VB Scripting
For CATIA V5

Nick Weisenberger

Expanded eBook Edition
Contents

Copyright Information.......................................................................................................................... 5
Disclaimer............................................................................................................................................ 5
Chapter 1: Getting Started.................................................................................................................. 6
What is a Macro and why do we use them?....................................................................................... 6
What is VB script? .................................................................................................................................. 6
How to Create Macros ....................................................................................................................... 7
How to Add an Existing Macro Library ............................................................................................ 7
How to Open and Run Macros........................................................................................................... 8
Create an Icon for each Macro.......................................................................................................... 8
How to Record a Macro .................................................................................................................... 8
Fundamentals for Creating Custom Macros .................................................................................... 9
VB Script Syntax .................................................................................................................................. 10
Programming Concepts: The CATIA Object .................................................................................... 11
Introductory Example ....................................................................................................................... 13
Chapter 2: Fundamentals.................................................................................................................. 14
Structure ........................................................................................................................................... 14
The Selection Object ........................................................................................................................ 14
Documents versus Windows ............................................................................................................. 16
Subroutines and Functions .............................................................................................................. 16
Save and Save As .............................................................................................................................. 17
Get Workbench ................................................................................................................................ 18
Work with Collections: Count and Item .......................................................................................... 18
Simple Viewer Commands ............................................................................................................. 19
Chapter 3: Building Blocks................................................................................................................. 21
Error Handling .................................................................................................................................. 21
Create a Yes/No Message Box ......................................................................................................... 24
Working with Parameters ................................................................................................................. 24
Step-by-step Tutorials

- Workshop 1: Fundamentals
- Workshop 2: Creating Your Own VBA Modules and Classes
- Workshop 3: Creating a Basic VBA Program from Scratch
- Workshop 4: Objects in CATIA VBA
- Workshop 5: Navigating a Part Document with Error Handling
- Workshop 6: Creating Sketch Geometry
- Workshop 7: Using Forms in CATIA VBA
Copyright Information
Copyright ©2012 by Nick Weisenberger

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing by the author. The only exception is by a reviewer, who may quote short excerpts in a review.

Cover and book design by Nick Weisenberger.

No affiliation with, or endorsed by anyone associated, or in any way connected with Dassault Systemes, Microsoft Corporation, UNIX, or any of their fantastic products.

Disclaimer

Although the author has attempted to exhaustively research all sources to ensure the accuracy and completeness of information on the subject matter, the author assumes no responsibility for errors, inaccuracies, omissions, or any other inconsistencies herein.

The purpose of this text is to complement and supplement other texts and resources. You are urged to read all the available literature, learn as much as you can and adapt the information to your particular needs. There may be mistakes within this manual. Therefore, the text should only be used as a general and introductory guide and not as the sole source for macro programming.

The author also assumes a general understanding of how to use CATIA V5 including geometry creation, various workbenches (mainly Part Design, Generative Shape Design, and Assembly Design). Please refer to the resources in the Appendix for more information about how to use CATIA.
Chapter 1: Getting Started

Through a series of example codes and tutorials I’ll explain how to use and create CATScript macros for CATIA V5. No programming experience is required! This information is not featured in the user help documentation. The purpose of this text is to show beginners how they can approach different problems and for users to rewrite code shown in the examples to suite their specific needs.

There are many CAD engineers, designers, and technicians who want to write macros but simply don’t have time to sit down and learn everything they need to know. This text will cover those core items to help teach beginners important concepts needed to create custom CATScript macros.

What is a Macro and why do we use them?

What is a macro? If you perform a task repeatedly, you can take advantage of a macro to automate the task. A macro is a series of functions, written in a scripting language, that you group in a single command to perform the requested task automatically. Macros use programming but you don’t need to be a programmer or have programming knowledge to use them (though it definitely helps).

Macros are used to save time and reduce the possibility of human error by automating repetitive processes, standardization, improving efficiency, expanding CATIA’s capabilities, and by streamlining tasks.

What is VB script?

VBScript is a subset of the Visual Basic Programming language (VBA). All elements of VBScript are present in VBA, but some VBA elements are not implemented in VBScript. The result of the slimming down process is a very small language that is easy to use. In general, script languages are easier and faster to code in than the more structured, compiled languages such as C+ and C++. Code specific to CATIA is saved as a .CATScript file.

CATIA V5 running on Windows can be automated with any application which can connect to Windows Component Object Model (COM), including VBA
(Excel, Word, CATIA, etc.), VBScript, JavaScript, Visual Basic 6.0, Microsoft Developers Studio.NET, and others. For CATIA V5 running on UNIX, emulators allow for VBScripts to be run with no interface building tools. Some CATScripts from this text may work under UNIX OS but not all due to differences between the two systems. However, this will not be covered in this text.

**How to Create Macros**

Macros are created by two primary methods:

1. Macro recorder
2. Write custom code with the macro editor

For macros recorded in a folder or in a CATPart or CATProduct, Dim statements (declarations) will be recorded for CATScript but not for MSVBScript. For macros recoded in a .catvba library, “MS VBA” is the only choice.

To create a CATScript, simply open NOTEPAD and PASTE in the code from one of the examples herein. Then save the file with your desired name and type extension CATScript instead of txt.

**How to Add an Existing Macro Library**

Use the following steps to setup an existing macro library:

1. Go to Tools>Macro>Macros
2. Click "Macro libraries…"
3. Make sure the Library type is set to "Directories" then click "Add existing library…”
4. Browse to "C:\MyCatScripts" or wherever your catscripts are saved then click ok.
5. Close the macros libraries window. You should see a list of .CATscript files. You only need to do this once as the library should load even after restarting CATIA.
How to Open and Run Macros

There are multiple ways to open the macros window:

1. Go to Tools>Macro>Macros
2. Use the Macros toolbar
3. Keyboard shortcut: Alt+F8
4. Create your own icon for each macro

If the macro editor cannot be opened you should talk to your system administrator because it probably has not been installed. No extra license is required to run it.

Create an Icon for each Macro

To create an icon for each macro go to Tools>Customize>Commands tab>scroll down to Macros>click "Show Properties." Under Commands, select the macro then click the “…” box. The icon browser pops up. There are over 6,000 unique icons to choose from. Select one then hit close. Finally, drag and drop the .CATScript file from the command window to whatever toolbar you would like the icon to appear on (such as “Graphic Properties”). Now you can click the Icon to run your macro! You can also setup custom keyboard shortcuts as well from within the same window.

How to Record a Macro

One method for creating macros is by recording your mouse actions. A few things to keep in mind when recording a macro:

1. DON’T: Switch workbenches while recording a macro.
2. DON’T: Record more than is absolutely necessary.
3. DON’T: Use the UNDO button when recording a macro.
4. DO: Be aware of CATSettings when recording.
5. DO: Exit sketches before stopping recording.
6. DO: Check each macro after it’s recorded.
Always UNDO what you just recorded and run the macro. If the macro works from within CATIA and repeats what you just did, then the macro obviously works fine. If it does NOT work from within CATIA, you need to fix it. If it does NOT work from within CATIA it will NOT work once you cut and paste it into a VB application.

Look through the recorded macro. Many times extra lines of code are added which are not necessary. This is based on the order of steps that you do as you record the macro. These unnecessary lines can be removed.

For example, if I record a macro to zoom in and then zoom out I might get the following code:

```vbnet
Dim viewpoint3D As Viewpoint3D
Set viewpoint3D = viewer3D.Viewpoint3D
Viewer3D.ZoomIn
Set viewpoint3D = viewer3D.Viewpoint3D
Viewer3D.ZoomOut
Set viewpoint3D = viewer3D.Viewpoint3D
```

Notice how the “Set Viewpoint” command appears multiple times? This is unnecessary in this situation. The viewpoint only needs to be set once after the Dim statement. Did I lose you yet? Don’t worry; setting and declaring will be explained in more detail in the upcoming pages.

Often times you might record a macro with a CATPart active and open it in its own window. All goes smoothly and the macro replays fine. Then, the next day you replay the macro again but this time you may have some other document type open or maybe a part is open but it is in a product assembly. Usually, the macro will fail because when the code was recorded a part was the active document but now it is not. This is one advantage to writing custom code and knowing the fundamentals of VB scripting.

