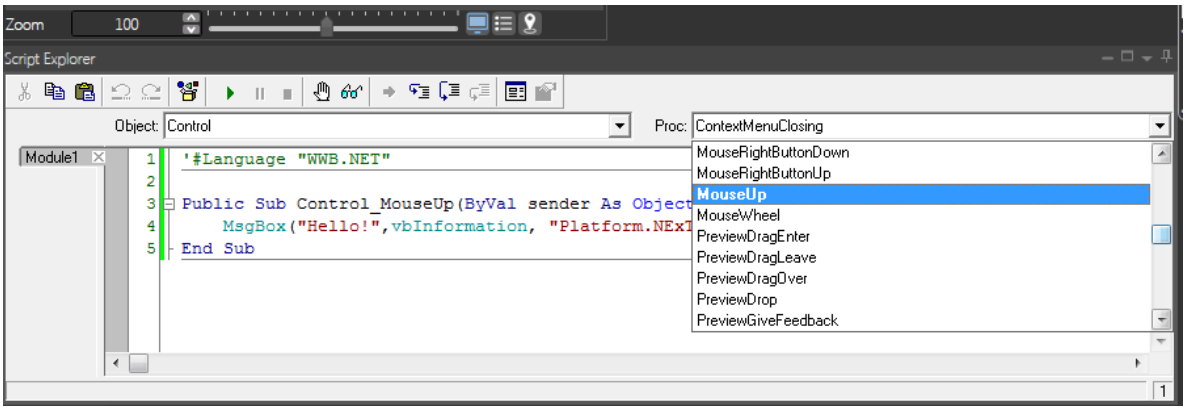
The Basic Script resources are able to identify the project's variables when written directly and in those cases where the tag name is written in blue. This comes in handy

when using the "CTRL+Space Bar" to open the list of available methods and properties as well as the project's tags where you will be able to select the name of a tag instead of having to write it. Therefore when you press the "CTRL+Space Bar" within the script, a list box will appear like the one below allowing you to scroll and select the tag desired:



2.9. WWB.NET™Into Objects

Symbols in screens and screens can contain basic script code. Basic codes inserted in objects are managed differently to those from Basic Script resource, whose execution is controlled in the project's logic. A series of events have been provided in the project that when selected and then inserted inside the basic script editor allow you to insert codes. The basic routines in these events will be executed when the events are triggered during project runtime,
Each object may have different properties, methods and events according its function type. Please refer to the lists in the specific paragraphs to get further details on these.



It is important to remember that the code associated to an event of an object is executed only when the object is loaded into memory and then managed by

Movicon Next. For instance, a drawing associated with a code is executed only when the screen container is loaded.

Please keep in mind that the codes inside drawings are not initialized straight away when the screen page is loaded but only when needed. If a symbol contains the "Loaded" event Movicon NExT is obligated to initialize the basic script code contained in that drawing straight away. This means that page loading is quicker when the drawings associated to it do not contain the "Loaded"" event. However this does not mean that the "Loaded" event should not be used altogether but only when necessary. The project variables (the Platform.NExT Address Space) can be used directly with their name or by using the appropriate functions from the "Document.GetVariableValue()", "Document .SetVariableValue()" for the code edited at screen object level.

The script codes of drawings are loaded only when they are needed and not when the page is being loaded.

2.10. Event Manager

Some framework classes that can be used within the script editor allow events to be handled as shown in the examples below:

The **Timer** class:

```
'#Language "WWB.NET"
Imports System.Windows.Forms
Dim WithEvents t As System.Windows.Forms.Timer
Sub Main
   t = New System.Windows.Forms.Timer
   'Preset Timer
   t.Interval = 1000
   'Start/Stop Timer
   If t.Enabled Then
      t.Enabled = False
   Else
      t.Enabled = True
   End If
End Sub
   'This event is triggered at the end of each set interval
Private Sub t_Tick(ByVal sender As Object, ByVal e As System.EventArgs) Handles t.Tick
  ScriptDocument.SetVariableValue("ActualTime", Format(Now, "dddd dd-mm-yyyy") &
vbLf & Format(Now, "hh:nn:ss"))
End Sub
```

FileSystemWatcher class is another example which handles an event that is triggered every time a file, that it is monitoring, is modified.

