Code Presentation Rules

Abstract

This article defines presentation conventions to be used in scripts using Dassault Systèmes product lines scripting capabilities as regards the Automation code layout to improve its readability.

General Considerations

Dassault Systèmes Native Client supports creation of macros in multiples languages. This document globally targets creation of scripting code but its examples and detailed syntaxes are mainly provided for VBScript and Visual Basic for Application languages. Detail of supported levels and languages can be found in the Native Client section of the Program Directory. This document does not cover all topics needed to develop a complex application using for example Visual Basic but most of the rules still apply to this kind of application.

As a general rule, unless overridden by the present document, apply conventions specified by the language vendor like *Microsoft (R) VBScript Coding Conventions* [1] or Style Guide for Python Code [2].

Layout

[l1] A copyright is mandatory for scripts to be delivered within any Dassault Systèmes product.

[l2] A global header is mandatory, it must contain:

Section	Description	Comment
Purpose	Description of what this sample does.	
Assumptions	List of external elements impacting the behavior of this macro, such as state of the session, selected objects.	
Author	For customer provided scripts, empty for samples delivered within a Dassault Systèmes product.	
Languages	Supported development languages.	Valid values are: CATScript, VBScript, VBA, VB.Net, Python and C#
Version	Original version. Available objects may change from one version to the other.	Ex: 3DEXPERIENCER2017x.
Regional Settings	Macros recorded or written for a specific locale may not work for another one. Use language names described in the interactive product documentation.	Ex: English (United States, French (France).

[l3] Indentation should be 4 spaces

[l4] Comment important variables on the same line

[l5] Prefer the ' syntax to the REM keyword for comments

[l6] Align comments concerning a nested block with its indentation

[l7] Insert a header for each internal function or procedure containing purpose, description of parameters and return values.

[l8] Keep lines length shorter than 80 characters

Naming Conventions for VBA and VBScript

[n1] Use verbs or verb phrases for Sub names, in mixed case with the first letter of any word capitalized:

```
Sub DoItBetter()
   ...
End Sub
```

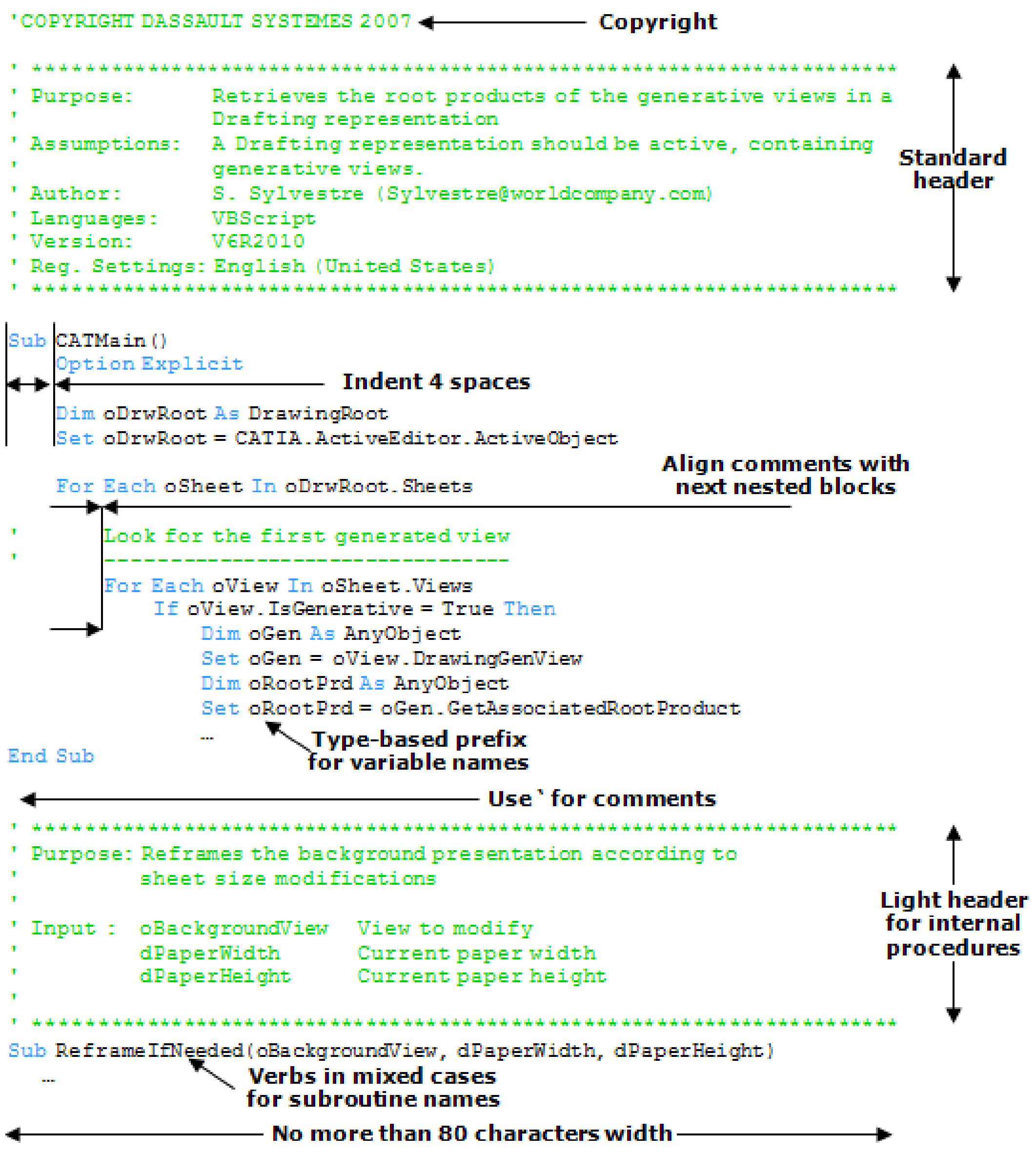
[n2] Use a one lowercase letter prefix for variable names describing the variable type. You may also use your own prefix system (as for example the *Microsoft (R) VBScript Coding Conventions* [1]). Otherwise, the following minimal prefixes are required:

Prefix	Type	Example
b	boolean	bIsUpToDate
d	double	dLength
s	string	sName
i	int	iNumberOfElements
o	objects	oSketch1
c	collections	cDrwViews

[n3] Constant names should be a sequence of one or more words, acronyms, or abbreviations, in uppercase letters, with components separated by underscore "_" characters. For example:

```
MAX_VALUE.
```

Example



```
'COPYRIGHT DASSAULT SYSTEMES 2007

' ***********************************************************************
' Purpose:      Retrieves the root products of the generative views in a
'               Drafting representation
' Assumptions:  A Drafting representation should be active, containing
'               generative views.
' Author:       S. Sylvestre (Sylvestre@worldcompany.com)
' Languages:    VBScript
' Version:      V6R2010
' Reg. Settings: English (United States)
' ***********************************************************************

Sub CATMain()
    Option Explicit

    Dim oDrwRoot As DrawingRoot
    Set oDrwRoot = CATIA.ActiveEditor.ActiveObject

    For Each oSheet In oDrwRoot.Sheets

'       Look for the first generated view
'       ----------------------------------
        For Each oView In oSheet.Views
            If oView.IsGenerative = True Then
                Dim oGen As AnyObject
                Set oGen = oView.DrawingGenView
                Dim oRootPrd As AnyObject
                Set oRootPrd = oGen.GetAssociatedRootProduct
                ...
End Sub

' ***********************************************************************
' Purpose: Reframes the background presentation according to
'          sheet size modifications
'
' Input :  oBackgroundView   View to modify
'          dPaperWidth       Current paper width
'          dPaperHeight      Current paper height
'
' ***********************************************************************
Sub ReframeIfNeeded(oBackgroundView, dPaperWidth, dPaperHeight)
    ...
```

Copyright
Standard header
Indent 4 spaces
Align comments with next nested blocks
Type-based prefix for variable names
Use ` for comments
Light header for internal procedures
Verbs in mixed cases for subroutine names
No more than 80 characters width

References

[1] Microsoft (R) VBScript Coding Conventions
[2] Style Guide for Python Code

History

Version: **4.0** [Oct 2016] Document updated
Version: **3.0** [Apr 2009] Document updated
Version: **2.0** [Jul 2007] Document updated
Version: **1.0** [Sep 2000] Document created

Visual Basic Coding Rules

Abstract

This article presents how to program scripts created in VBScript or Visual Basic for Applications using Dassault Systèmes product lines scripting capabilities.

General Rules and Principles

This document targets scripts developed using Dassault Systèmes scripting languages and mode specifically script written in VBScript, CATScript or Visual basic for Applications. It does not cover all topics needed to develop a complex application using for example Visual Basic but most of the rules still apply to this kind of application.

[g1] Use Option Explicit

It prevents the compiler/interpreter from automatically creating non declared variables, possibly hiding misspelled accesses to existing variables.

It has to be the very first directive:

```
Option Explicit
...
```

If you are using a recorded macro as a basis, remove the valuation of the "Language" variable instead of declaring it like in:

```
Dim Language As String ' Useless: remove
Language="VBScript"    ' Useless: remove
```

Metrics	Statement is present
Objective	Y

[g2] Handle Errors

If errors are not systematically handled, which is often the case when using scripting languages, a lot of unpredictable events can take place before the end user identifies a problem. As a default behavior the interpreter will stop and display an error message box whenever an error is raised.

When you want to take corrective actions on an error, disabling the automatic error handing mechanism using `On Error Resume Next` becomes mandatory.