**Fundamentals for Creating Custom Macros**

A CATIA VB program or “macro” consists of a “Subroutine” named CATMain(). Variables are “dimmed” (declared) and then “set” (given a value). Variables are “dimmed” (declared) as a type. A type is either a “primitive” type
(single, double, integer, string, etc.) or an object type (more complex). Strings are especially useful because they hold text. Message boxes are frequently used to display strings to users while the program is running. Looping is often used to perform iterative actions

- **Object** – An entity (in CATIA or VB). Points, Pads, Parameters, etc. are all examples of CATIA objects.
- **Property** – A characteristic of an object. For example, the name of a PartDocument is a property of that object.
- **Method** – An action that an object can perform. For example PartDocument.SaveAs() is an action that the object can perform.
- **Collection** – A group or list of similar objects which are put together for a specific reason.

Object oriented programming came about due to the need to represent more complex ideas within a program. For example, you could say that a person is described by his height, weight, and hair color, and that every person has certain actions that they can perform, such as walking, eating, and sleeping. These “properties” and “methods” make up the “class” called “Person.” Objects of this class can then be used in a program to represent individual people.

**VB Script Syntax**

Syntax is defined as the ordering of and relationship between the words and other structural elements in phrases and sentences. Each scripting language is composed of its own syntax. Learning the syntax of each programming language is crucial to creating successful macros. Here are some of the key features of VB Script’s syntax:

- **Indentation:** Indent or outdent script to reflect the logical structure and nesting of the statements.

- **Parentheses:** To achieve the desired result and to avoid errors, it is important to use parentheses () correctly in any statement.

- **Text Strings:** When a value is entered as a text string, you must add quotation marks before and after the string.

- **Case Sensitivity:** By default, VBScript is not case sensitive and does not differentiate between upper-case and lower-case spelling of words, for example, in variables, object and method names, or constants.
**Spaces:** Add extra blank spaces to your script to improve clarity. These spaces are ignored by VBScript.

**Comments:** Add comments to your statements using an apostrophe ('), either at the beginning of a separate line, or at the end of a statement. It is recommended that you add comments wherever possible, to make your scripts easier to understand and maintain, especially if another user has to make changes to it later on down the road.

**Naming:** Avoid naming conflicts. Two variables cannot have the same name.

To return a single quotation mark which does not indicate a comment, needed in a formula for example, you’ll have to use the Chr function. Chr() is a built-in VBA function that returns the character that corresponds to the numeric value of its argument, using the ASCII coding scheme. If you provide Chr with an integer value (such as 39) it will report back the character that corresponds to that value. The ASCII value of 39 is the single quote mark. Chr(34) is for the double quote mark.

**Programming Concepts: The CATIA Object**

The CATIA object is usually the first object that is referenced in any CATIA macro. This object represents the CATIA application itself, from which the macro is run. The CATIA object has many properties. For instance, it has a property called “FullName” which is a string. Another property is called “ActiveDocument”. This property is an object itself, and even more specifically, it is a Document object. More about that later.

The CATIA object is dimmed or “declared” as it exists by default. The purpose of declaring and setting variables is to hold the properties of an object. Variables that hold objects require the “Set” keyword. The properties of objects can be accessed using the notation: Object.Property. Here is an object property example using the FullName property:
Sub CATMain()
    Dim strFullName as String
    strFullName = CATIA.FullName
    Dim doc1 as Document
    Set doc1 = CATIA.ActiveDocument
End Sub

The “Dim” command is used to allocate a variable of any type. Primitive variables are populated with a value. In this case, the “Set” command is not used. For object variables, the “Set” command is used to “point” the variable to the object. Within the program, the variable then represents that object unless it is “Set” to a different one.

The Document object has a string property called “FullName”, which returns the file path of the document. You can “grab” the FullName property and display its value or you can access this property more directly, as shown in the following examples.

Remember, a single quotation is used to display comments or notes. I will use these to help explain each code step-by-step. It’s a good idea to add comments to your own code to help you remember what your thought process was or in case another user needs to edit your code they won’t be completely lost.

' Display the file path name in a message box
Sub CATMain()
    Dim doc1 as Document
    Set doc1 = CATIA.ActiveDocument
    MsgBox doc1.FullName
End Sub

-or-

12
Sub CATMain()
    'Display a message box with the full name of the document
    MsgBox CATIA.ActiveDocument.FullName
    'Display a message box with the number of selections in the document
    MsgBox CATIA.ActiveDocument.Selection.Count
End Sub

Introductory Example

The following code will display a message box with the text "Hello". strHello is a variable declared as a string (or text) object. strHello is then defined. Every Sub must end with "End Sub".

Sub CATMain()
    Dim strHello As String
    strHello = "Hello"
    MsgBox strHello
End Sub

Congratulations! You’ve just created your first CATIA V5 macro! You’re on your way to automating complex and time consuming tasks. We’ll continue with more programming fundamentals in chapter two.

Workshop Tutorial 1: It is recommended that you now flip to the back of this book and walk through workshop number 1, a step-by-step tutorial which will walk you through some of the concepts just explained in chapter one.
Chapter 2: Fundamentals

Structure

There are three main CATIA objects: Document, PartDocument, and ProductDocument. All three are classes, but both PartDocument and ProductDocument are classes that inherit properties and methods from the Document class. Therefore, the PartDocument and ProductDocument class have in common all the properties and methods that they inherit from the Document class, but they each have their own unique properties and methods as well. The specification tree of a part document roughly corresponds to the underlying programmatic structure of the part object, but not exactly.

The Selection Object

To interactively access properties of an object in CATIA V5 you would simply right click on the object and select properties. The object has to be selected first. In the CATIA object model, there is a special object called “selection” which belongs to different documents. This Selection object is a container that holds anything that is currently selected in a session of CATIA V5. It can be accessed like this:

```vbscript
Dim oSelection As Selection
Set oSelection = CATIA.ActiveDocument.Selection
```

If nothing is selected then the selection is empty. If one or more objects are selected then the selection contains those one or more objects. To add an element to the selection:
Dim oSel As Selection

Set oSel = CATIA.ActiveDocument.Selection

oSel.Clear

oSel.Add(ObjectToAdd)

A good practice is to always clear the selection before and after you use it. To check what has been selected you could use the following code, which will loop through all selected objects and display the name in a message box for each one:

For I = 1 to oSel.Count
    MsgBox oSel.Item(i).Value.Name
Next 'i

You can search through selections by several different methods, including searching by name, type, color, etc. Once the search command has been issued, you then need to loop through the selection object to get the items that have been found.

• To look for all object named “pad”: objSel.Search "Name=Pad.*, all” Or objSearch("Name=Weld_Center*,all")
• To look for all points: objSel.Search “Type=Point, all”
• To look for items colored red: objSel.Search “Color=Red, sel”
• To look for items on layer 10: objSel.Search “Layer=10, all”

The selection object can also be used for a variety of other tasks:

• Copy elements: objSel.Copy
• Pasting elements: objSel.Paste
• Delete elements: objSel.Delete
• Change graphical properties:
Dim VisProps As VisPropertySet

Set VisProps = objSel.VisProperties

'gets the color
VisProps.GetRealColor(255, 0, 0)

'sets the color
visProps.SetRealColor(255, 0, 0, 1)

VisProps.SetShow(...) 'set show or hide

Documents versus Windows

There are two main collections under the CATIA application object:

Documents - There are many types of documents that are used in V5: CATPart, CATProduct, CATProcess, CATDrawing, CATAnalysis, etc. These are all housed in the Documents collection and they contain all geometry, process, and product information.

Windows – This collection contains information about how the data from the documents collection will be seen in the CATIA window. It controls items such as:

- Are the parts shaded, wireframe, etc?
- What orientation is the part being viewed in?
- How close or far away is the zoom level on the part?
- What color is the background screen?

Everything in this collection has to do with HOW the data is seen.

Subroutines and Functions

Subroutines and functions are good for encapsulating code that needs to be called repeatedly. Functions “return” a value, subroutines do not. Arguments may be passed in as “ByRef” or “ByVal” (ByVal is the default). The following code is
another example of how to display a “Hello” message box using multiple sub statements.

Sub CATMain()
    Dim strMessage as String
    CallMe strMessage
    MsgBox strMessage
End Sub

Sub CallMe(ByRef strMsg As String)
    strMsg = "Hello"
End sub

Save and Save As

The Document object includes the “Save” and “Save As” methods. The “Save” method takes no arguments and returns nothing.