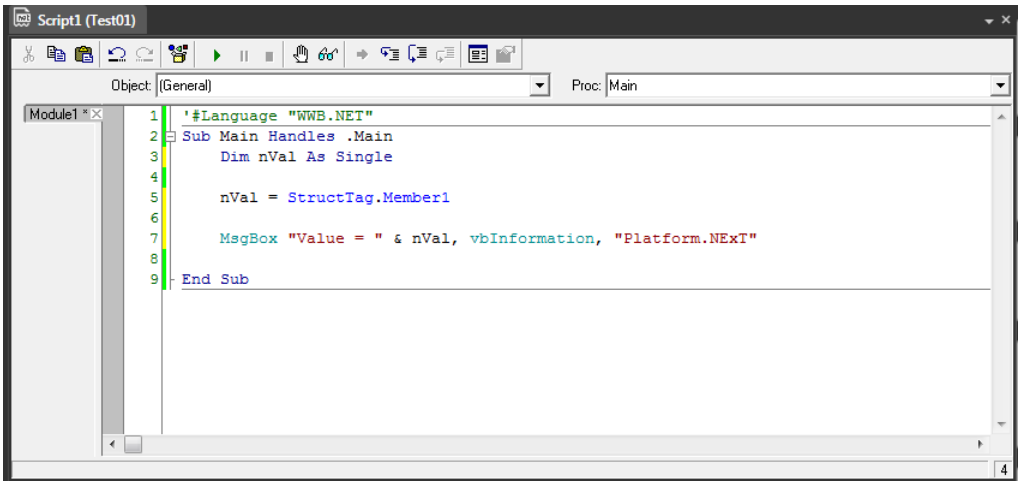
```
'#Language "WWB.NET"
Imports System.IO
Sub Main Handles .Main
Try
```

```
    Dim  path As String= "c:\Test"
   'WatchTest.Log è il file che viene monitorato
   Dim watch As New FileSystemWatcher(path, "WatchTest.Log")
   'Add an event handler  for the "Change" event provided"
   AddHandler watch.Changed, AddressOf watch_Changed
   watch.EnableRaisingEvents = True
   'the loop keeps the script active and enables the event to be triggered that in cases of
the  FileSystemWatcher is run from a distinct thread by the caller
   Do
      DoEvents
   Loop Until ScriptDocument.IsInStoppingMode
Catch ex As System.Exception
   Debug.Print ex.Message
Finally
End Try
End Sub
Private Sub watch_Changed(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim actualVal As Integer
   'VariableChange will be the variable that counts the number of changes
   actualVal= ScriptDocument.GetVariableValue("VariableChange")
   ScriptDocument.setVariableValue("VariableChange",actualVal+1)
End Sub
```

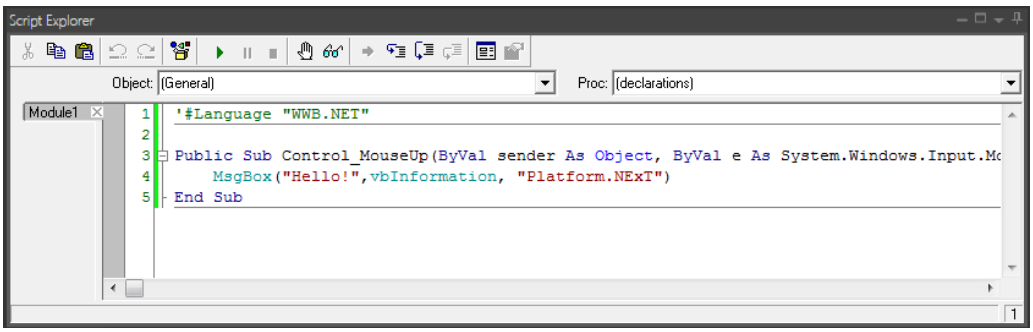
3. Basic Script Editor

3.1. Basic Script Editor

The basic script editor is composed of a window containing an area with a white background in which you can edit codes, plus a series of commands as described below. The window can be resized with the possibility to open more than one window at the same time so that Copy&Paste operations from different projects can be performed faster. To open the editor simply click on the Basic Script resource.



The script code editing of objects and screens is done using the Platform.NExT **"Script Explorer"** window. The contents of this window change dynamically based on the component selected, by show its associated script code.



Both the editing windows shown above provide two drop-down lists, **"Object:"** and **"Proc:"**.