- Keep as much as possible the automatic error handling mechanism active
- When you deactivate it, isolate its use to places where it's mandatory and explicitly handle all errors meanwhile:

```
Dim CATIA As Object
On Error Resume Next         ' Disable automatic error handling
    Set CATIA=GetObject(,"CATIA.Application")
    iErr = Err.Number        ' For BasicScript parser (Unix)
    If (iErr <> 0) Then      ' Manually handle all errors
        On Error Goto 0      ' Invalidates the Resume Next and clears the error
        Set CATIA=CreateObject("CATIA.Application")
    End If
On Error Goto 0              ' Invalidates the Resume Next and clears the error
```

Metrics Number of lines between an `On Error Resume Next` and an `On Error Goto 0` that are not followed by a block `If (Err.number <> 0) ... End If`

Objective 0

[g4] Declare Supported Languages, And Initial Release

The following comments are mandatory. Valid values for *Language* are "VBScript", "CATScript", "VBA", "VB.NET", or "C#". *Release* identifies the first supported release.

```
' Language: VBScript
' Release:  V6R2010
```

Look in [1] for a complete standard header.

Note that CATScript is kept for compatibility purpose.

Metrics	The two statements are present and contain valid values
Objective	Y

[g5] Always Type Your Variables (VBA/CATScript/VSTA)

Typing objects eases programming and debug:

```
Dim VariableOfTypeX As TypeX
```

An exception is when you need to use a method of an object that takes an array as an argument (see [2]). In this case you won't be able to type your object (VBA/VSTA refuses to compile).

Metrics	**Objectives**
Number of Dim statements without an As statement	0
Number of wrongly defined types	0

Rules for Ensuring Testability

To allow integration in automated test suites, automation scripts must follow specific rules.

[t1] Always Give a Default Value to InputBox

In a test case context, this default value will be used as the result of `InputBox`.

```
Dim sName As String
Set sName = InputBox ("Enter Body Name: ", "", "NEWBODY")
```

When using `MsgBox`, if you store the returned value, it will be set to 0.

Metrics	Number of `InputBox` without a third parameter
Objective	0

[t2] Use Err.Raise to exit on error

In a test case context, an error needs to be identified as such.

Use `MsgBox` for information messages and `Err.Raise` for errors. The first and third parameters must be valuated, like in:

```
If (iShape > cShapes.Count) Then
    Err.Raise _
        9999,_                      ' Number greater than 1000
        "MyMacro",_                 ' Optional: Name of the Script or the sub
        "Shape Number is too big"   ' Explicit error message
End If
```

Don't use:

```
If (iShape > cShapes.Count) Then
    MsgBox "ERROR: Shape Number is too big", VBOkOnly
    Exit Sub
End If
```

Metrics	None (program semantics involved)
Objective	None

Rules for Writing Cross-Platform Scripts

The CATScript language has been kept for compatibility purpose. It is processed by a VBScript engine after removal of typing information. CATIA uses slightly different VBScript interpreters depending on the platform it is being run on. Even though these interpreters have a lot in common, they also have their own behaviors and list of supported features.

[p1] Use only V6 Objects

With Windows you can use the `CreateObject` and `GetObject` functions to access ActiveX objects. This is not possible with Unix.

If a V6 alternative exists, use it. A good example of this is the file system access. VBScript provides access to the file system through a set of ActiveX objects bundled with the Windows Scripting Host (`CreateObject("Scripting.FileSystemObject"`).

This is not portable and can be replaced by the portable **FileSystem**, **File**, and **Folder** objects, accessible from the **Application** object (`CATIA.FileSystem`).

Metrics	Number of `GetObject` or CreateObject
Objective	0

[p2] Use CATIA.SystemService.ConcatenatePaths to Manage Paths Concatenation

Some scripts may require to calculate folder paths using a concatenation of paths from different sources. For example, the following calculates a file absolute path from the content of an environment variable (`ROOT_FOLDER`) and a literal value:

```
Dim sRootPath As String
sRootPath = CATIA.SystemService.Environ("ROOT_FOLDER")
Dim sFilePath As String
sFilePath = sRootPath & "/drw/myData.txt"
```

This is not portable, assuming that the environment variable will contain a valid path for the platform on which the macro runs:

Platform	ROOT_FOLDER value	sFilePath value
Unix	/u/users/me	/u/users/me/drw/myData.txt
Windows	E:\me	E:\me/drw/myData.txt

Not all functions will be able to deal with a path like the second one, containing at the same time '/' and '\' characters as separators.

Use the `ConcatenatePaths` method of the **SystemService** object for that purpose. It always get a path containing the right separators:

```
Dim sRootPath As String
sRootPath = CATIA.SystemService.Environ("ROOT_FOLDER")
Dim sFilePath As String
' here you can use '/' or '\' in the constant part of the path
sFilePath = CATIA.SystemService.ConcatenatePaths(sRootPath, "drw/myData.txt"
```

This results in the following:

Platform	ROOT_FOLDER value	sFilePath value
Unix	/u/users/me	/u/users/me/drw/myData.txt
Windows	E:\me	E:\me\drw\myData.txt

Metrics	Number of strings containing '/' or '\' characters and used in a string concatenation operation
Objective	0

References

[1] Code Presentation Rules
[2] About Microsoft Automation Languages, Debug, and Compatibility

History

Version: **4.0** [Oct 2016]	Document updated
Version: **3.0** [Jul 2007]	Document updated
Version: **2.0** [May 2004]	Document updated
Version: **1.0** [Jan 2000]	Document created