    CATIA.ActiveDocument.Save

The “Save As” method takes a string as an argument but returns nothing.

    Dim doc1 as Document
    Set doc1 = CATIA.ActiveDocument
doc1.SaveAs "C:\Example.CATPart"
Get Workbench

The Document object also has methods such as the “GetWorkbench” method. The GetWorkbench command takes a string as an argument and returns a Workbench object. To load the structural workbench:

```vbscript
Dim doc1 as Document
Set doc1 = CATIA.ActiveDocument
Dim workbench1 as Workbench
Set workbench1 = doc1.GetWorkbench(“Structural”)```

Work with Collections: Count and Item

Collections are special kinds of objects that hold a list of objects of a certain class. The property “Count” and the method “Item()” are frequently used on collections in CATIA. For example:

```vbscript
Sub CATMain()
    ’Count the number of open documents
    Dim docs1 as Documents
    Set docs1 = CATIA.Documents
    MsgBox “The number of open documents is “ & docs1.Count
    ’Accessing the first Document object in the collection
    Dim doc1 as Document
    Set doc1 = docs1.Item(1)
    MsgBox “The FullName of the first document is “ & doc1.FullName
End Sub```
Simple Viewer Commands

There are a few steps and methods to change the viewpoint of a CATIA document. First, we need to access the 3D viewer from the current CATIA window:

```vba
Dim objViewer3D As Viewer3D
Set objViewer3D = CATIA.ActiveWindow.ActiveViewer
```

Next, we need to access one of the “Camera” objects from the current document. A camera object is a static version of the window viewer object:

```vba
Dim objCamera3D As Camera3D
Set objCamera3D=objCATIA.ActiveDocument.Cameras.Item(1)
```

Unlike CATIA V4, in V5 the views (or Cameras) for a particular document are saved in that document. So, to change the view of a document we need to set the 3DViewer viewpoint to the Camera viewpoint:

```vba
objViewer3D.Viewpoint3D = Camera3D.Viewpoint3D
```

Once you access the ActiveViewer of the current window you can also control many other display properties:

```vba
Dim objViewer As Viewer3D
Set objViewer = CATIA.ActiveWindow.ActiveViewer

'Translating the view
objViewer.Translate(translationVector)

'Rotating the view:
objViewer.Rotate(axisOfRotation, rotationAngle)
```
'Creating a new view (or Camera):

objViewer.NewCamera()

'To reframe a window

objViewer.Reframe()

'To Zoom In

objViewer.ZoomIn()

'To Zoom Out

objViewer.ZoomOut()

Turn off the specification tree:

Dim objSpecWindow As SpecsAndGeomWindow

Set objSpecWindow = CATIA.ActiveWindow

objSpecWindow.Layout = WindowGeomOnly

'This shows both the spec tree and the geometry WindowSpecsAndGeom

'This shuts off the geometry and shows only 'the spec tree WindowSpecsOnly

Go to the back of the back and walk-through Workshop Tutorial 2.
Chapter 3: Building Blocks

Now it’s time to build upon the fundamentals we’ve learned to make our programs more complex and robust.

Error Handling

When writing CATScripts, it’s a good idea to avoid all errors at all cost. Don't let your users think, they are "only users". Write them out of the equation. It is important to debug your programs to avoid errors & program crashes. Trust me, it’s not much fun if another user runs my code and tells me “Hey, your code doesn’t work.” Thus, my goal is to never have a code break because of a run-time error. If you can't avoid an error then use some form of error handling.

Even if the program you write has the correct syntax, it may still encounter a “run time” error as it is running. This is why error handling is necessary. For example, the following code would produce an error if the “ActiveDocument” in CATIA was a product document and not a part document.

```vba
Sub CATMain()

    Dim pdoc1 as PartDocument

    Set pdoc1 = CATIA.ActiveDocument

    MsgBox pdoc1.FullName

End Sub
```

The program would stop at the third line of the above code. By using the statement “On Error Resume Next”, the program will ignore errors that are generated and proceed through the subsequent lines. However, in this example where an error is ignored and the program proceeds, another error is generated on the next line. This is because the variable “pdoc1” is “Nothing.” It wasn’t successfully “Set” in the previous line.
On Error Resume Next
Dim pdoc1 as PartDocument
Set pdoc1 = CATIA.ActiveDocument
MsgBox pdoc1.FullName

The behavior of the program can be made even more intelligent by using the “Err” object, which is an object that exists in every VBA program. The Err object holds information regarding errors that have been generated. The “number” property of the “Err” object indicates whether an error has been generated or not.

Sub CATMain()
On Error Resume Next
    Dim pdoc1 as PartDocument
    Set pdoc1 = CATIA.ActiveDocument
    If Err.Number = 0 Then
        MsgBox pdoc1.FullName
    Else
        MsgBox "Active document is not a part document!"
    End If
End Sub

Calling the “Clear” method on the Err object sets the “Number” property of the Err object to zero (“clean slate”, as they say).
Sub CATMain()

On Error Resume Next

' Insert various programming statements here along with
' logic and conditionals that deal with errors that are
' generated. Assume Err.number is not zero after these
' statements run

Err.Clear

' Proceed with the error number cleared

' (Err.number is now zero)

End Sub

The command “On Error Goto 0” sets the execution of the program so that
errors aren’t ignored for the subsequent lines.

Sub CATMain()

On Error Resume Next

'=====================================  
' If a runtime error occurs here, the program
' proceeds  
'=====================================  

On Error Goto 0

'=====================================  
' Now when a runtime error occurs, the program
' will stop  

End Sub
Create a Yes/No Message Box

Use the following code to create a pop-up message box with the option to click Yes or No:

```vbnet
If MsgBox("To export results to PowerPoint click Yes. Otherwise click No.", vbYesNo) = vbYes Then
Else
End If
```

Working with Parameters

Many times a VB application is developed to create, read, or modify Parameters. Parameters are all accessed from the “Parameters” collection of a part:

```vbnet
Dim objParameters As Parameters
Set objParameters = objPart.Parameters
```

To create a parameter:

```vbnet
Dim objParm As Parameter
Set objParm = objParameters.CreateDimension("Name","Length",10.0)
```

For Loop

Just like any other part of CATIA, there are multiple methods to accomplish the same task. One example of this in VB scripting is the For Loop. A for-each loop can be used to iterate through most collections in CATIA.
Method 1:

Sub CATMain()

    Dim documents1 as Documents
    Set documents1 = CATIA.Documents

    Dim doc1 as Document

    For Each doc1 in documents1

        MsgBox doc1.Name

    Next

End Sub

Method 2:

Sub CATMain()

    Dim documents1 as Documents
    Set documents1 = CATIA.Documents

    Dim doc1 as Document

    Dim i As Integer

    For i = 1 To CATIA.Documents.Count

        Set doc1 = CATIA.Documents.Item(i)

        MsgBox doc1.Name

    Next

End Sub
Changing the Background

Background Color can be set by passing values to or from an array. One use of changing the background color may be to take a screen capture with a white background for easy printing (and using less ink)! Use this code to change the background color to white:

```
' This array holds the original
ReDim dblBackArray(2)
' background color
objViewer3D.GetBackgroundColor(dblBackArray)
' This array will hold the white color
ReDim dblWhiteArray(2)
   ' (the 3 values of this array are the
   dblWhiteArray(0) = 1
   ' standard RGB values but instead
   dblWhiteArray(1) = 1
   ' of 0-255 the values are normalized)
   dblWhiteArray(2) = 1
objViewer3D.SetBackgroundColor(dblWhiteArray)
' To change the color back:
objViewer3D.SetBackgroundColor(dblBackArray)
```
Example: Display the Density of a Part

The “&” symbol works much the same as the concatenate formula in Microsoft Excel by combining multiple elements. For example, if you want a pop up message box to read “The density is 55” where 55 is a variable which will update with the part try this code:

```
Sub CATMain()
  Dim productDocument1 As Document
  Set productDocument1 = CATIA.ActiveDocument
  Dim partRoot As Document
  Set partRoot = productDocument1.Part
  MsgBox "The density is " & partRoot.Density
End Sub
```

Recommended Exercise: Workshop Tutorial 3
Chapter 4: Additional Tools

The Object Browser

Documentation can be found from within the CATIA VBA editor by way of the menu. Go to View>Object Browser or simply hit F2. As an example, let’s say you want to create a pad object. Type in the unknown object type into the search window at the top of the object browser. Look in the third column of results (“member”) for something that makes sense. In this case, “AddNewPad” sounds good. Look in the second column (“class”). This will tell you what object is needed to use the function in the third column. Now look at the bottom of the window. This will tell you how to use the function and what objects are needed.