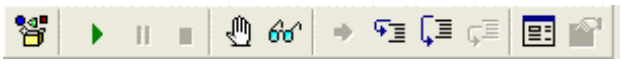
The **"Object:"** list displays a list of basic set objects in Platform.NExT which corresponds to **"(General)", "Control"** and **"ScriptDocument".** The selection of one object in respect to another changes the list of procedures in the **"Proc:"** list available for that specific object.
"(General)" : identifies the procedure programmed for that specific object and the list may change according to the object selected on screen.
"Document": this is an interface that refers to the screen. If for example the "VariableChanged" event is used, variable changes will be notified on the screen.

"Control": identifies a screen object's events and is only available when the code associated the screen object is being edited.
"Entity": identifies the control as framework object.
"Parent": this is an interface that refers to the project.

A list of events available in the **"Proc:"** list is displayed according to the selection made in the "Object:" list.

3.2. Basic Script ToolBar

When opening the basic script code edit window of a resource or from the "Script Explorer", the Basic Script tool bar will be displayed at the top.



Some commands can also be executed from the "Debug Menu".
The keys' functions have been listed below::

Browse

This command is used for accessing the list of functions and properties available in the basic script. The window which appears is called Browse function.

Start/Resume

This command starts the basic script. If the code is referred to a project's basic resource, the contents of the Sub Main procedure are executed. Otherwise, if the code is referred to an object, it enables the events management in that object. It is used to run a check on the syntax used in the written code, in cases where it is not referred to a project resource, which requires the project to be started up in runtime (eg. to read variables) where the code will be executed as well (eg. to read values from DB).

Pause

This command pauses the basic script by positioning the cursor on the line being executed which will automatically be highlighted in yellow.

End

This command aborts the basic script run. The code must be paused beforehand in order for this command to work.

Break Point

This command inserts or removes a breakpoint on the line in which the cursor is present. This function is also made available by clicking outside the extreme left editor window border which coincides with the line in which the breakpoint is to be inserted. The F9 key can also be used as an alternative. The moment in which the basic has to execute a line of code which has a break point, the execution will be automatically placed in pause mode.

Quick Watch

This command returns the result of the highlighted function, in a basic script dialog window or in the debug window (Watch window).

Show Current Statement

This command allows you to position the cursor at the top of the next instruction to be executed in the routine debug phase.

Step Into

This command executes the step into one function at a time, each time it is pressed.

Step Over

This command steps over one function at a time, each time it is pressed.

Step Out

This command executes the all the functions up to the line in which the cursor is situated.

Edit UserDialog

This command opens the tool for creating the basic script's dialog window graphics. When exiting with OK, all that has been created graphically will be translated into code. When positioning and activating the command on this window all the new graphics will be reproduced from the translated basic script code.

Edit Module Properties

3.3. Basic Scrip Debug

Platform.NExT allows the complete debugging of the project's basic script logic, whether being project resources or contents in screen or alarm graphic screens.
In the project design time phase you can debug the Basic Scripts set as Resources only, but you cannot test the scripts associated to objects as these are managed on event. However all these scripts can be debugged during the RunTime phase when Break-Points are inserted into their codes during the programming phase.

Therefore when a script is executed in RunTime, a debug window will open when the execution stops at a Break-Point so that the script can be executed step by step, etc, by using the debug bar at the top.

Break-Points become permanent when inserted inside scripts, meaning that they are saved in the script's properties, and therefore they are also active in RunTime mode. So for this reason we advise you to insert Break-Points for carrying out the debug phase only and remove them straight after the test has been executed.

Some basic script functions verified in programming mode may return with different values when executed in runtime, therefore we advise you to use the debug in the programming phase to check codes roughly and repeat the test by executing the project in runtime to get an accurate test.

4. Basic Script Properties

Each **"Basic Script"** routine inserted as a resource in the "Project Explorer" window can receive in association Properties to setup how they function during a run. In order to this just select the "Basic Script" desired and then change its settings through the Movicon **"Properties Window"**.

General

Cycle Time Tag

This field is used to associate a Tag in which the Basic Script resource's cycle time will be reported. This value is set in milliseconds and will update when the script has terminated from being run. However if there is an infinite loop within the Script, the cycle time value will remain set at zero.