Function AddNewPad(iSketch As Sketch, iHeight As Double) As Pad

    Member of PARTITF.ShapeFactory

Where Argument 1 is a sketch and I means input. The Pad object must be declared.

Windows Explorer

To open an Explorer window relative to the CATIA Active Document path:

Sub CATMain()

    Set wshShell = CreateObject("WScript.Shell")
    wshShell.Run "explorer.exe " & CATIA.ActiveDocument.Path

End Sub

User Forms

A great way to enhance the experience for your end users and one of the most important features of Visual Basic programs is by creating userforms.
Userforms give you the ability to quickly and easily create graphical user interfaces (GUI) for your macro programs. The foundation of any GUI in a Visual Basic program is a “Form”, onto which various buttons, text fields, list fields, etc. may be dragged onto. Users can input text, choose from a list of options, or click a command button which runs a subroutine that uses the user input options.

Forms, buttons, text fields, etc. can be thought of as special kinds of objects that instead of running “methods” have “events”. In the same way that a method is a function or a subroutine that runs whenever called from within a program, an “event” is a subroutine that runs whenever a user interaction triggers that event. For example, each button on a form has a “Click” event that runs whenever a user clicks on the button.

To create a new UserForm press ALT-F11 to open the VBA editor. In the Project explorer window (top left side), right-click on the project then select Insert – UserForm. Double check to make sure it is named UserForm1 in the properties window. You can drag command button icons to your new UserForm. For more on UserForms see Workshop 7.

**Design Mode**

If you're working with large assemblies, you may want to automatically set each item in the tree to design mode (individually opened parts are typically automatically opened in design mode already):

```vba
Dim productDocument1 As Document
Dim product1 As Product
Dim products1 As Products
Set productDocument1 = CATIA.ActiveDocument
Set product1 = productDocument1.Product
Set products1 = product1.Products
Products1.item(i).ApplyWorkMode DESIGN_MODE
```
Parts and Products

A useful code you may find helpful is to identify whether a file is a CATProduct or a CATPart. One method is like this:

```vbscript
Dim parentFileName As String
parentFileName = products1.item(i).ReferenceProduct.Parent.name

'If the file name contains ".CATProduct" then loop 'through the child parts
If Instr(parenFileName, ".CATProduct") <> 0 Then

Else

In order to go from a known “Product” object to get the “PartDocument” and “Part” that are associated with it you need to go through the GetMasterShapeRepresentation method.

```vbscript
Dim prod1 As Product
'Assume that prod1 gets set to an instance here
Dim partDoc1 As PartDocument
Set partDoc1 = prod1.GetMasterShapeRepresentation(True)
Dim part1 as Part
Set part1 = partDoc1.Part
```

To create a new part in the Assembly, you would use the AddNewComponent method.

```vbscript
Dim objProduct As Product
```

Keep in mind, that when the new part is added, the objProduct object will reference the “Product” level of the Part. The Product class in CATIA VBA has a peculiar property called “ReferenceProduct”. This property returns a Product.
Also, there are two internal names for products inside CATIA V5: the Product Name and the Instance Name. These can both be accessed via the CATIA Object Model using two different properties:

Example: SubAssy1(SubAssy1.1) where the product name is SubAssy1 and the instance name is AubAssy1.1:
Product Name: objProduct.Name
Instance Name: objProduct.PartNumber

Export to Image format

The viewer gives you the capability of exporting the 3D window into a picture format:

```
objViewer3D.CaptureToFile(catCaptureFormatJPEG, "C:\myPicture.jpg")
```

- Other available formats:
  1. BMP
  2. TIFF
  3. CGM
  4. EMF
  5. TIFFGreyScale

The size of the picture is, by default, the size of the active window. You can change the size of the picture with this code:

```
Set window=CATIA.ActiveWindow
H=win.Height
W=win.Width
sLF=Chr(10)
```
win.Height=InputBox("Enter height, reference is upper left hand corner", "Change height", win.height)

If win.Height="" Then

Exit Sub

End If

Then do the same thing for the width and restore original settings at the end.

### 2D Drawing Viewers

There is also a special viewer for Drawings called Viewer2D. The Viewer2D contains many of the same properties and methods of a viewer3D, but it acts upon a CATDrawing document. You can set it the same way as a Viewer3D, but only if the active window contains a CATDrawing. If not, you will get a type mismatch error that you need to use error handling (presented earlier in this text) to trap, otherwise your application might crash.

### Activation State

The following returns the activation state (active or de-active) of a product:

```vbscript
Function GetActivationState(ByVal aProduct As Product) As Boolean
Dim oParameter As BoolParam
Return oParameter.Value()
End Function
```
To update a part use the update function:

```vba
Dim part1As Part
Set part1 = partDocument1.part
Part1.Update
```

Measurement Macros

The measurement tools in CATIA V5 ("Measure Item" & "Measure Between") have not yet been exposed by Dassault Systemes, but there are other workaround methods to do measurements using the CATIA Object Model. The first way is to use a parameter and the "measure" tools which are a part of the formula editor in V5. This can be done by using the parameter and reference interfaces that we looked at earlier. The only difference is that the "measure" functions are input as the driving argument in the CreateFormula method. For example, to create a parameter that will measure the distance between two points, called “MyEndPt1” and “MyEndPt2” you would use:

```vba
Dim objDistance As Parameter

Set objDistance = Parms.CreateDimension("Distance", "LENGTH", 0)

Dim objDistanceRel As Relation

Set objDistanceRel = Part.Relations.CreateFormula("DistanceForm", "", myDistance,_ "distance(`Geometrical Set.1\MyEndPt1`, `Geometrical Set.1\MyEndPt2`)")

MsgBox "The endpoints are " & myDistance.ValueAsString & " apart."
```

Another method to measure elements is with the DMU workbench properties and methods. This requires a license of DMU. Without the license the calls will not work. Examples of how to use these DMU programming objects can be found in the CATIA On-Line documentation for automation.
Dealing with Different Versions of CATIA

Only one version of CATIA V5 can be registered with Windows at a given time. If you have R16, R17, and R18SP2 installed on your PC, only one of these is currently registered. You can find which one is registered by looking at the directory when you register. To automate a specific version of CATIA you first have to register that version with Windows. Here are the steps to un-register or re-register a CATIA V5 version:

1. Reboot your pc
2. Open a command prompt
3. CD to the unload directory of the version of CATIA you want to un-register
   (i.e. “c\program files\Dassault systemes\B15\intel_a\code\bin”)
4. Run the command “cnext-unregserver”
5. Open up the task manager and wait until the “cnext” process stops running (which may take a few minutes)
6. CD to the unload directory of the version of CATIA you want to register
   (i.e. “c\program files\Dassault systems\B16\intel_a\code\bin”)
7. Run the command “cnext-regserver”
8. In the task manager again, wait until the “cnext” process stops
9. Open up VB and check that the correct version is registered
10. Please note this only needs to be done between full releases and not service packs!

Recommended Exercise: Workshop Tutorial 4
Chapter 5: Export to Excel

How to export to Excel

Spreadsheets are used in the world of engineering to create part lists and bills of material. These are typically created in Microsoft Excel. A macro can be created to export data from CATIA into an Excel spreadsheet, quickly automating this process.

Before launching Excel we need to declare all of our objects and variables including the Microsoft Excel application itself, each workbook, each worksheet within each workbook, etc.

```vba
Dim Excel As Object
Dim workbooks As workbooks
Dim workbook As workbook
Dim Sheets As Object
Dim Sheet As Object
Dim worksheet As Excel.worksheet
Dim myworkbook As Excel.workbook
Dim myworksheet As Excel.worksheet
Set workbooks = Excel.Application.workbooks
Set myworkbook = Excel.workbooks.Add
Set myworksheet = Excel.ActiveWorkbook.Add
Set myworksheet = Excel.Sheets.Add
```
Launching Excel from CATIA

There are several methods to go about launching the Microsoft Excel program from within CATIA. One method forces the user to close any open Excel files before running the macro through error handling:

```vbscript
On Error Resume Next
Set Excel = GetObject(, "EXCEL.Application")
If Err.Number <> 0 Then
  Err.Clear
  Set Excel = CreateObject("EXCEL.Application")
Else
  Err.Clear
MsgBox "Please note you have to close Excel", vbCritical
  Exit Sub
End If

Or, we may just want to create a new worksheet and not make the user have to close Excel first (which is more user friendly), thus we will use this preferred code:

```vbscript
On Error Resume Next
Set Excel = GetObject(, "EXCEL.Application")
If Err.Number <> 0 Then
  Err.Clear
  Set Excel = CreateObject("Excel.Application")
End If
```
Screen Updating and Excel Visibility

When you do not want to see your screen follow the actions of your VBA procedure (macro), you start and end your code with the following sentences:

At the start:
\[\text{Application.ScreenUpdating} = \text{False}\]

Then at the end:
\[\text{Application.ScreenUpdating} = \text{True } \text{make Excel visible}\]

To display your completed Excel file, type this line of code at the point in your program when you want Excel to appear.

\[\text{Excel.Visible} = \text{True}\]

Entering Cell Values

To write text in a specific cell, such as the first row's header values, use Excel.Cell (row #, column #).

\['\text{row one headers}\]
\['\text{Cell A1}\]
\[\text{Excel.Cells}(1,1)\text{=}"\text{Part Number}"\]
\['\text{Cell B1}\]
\[\text{Excel.Cells}(1,2)\text{=}"\text{Fasteners}"\]
\[\text{Excel.Cells}(1,3)\text{=}"\text{Name}"\]
\[\text{Excel.Cells}(1,4)\text{=}"\text{Thickness}"\]
\[\text{Excel.Cells}(1,5)\text{=}"\text{Material}"\]
\[\text{Excel.Cells}(1,6)\text{=}"\text{Mass}"\]
\[\text{Excel.Cells}(1,7)\text{=}"\text{Sorter}"\]
You can use a variable to assign a cell number. For example, if you have a For Loop and want the Excel row number to increase by one for each iteration of the loop you might use this:

```vbnet
Dim RwNum As Integer
For I = 1 To 10
    Excel.Cells(RwNum, 4) = getThickness
    Excel.Cells(RwNum, 5) = getMaterial
    Excel.Cells(RwNum, 6) = getMass
    RwNum = RwNum + 1
Next 'i
```

Here is an example using an IF THEN formula. If the name of the CATIA document does not equal PERMANENT_FASTENERS then enter the name of the document and add one to the row count, otherwise enter no cell value and leave the row count the same. “Namebody” is the name of a body within a catpart.

```vbnet
If namebody <> "PERMANENT_FASTENERS" Then
    Excel.Cells(RwNum + 1, 2) = namebody
    RwNum = RwNum + 1
End If
Else
    RwNum = RwNum
```

**Excel Formulas**

You may also need to include formulas to attain the correct cell value. Add a .Formula after the Excel.Cell().
'NOW returns the current date and time

_Excel.Cells(2,13).Formula = "=NOW()"

'Specify date format for NOW formula

_Excel.Cells(2,13).NumberFormat = "m/d/yyyy"

Some Excel formulas require quotation marks. If this is the case you will have to use Chr(34) character representation instead.

_Excel.Cells(1,9).Formula = "=SUMIF(I5:I" & RwNum + 2 & ", " & Chr(34) & "&" & Chr(34) & ")"

Now let's combine a For Loop with a couple of SUM and IF formulas.

_Dim RwNumX As Integer

RwNumX = 1

_For x = 1 to RwNum

_Excel.Cells(RwNumX, 11).Formula = "=SUM($J$2:J" & RwNumX & ")"

_Excel.Cells(RwNumX, 12).Formula = "=IF(J" & RwNumX & ") = 0, M1, K" & RwNumX & ")"

_Next 'x

Defining Excel Constants

Excel contains several values as constants which we will need to define in our code in order for our macro to work correctly. These constants may pertain to the border of a cell, if the text is centered vertically or horizontally, etc. Listed below are a few common constant values. Use Excel’s VBA editor to find the constant value you are looking for.
Const xlCenter = -4108
Const xlAscending = 1
Const xlYes = 1
Const xlSortOnValues = 0
Const xlSortNormal = 0
Const xlTopToBottom = 1
Const xlPinYin = 1
Const xlDown = -4121
Const xlBottom = -4107
Const xlThick = 4
Const xlEdgeBottom = 9
Const xlSolid = 1
Const xlAutomatic = -4105
Const xlThemeColorDark1 = 1
Const xlContinuous = 1
Const xlDiagonalUp = 6
Const xlDiagonalDown = 5
Const xlThin = 2
Const xlNone = -4142
Const xlEdgeRight = 10
Inserting Rows and Columns

Inserting new rows and columns into your spreadsheet is very easy. First, select a cell in the spreadsheet and then specify if you want to insert a row above it as in these examples:

'insert a row at the top for headers

Excel.Cells(1,1).Select
Excel.ActiveCell.EntireRow.Insert

'insert column to the left of the active cell

Excel.ActiveCell.EntireColumn.Insert

'insert column to the right of the active cell

Excel.Cells(1,8).Select
Excel.ActiveCell.EntireColumn.Offset(0, 1).Insert

Sorting the Excel Spreadsheet

Parts listed under a product in CATIA may not be in numerical order. It is very easy to sort these part numbers in Excel.

'sort parts in numerical order G1, then sort by
'fasteners A1

Excel.Range("A:G").Select
Excel.Selection.Sort
Excel.Range("G1"),1,Excel.Range("A1"),,1,Excel.Range("B1 "),1,1,1,False
Deleting Rows and Columns

Sometimes it is helpful to create extra columns in your spreadsheet to help with sorting, but they are not needed and can be removed at the end. To delete a row or column in your exported Excel file, activate the sheet and specify the row or column number to be deleted.

'delete 2nd row because it is empty

Excel.ActiveWorkbook.ActiveSheet.Rows(2).Delete

'delete sorter column G

Excel.ActiveWorkbook.ActiveSheet.Columns(7).Delete

Formatting Excel

After all the formulas and sorting on the spreadsheet is complete, then it’s time to format to get it the way you want it to look. I recommend leaving the formatting until the end. Some formatting examples:

'Select range to format
Excel.Rows("1:1").Select

'change font to bold
Excel.Selection.Font.Bold = True

'change font size to 12
Excel.Selection.Font.Size = 12
More examples:

`change the font type, size, borders, colors, text wrap, etc.

With Excel.Range("A"&"1", "G"&RwNum)
  .Font.Name = "Arial"
  .Font.Size = 9
  .HorizontalAlignment = xlCenter
  .VerticalAlignment = xlCenter
  .ColumnWidth = 25
  .RowHeight = 20
  .BordersLineStyle = xlContinuous
  .Borders.Weight = xlThick
  .Borders.ColorIndex = 1
  .WrapText = True
  .EntireColumn.Autofit

End With

'merge cells in selected range
Excel.Range("Z1:AA1").Select
Excel.Selection.Merge

'wrap text within selected range
Excel.Range("W3","AA3").WrapText = True

'change the font color to red
To insert a diagonal line through a cell use the following code:

```vbscript
Excel.Range("K"&Last,"M"&Last).Select
Excel.Selection.Borders(xlDiagonalDown).LineStyle = xlNone

    With Excel.Selection.Borders(xlDiagonalUp)
        .LineStyle = xlContinuous
        .ColorIndex = 1
        .TintAndShade = 0
        .Weight = xlThin
    End With
```

Change the entire sheet background color to white:

```vbscript
Excel.Cells.Select

    With Excel.Selection.Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .ThemeColor = xlThemeColorDark1
        .TintAndShade = 0
        .PatternTintAndShade = 0
    End With
```

Recommended Exercise: Workshop Tutorial 5
Chapter 6: Export to PowerPoint

Before launching PowerPoint we need to declare all of our objects and variables, including declaring the PowerPoint application. The following code creates a new PowerPoint application if none are open or starts a new presentation if PowerPoint is already being used.

'export to PowerPoint

Dim oPPT As Object

Set oPPT=CreateObject("PowerPoint.Application")

On Error Resume Next

Set oPPT = GetObject("PowerPoint.Application")

If Err.Number <> 0 Then

Set oPPT = New PowerPoint.Application

End If

First, we must Dim or declare the PowerPoint presentation, and slides. We will also maximize the window and prepare to add new, blank slides to the presentation.