For performance reasons, the Script starts running without waiting until the Cycle Time Tag has been subscribed to the Server. This may mean that the Cycle Time Tag may start updating after the Script has gone into execution.

Current Tag Status

This field is used to associate a Tag in which the Script execution state will be reported. The Script states are managed in bits whereby each Tag bit has the following meaning:

Bit 0: not used.
Bit 1: Error. The bit changes to True each time the Script goes into error.
Bit 2: not used.
Bit 3: not used.
Bit 4: DoEventing. The bit changes to True each time a DoEvent instruction is run within the Script.
Bit 5: Starting. The bit changes to True when the Script is in startup mode.
Bit 6: Stopping. The bit changes to True when the Script is in stop mode.
Bit 7: Running. The bit changes to True when Script is running.

For performance reasons, the Script starts running without waiting until the Cycle Time Tag has been subscribed to the Server. This may mean that the Cycle Time Tag may start updating after the Script has gone into execution.

Thread Priority

A basic resource can be executed with different priorities: High, normal and low:

Lowest: very low priority
BelowNormal: Less than normal priority
Normal: Normal priority
AboveNormal: higher than normal priority
Highest: maximium priority

Force Writing On Server

The property is set to false for default. When set to true, the script will force the writing of the value of a valuable unconditionally to that of the previous one. This will permit same value to be rewritten within a tag.

Execution

Stop Command Timeout

The timeout period for the stop command.

Sleep Time

The sleep time is set in milliseconds and is used to lighten the processor's basic script execution workload. The higher the sleep time, the less the basic script will engage the processor to execute it. As a consequence the script will take longer to execute.

Connection Settings

Session Name

When specifying a name in this field the Script will run in a different Server connection session to the one used by the Movicon Client. When the Script runs in a different session this means that the Server connection parameters will be those defined in the Script resource. However, if the Script is run in the same session used by the Movicon Client, the 'Session Name' parameter is left empty and the Server connection parameters will be the same as those of the Movicon Client which are defined in the project's 'Connection Settings' properties.

Remove Item Delay

This parameter is only handled when the script's "Session Name" property has been set and is used to set the delay time in seconds with which to remove the OPC UA subscriptions of inactive ITEMs.

Remove Item Num.

This parameter is only handled when the script's "Session Name" property has been set and is used to set the number of inactive OPC Items to be removed at each time interval.

Only Secure Connections

This parameter is only handled when the script's "Session Name" property has been set. When this option is enabled only 'secure' Server connections will be allowed.

Fast Sampling Interval

This parameter is only managed when the script's "Session Name" property has been set. This parameter is used to define the rate in which the active and existing variable is to be updated. This parameter is passed to the Server, and any eventual Driver, when the scipt is loaded and the variables go into use. In cases where the script has not been defined with a session, the one defined at project level will be used and as a consequence so will the general value set in the 'Fast Sampling Interval'.

Slow Sampling Interval

This parameter is only managed when the script's "Session Name" property has been set. This parameter is used to define the update rate of a variable that is going out of use but still exists. This parameter is passed to the Server, and any eventual Driver, when the script is unloaded. In cases where the script has not been defined with a session,

the one defined at project level will be used and as a consequence so will the general value set in the 'Slow Sampling Interval'

Disable When Not Used
Sets the Tag as "Inactive" when not in use.

Publishing Interval
Nofication time of Tags towards Server.

Debug

Enable Log
Enabling this selection box, an xml file will be created in the script's folder for the Basic Script routine during runtime. This file will contain printed messages belonging to the script such Debug.Prints. This file will reaming in use until the script stops running due to performance reasons.
This file will always be regenerated at startup.

Enable SysLog
when enabling this property, the messages from the Debug.Print function will also be printed in the project's Historical Log as well.

INDUSTRIAL AUTOMATION SOFTWARE

progea

Progea Srl
Via D'Annunzio, 295
I-41123 Modena
info@progea.com
Tel +39 059 451060

Progea International SA
via Sottobisio, 28
6828 Balerna (CH)
international@progea.com
Tel +41 91 96 76 610

Progea Deutschland GmbH
Marie-Curie Str., 12
D-78048 VS Villingen
info@progea.de
Tel +49 (0)7721 99838 0

Progea North America Corp.
2380 State Road 44, Suite C
Oshkosh, WI 54904
info@progea.us
Tel. +1 (888) 305-2999