Dim oPPTPres As PowerPoint.Presentation

Dim oPPSlides As PowerPoint.Slides

Dim oPPSlide1, oPPSlide2, oPPSlide3 As PowerPoint.Slide

oPPT.WindowState=WindowMaximized

Set oPPTPres=oPPT.Presentations.Add()
Now, we are going to add a title slide to our PowerPoint presentation. There are a number of different slide styles you can add, which is designated by the second number inside the parenthesis after the “Add.” The first number is the slide number. You can also set the header text box of the slide to display the part name of your CATIA object or any other custom text.

'title slide

Set objSlide = oPPTPres.Slides.Add(1, 1)

objSlide.Shapes(1).TextFrame.TextRange.Text = partName

objSlide.Shapes(2).TextFrame.TextRange.Text = "Annotation Data"

It is often useful to add images captured from the 3D model into your PPT. We do this by inserting our image capture macro code within our PowerPoint exporter.

Dim viewer1 As Viewer

Set viewer1 = CATIA.ActiveWindow.ActiveViewer

Dim viewpoint1 As Viewpoint3D

Set viewpoint1 = viewer1.Viewpoint3D

Dim ObjViewer3D As Viewer3D

Set objViewer3D = CATIA.ActiveWindow.ActiveViewer

If you want to add ten slides to your ppt, set i=10 and use a For Loop. Then when you add new slides to the ppt use the i value again to add the new slide to the end of the slideshow.
Dim objSlide As PowerPoint.Slide

Set objSlide = oPPTPres.Slides.Add(i+1, 36)
'capnamer is the name of the catpart


Set pic=objSlide.Shapes.AddPicture(strName, False, True, 50, 5, 576, 350)

Adding headers and footer automatically to your presentation is very easy. Common footers to add include slide number, date, time, presenter name, company name, etc.

objcSlide.HeadersFooters.Footer.Visible = True
objcSlide.HeadersFooters.Footer.Text = partName
objcSlide.HeadersFooters.DateAndTime.Visible = True
objcSlide.HeadersFooters.DateAndTime.UseFormat = True
objcSlide.HeadersFooters.DateAndTime.Format = 1
objcSlide.HeadersFooters.SlideNumber.Visible = True

If you insert image captures into your slideshow they may not be the right size so you might have to adjust them manually. Before you make modifications, it is a good idea to lock the aspect ratio. Use this code to lock the aspect ratio for all the pictures as you insert them:

Pic.LockAspectRatio=True

At the end of your code you will probably want to make PowerPoint visible:

oPPT.Visible =True

Now you can export data from CATIA V5 to Microsoft’s PowerPoint!

Recommended Exercises: Workshop Tutorial 6 and 7
Chapter 7: Additional Information, Help, and Examples

Useful Windows Functions

There are many other useful functions that are available through making calls to Windows libraries. These are available from Microsoft’s website. Some examples:

- Sleep – insert a delay into the VB application
- Killing processes (like CNEXT to instantly stop CATIA)
- Checking for processes (to see if something is running)

Complete Macro Examples

It’s always a good idea to add some notes at the beginning of your code identifying who wrote or modified the code, when the last revision was made, what the code actually is supposed to do, etc. The following are a couple of CATIA macros I wrote and have used throughout my professional career in one form or another. Please feel free to use them for your own purposes and improve upon them!

Return the Parent Name

'Script by Nick Weisenberger
'Revised 11/7/11
'This script returns the parent name of a selected object
'*******************************************************************************

Sub CATMain()

Dim SelectedElement
Set SelectedElement = CATIA.ActiveDocument.Selection

On Error Resume Next

Dim oSelElem
Set oSelElem = SelectedElement.Item(1).Value
Dim info
Set info = oSelElem.Parent.Parent

If Err.Number=0 Then

MsgBox "Parent Name = " & info.Name

Else

Msgbox "You must select an object to return the parent name."

End If

End Sub

Export Spec Tree

'Script by Nick Weisenberger
'Revised November 18, 2011
'this script exports the specification tree to Excel or 'txt file based upon user input
'=======================================

Language="VBSCRIPT"

Sub CATMain()

Dim productDocument1 As Document
Set productDocument1 = CATIA.ActiveDocument

'Input box to select txt or xls
Dim exportFormat As String
exportFormat = Inputbox ("Please choose format to export the tree as. Type either 'xls' or 'txt'")
' Screen Capture - Complete Code

' Script by Nick Weisenberger  
' Last revised January 10, 2012  
' This macro takes a screen shot with a white background  
' and saves it in a folder

Sub CatMAIN()

Dim ObjViewer3D As Viewer3D  
Set objViewer3D = CATIA.ActiveWindow.ActiveViewer  
Dim objCamera3D As Camera3D  

End Sub
Set objCamera3D = CATIA.ActiveDocument.Cameras.Item(1)

'Input box to name the screen capture image file
Dim partname As String
partName = Inputbox ("Please name the image.")

If partName="" Then

MsgBox "No name was entered. Operation aborted.", vbExclamation, "Cancel"

Else

'turn off the spec tree
Dim objSpecWindow As SpecsAndGeomWindow
Set objSpecWindow = CATIA.ActiveWindow
objSpecWindow.Layout = catWindowGeomOnly

'Toggle Compass
CATIA.StartCommand("Compass")

'change background color to white
Dim DBLBackArray(2)
objViewer3D.GetBackgroundColor(dblBackArray)
Dim dblWhiteArray(2)
dblWhiteArray(0) = 1
dblWhiteArray(1) = 1
dblWhiteArray(2) = 1
objViewer3D.PutBackgroundColor(dblWhiteArray)

'file location to save image
Dim fileloc As String
fileloc = "C:\Macro Files\"

Dim exten As String
exten = ".png"

Dim strName As String
strName = fileloc & partName & exten
If MsgBox ("To reframe and automatically switch to ISO view click Yes. To take the image as shown on screen click No.", vbYesNo) = vbYes Then

    objViewer3D.Viewpoint3D = objCamera3D.Viewpoint3D

    'reframe
    objViewer3D.Reframe()

    'zoom in
    objViewer3D.ZoomIn()
    'objViewer3D.ZoomIn()

    'clear selection for picture
    CATIA.ActiveDocument.Selection.Clear()

    'increase to fullscreen to obtain maximum resolution
    objViewer3D.FullScreen = True

    'MsgBox strname

    'take picture with auto ISO view and reframe ON***
    objviewer3D.CaptureToFile 4,strname

    '****take picture as is with NO reframe or iso view****

Else

    zoom in
    objViewer3D.ZoomIn()

    'clear selection for picture
    CATIA.ActiveDocument.Selection.Clear()

    'increase to fullscreen to obtain maximum resolution
    objViewer3D.FullScreen = True
'take picture
objviewer3D.CaptureToFile 4, strname

End If

'************************RESET*****************************

objViewer3D.FullScreen = False

'change background color back
objViewer3D.PutBackgroundColor(dblBackArray)

'turn the spec tree back on
objSpecWindow.Layout = catWindowSpecsAndGeom

'toggle compass
CATIA.StartCommand("Compass")

End If
End Sub

Cycle through 3D Annotation Data

'Script by Nick Weisenberger

'Last revised December 1, 2011

Sub CATMain()

Dim partDocument1 As PartDocument
Set partDocument1 = CATIA.ActiveDocument

partName = Left(partDocument1.name, Len(partDocument1.name) - 8)

Dim oAnnotationSets As AnnotationSets
Set oAnnotationSets = partDocument1.Product.GetTechnologicalObject("CATAnnotationSets")
Dim viewer1 As Viewer
Set viewer1= CATIA.ActiveWindow.ActiveViewer
Dim viewpoint1 As Viewpoint3D
Set viewpoint1= viewer1.Viewpoint3D

Dim ObjViewer3D As Viewer3D
Set objViewer3D = CATIA.ActiveWindow.ActiveViewer


numSets=oAnnotationSets.Count

Dim CurAnnoSet As AnnotationSet
Set CurAnnoSet = oAnnotationSets.Item(1)

numCap=CurAnnoSet.Captures.Count

msgbox "Number of annotation sets is: ",numSets & "
Number of captures is: ",numCap

'error handling
On Error Resume Next

Set partDoc = CATIA.ActiveDocument
Set Part = partDoc.Part

If Err.Number=0 Then

'change text to black

Dim oSel As Selection
Dim visProperties1 As INFITF.VisPropertySet

Set oAnnotationSets = Part.AnnotationSets
Set oSel = CATIA.ActiveDocument.Selection
Set objViewer3D = CATIA.ActiveWindow.ActiveViewer

For IdxSet = 1 To oAnnotationSets.Count
Set oAnnotationSet = oAnnotationSets.Item(IdxSet)
Set oAnnotations = oAnnotationSet.Annotations
For IdxAnnot = 1 To oAnnotations.Count
Set oAnnotation = oAnnotations.Item(IdxAnnot)
If Not oAnnotation Is Nothing Then
With oSel
  .Clear
  .Add oAnnotation
Call .VisProperties.SetVisibleColor(0, 0, 0, 0)
End With
oAnnotation.ModifyVisu
End If
Next
Next

For i=1 To numCap
  Set oCapture = CurAnnoSet.Captures.Item(i)
oCapture.DisplayCapture()
capNamer=CurAnnoSet.Captures.Item(i).Name
Set objViewer3D = CATIA.ActiveWindow.ActiveViewer
Set objCamera3D = CATIA.ActiveDocument.Cameras.Item(1)
  'increase to fullscreen to obtain maximum resolution
  objViewer3D.FullScreen = True
fileloc = "M:\Macro Files\"
exten = ".png"
strname=fileloc&partName&" "&capNamer&exten
  'take picture
  objviewer3D.CaptureToFile 4,strname
Next 'i
End If

'change text back to white
For IdxSet = 1 To oAnnotationSets.Count
  Set oAnnotationSet = oAnnotationSets.Item(IdxSet)
  Set oAnnotations = oAnnotationSet.Annotations
  For IdxAnnot = 1 To oAnnotations.Count
    Set oAnnotation = oAnnotations.Item(IdxAnnot)
    If Not oAnnotation Is Nothing Then
With oSel
  .Clear
  .Add oAnnotation
Call .VisProperties.SetVisibleColor(255, 255, 255, 255)
End With
oAnnotation.ModifyVisu
End If
Next
Next

'error handling
Else
Msgbox "Not a part document! Open in new window."
End If

End Sub

Help Documentation

The file B18\intel_a\code\bin\V5Automation.chm is a useful source for CATIA “Application Programming Interface” (or API) documentation. Navigate to the directory B18\intel_a\code\bin, right-click on the file V5Automation.chm, and choose “Create Shortcut”, which will create a shortcut to this file in this same directory. Move the shortcut to your desktop and double-click on it to open it.

After opening the .chm file, click on the “PartDocument” element of the diagram, which should navigate to the API documentation of the PartDocument class. The API documentation shows the inheritance tree, and a listing of the Properties and Methods of the PartDocument class. Note there is no “Close” method listed. This is because the “Close” method is inherited from the Document class. To see all of the methods available for a particular class, you should check every class that it inherits from up to “AnyObject”. Classes above that are not of any interest. To navigate back to the CATIA object diagram, click on the “Contents” tab, expand the “CAA V5 Visual Basic Reference” node, and select “CAA V5 Objects.”

Online help documentation is another source of API documentation. The Automation Documentation is found under the “Infrastructure” Icon. There is also a special set of On-Line help that is available with the CAA RADE interfaces (it’s on this CD). A RADE license is NOT required to access the help files.
Final Thoughts

Now you know how to write VBScript macros for CATIA V5! You’re on your way to automating those repetitive tasks and impressing your coworkers and bosses.

Good luck and happy programming!

-Nick Weisenberger
About the Author

Nick Weisenberger grew up in Columbus, Ohio and graduated from the University of Toledo, in Toledo, Ohio, with a degree in Mechanical Engineering. An avid traveler, one of Nick’s favorite pastimes is visiting amusement parks all over the country. You can read more about his adventures at http://www.coaster101.com. He has also combined his love of coasters, engineering, and writing in his latest book Coasters 101: An Engineering Guide to Roller Coaster Design. He’s also written a non-fiction story about an engineer’s adventures in China, entitled Nike’s Chinese New Year. Nick currently works as an engineer and resides with his wife near Columbus, Ohio.
Appendix I: Acronyms

The following terms are used throughout this text (in alphabetical order):

- **API**: Application Programming Interface
- **CAD**: Computer Aided Design
- **CATIA**: Computer Aided Three Dimensional Interactive Application
- **COM**: Component Object Model
- **DLL**: Dynamic Linked Library
- **DMU**: Digital Mock-Up
- **GUI**: Graphical User Interface
- **MDB**: Model Based Definition
- **PBD**: Product Based Definition
- **TLB**: Type Library File
- **VBA**: Visual Basic for Applications

Appendix II: Resources

For more information see the following list of resources:

1. VB Scripting for CATIA V5: [http://www.scripting4v5.com](http://www.scripting4v5.com)
2. CAD Systems Help: [http://cadsystemshelp.blogspot.com](http://cadsystemshelp.blogspot.com)
3. Excel Spreadsheets Help: [http://excelspreadsheetshelp.blogspot.com](http://excelspreadsheetshelp.blogspot.com)
4. CATIA V5: [http://www.3ds.com](http://www.3ds.com)
WORKSHOP Tutorials

The following pages contain seven step-by-step tutorials designed to help teach you macro programming in CATIA V5.

Contents

• Workshop 1: Fundamentals
• Workshop 2: Creating Your Own VBA Modules and Classes
• Workshop 3: Creating a Basic VBA Program from Scratch
• Workshop 4: Objects in CATIA VBA
• Workshop 5: Navigating a Part Document with Error Handling
• Workshop 6: Creating Sketch Geometry
• Workshop 7: Using Forms in CATIA VBA
VB Scripting for CATIA V5 - Workshop 1,
Nick Weisenberger, February 2012

WORKSHOP 1

Fundamentals

Language="VBSCRIPT"

Sub CATMain()

Dim productDocument1 As Document
Set productDocument1 = CATIA.ActiveDocument

'Input box to select txt or xls
Dim exportFormat As String
exportFormat = Inputbox ("Please choose format to export the tree as. Type either 'xls' or 'txt'.")

'Input box to enter name of file
Dim partName As String
partName = Inputbox ("Please enter the file name.")
Description

VBScript is a subset of the Visual Basic Programming language (VBA). All elements of VBScript are present in VBA, but some VBA elements are not implemented in VBScript. The result of the slimming down process is a very small language that is easy to use. Code specific to CATIA is saved as .CATScript. In this workshop you will learn some of the basic fundamentals needed to create and run macros.

Outline

1. How to add a macro library
2. How to open and run macros
3. How to create an icon for your macros
4. Create a "hello" message box macro
Why use macros?

Save time and reduce the possibility of human error by automating repetitive processes.

- Standardization
- Improve efficiency
- Expand CATIA capabilities
- Streamline tasks

Macros in CATIA are created by two primary methods:

1. Macro recorder
2. Write custom code with the macro editor
Use the following steps to setup an existing macro library:

Go to Tools>Macro>Macros
Click "Macro libraries..."
Make sure the Library type is set to "Directories" then click "Add existing library…"
Browse to "C:\MyCatScripts" or wherever your catscripts are saved then click ok.
Close the macros libraries window.

After you have created some macros you will see a list of the .CATscript files here.

You only need to do this once- the library should load even after restarting CATIA.
To open and run macros

There are multiple ways to open the macros window:

• Tools>Macro>Macros
• Macros toolbar
• Keyboard shortcut: Alt+F8
• Create your own icon for each macro
Recording a Macro

To record a macro go to: Tools>Macro>Start Recording…
Pick a library, language, and macro name.

To stop recording:
Tools>Macro>Stop Recording or click the stop button
To create an icon for each macro:

Go to Tools>Customize>Commands tab> scroll down to Macros>click "Show Properties"
Under Commands, select the macro then click the "..." box. The icon browser pops up. Over 6000 to choose from. Select one then hit close.
• Finally, drag and drop the .CATScript file from the command window to whatever toolbar you would like the icon to appear on
• Now you can click the Icon to run your macro!
• You can also setup a custom keyboard shortcut as well.
• Code can be sent to other user as a lightweight .txt text file
Hit `alt+F8` to open the macro window. Create a new CATScript macro named `msgbox`. Click the Edit button. Copy and paste the code below. The following code will display a message box with the text "Hello." `strHello` is a variable declared as a string (or text) object. `strHello` is then defined. Every `Sub` must end with "End Sub". Click Save. Close the editor and run your macro. The "Hello" message box should appear.

```
Sub CATMain()
    Dim strHello As String
    strHello = "Hello"
    MsgBox strHello
End Sub
```
This concludes workshop 1. In this workshop you have learned to:

• Create a new macro
• Create an icon for your macro
• How to record a macro
• Create a "Hello" message box
WORKSHOP 2

Creating Your Own VBA Modules and Classes Tutorial

VB Scripting, for CATIA V5 – Workshop 2
Nick Weisenberger, February 2012

WS-17
Workshop 2 – Introduction

■ Description

Although most of the CATIA VBA programming that you will do will involve the use of classes that are defined by the CATIA programming API, you will likely find it useful to define your own classes. In this workshop you will create a custom class in order to demonstrate the fundamentals of VBA object design.

■ Outline

1. Create a new class module in a new VBA library named “Messenger”.
2. Give the Messenger class a simple “Public” property and use the class in a CATIA macro.
3. Enhance the Messenger class so that the “getting” and “setting” of its properties are controlled by “Get” and “Let” methods.
4. Give the Messenger class a method called “Capitalize” that serves to manipulate the string information that is stored in its properties.
Step 1

- Open the VBA editor by hitting Alt + F11. Create a new macro library and VBA project, called "myVBA".
- Double click on your newly created library (which will appear in Current Libraries).
Step 2

• Ensure the project and properties boxes are visible by going to the top menu bar and clicking View > View Project Explorer and View Properties Window
• Right click on VBAProject (myVBA in this example) > Insert > Module.
• Use the (Name) field in the properties box to rename it "Create_New_Part_Document"
Step 3

Right click on the “Modules” folder of the VBA project and select Insert > Class Module.

Rename the resulting class module as “Messenger” by clicking on Class1 then renaming the "Name" field in the properties box.
Step 4

• In the code window enter the line:

   **Public Message As String**

• Create another module on the VBA project. This needs to be a regular module, not a class module. Name it “UseTheMessenger”.

---

WS-22
Enter the code seen to the right into the “UseTheMessenger” module. Note the following:

- A Messenger object is dimmed. Because you created the class module named “Messenger” this class is immediately available in your VBA project.
- The “New” keyword orders the creation of a Messenger object. The variable “oMssgr” is set to this new object.
- The property “Message” gets set to a specific value here.
- The Message property then gets used as it is passed to the message box.

Run the UseTheMessenger module by clicking the play button icon. A message box displaying “Hello” should result in your CATIA Window.
Step 6

• Return to the code window for the class module “Messenger” and make the changes shown below. These changes have the effect of “hiding” the strMessage variable, but then create a read-only property named “Message” whose value is stored in the strMessage variable.

• Now go back to the “UseTheMessenger” module and make the changes shown below and to the right.

```
Private strMessage As String

Property Let Message(MessageIN As String)
    strMessage = MessageIN
End Property
```

```
Sub CATMain()
    Dim oMssgr As Messenger
    Set oMssgr = New Messenger
    oMssgr.Message = "Hello"
    MsgBox oMssgr.Message
End Sub
```
Attempt to run “UseTheMessenger” and note that although the Message property can be “set”, it fails when the property is “gotten”. This is because no “Property Get” method has been defined (although, “Property Let” has been defined).
Step 8

• Return to the code window for the class module “Messenger” and add a “Property Get” method as shown below.

• Run the UseTheMessenger module again. It should work and the message “Hello” should be displayed.

Note: the advantage of strictly defining these “Let” and “Get” methods is that it gives the programmer control over whether a variable is read-only or read-write. Also, the code that is in the “Let” and “Get” methods can contain more complex operations and logic.

```vbnet
Private strMessage As String

Property Let Message(strMessageIN As String)
    strMessage = strMessageIN
End Property

Property Get Message() As String
    Message = strMessage
End Property
```
Assume it would be desirable to keep count of how many times the Message property’s value is changed. To do this, create a private integer variable “iCount”, change the “Property Let” method of the Message property, and add a “Property Get” method for a new property named “MsgChangeCount”.

Private strMessage As String
Private iCount As Integer

Property Let Message(strMessageIN As String)
    strMessage = strMessageIN
    iCount = iCount + 1
End Property

Property Get Message() As String
    Message = strMessage
End Property

Property Get MsgChangeCount() As Integer
    MsgChangeCount = iCount
End Property
Step 10

• Change the “UseTheMessenger” module as shown below. Run it and a message should be displayed saying “Message changed 1 times”.
• Change the “UseTheMessenger” module as shown to the lower right and run it again. A message should be displayed saying “Message changed 2 times”.

Sub CATMain()

    Dim oMssgr As Messenger
    Set oMssgr = New Messenger

    oMssgr.Message = "Hello"

    MsgBox "Message changed " & oMssgr.MsgChangeCount & " times."

End Sub

Sub CATMain()

    Dim oMssgr As Messenger
    Set oMssgr = New Messenger

    oMssgr.Message = "Hello"
    oMssgr.Message = "Hello again"

    MsgBox "Message changed " & oMssgr.MsgChangeCount & " times."

End Sub
• Although the class works presently, there is one area where the class code could be more explicit. The “iCount” variable is incremented by one every time the Message property is changed, but it isn’t clear what value iCount starts at. Testing the code has shown that it does start at zero, however it’s best to be explicit.

• In the code window for the Messenger class module, click the left drop-down menu and choose “Class”.
Step 12

The result of the previous step should be that the text for the subroutine “Class_Initialize” appears in the code window. Enter the code shown below into this sub. This has the effect of setting the value of the iCount to zero when a Messenger object is created with the “New” command.

```vbscript
Private Sub Class_Initialize()
    iCount = 0
End Sub
```
Step 13

- Properties have been defined for the messenger class. Now create a method for this class. This method will capitalize the message that is stored in the Message property. Enter the code seen below in the Messenger class module.
- The function “UCase” is a standard VBA function that takes a String as an argument and returns the same string in all capital letters.

```vba
Private strMessage As String
Private iCount As Integer

Property Let Message(strMessageIN As String)
    strMessage = strMessageIN
    iCount = iCount + 1
End Property

Property Get Message() As String
    Message = strMessage
End Property

Property Get MsgChangeCount() As Integer
    MsgChangeCount = iCount
End Property

Public Sub Capitalize()
    strMessage = UCase(strMessage)
End Sub
```
Step 14

- Change the “UseTheMessenger” code so that it calls this new capitalize method as shown.

- Run the code. Note that the capitalize method has the intended effect of changing the message box text from “Hello” to “HELLO”.

```vbscript
Sub CATMain()
    Dim oMssgr As Messenger
    Set oMssgr = New Messenger
    oMssgr.Message = "Hello"
    oMssgr.Capitalize
    MsgBox oMssgr.Message
End Sub
```
This concludes Workshop 2. In this workshop you have learned to:

• Create a VBA class

• Define a read-only property for the class (only provide a “Property Let” method)

• Define a read-write property for the class (provide both “Property Let” and “Property Get” methods)

• Use the “Class_Initialize” subroutine to assign initial values to class variables

• Define a method for a class that operates on private variables of that class
WORKSHOP 3

Creating a Basic VBA Program from Scratch
Description

In this workshop, as opposed to starting a CATIA VBA program by recording a macro, you will instead insert a new module into an existing CATIA VBA library and type the program statements in manually. You will also see the difference between Subs and Functions, and get experience using “primitive” variable types, arrays, and For loops.

Outline

1. Create a new module in the macro library created in Workshop 2 called myVBA_01
2. Program a routine that creates two integer variables, adds them, and then displays the result in a “message box”.
3. Create a new module with a routine using a Sub
4. Create a new module with a routine using a Function
5. Create a new module with a routine using arrays and a For loop.
Step 1: Create a new module named Workshop2_01

- Right-click on the myVBA project on the Project tree and select Insert > Module.
- Rename the newly created module “Workshop2_01”
Step 2

- In the code window, type the code shown below
- Click **Save** on the **File** menu to save your changes.

```vbnet
Sub CATMain()

'Declaration of variables
Dim intA As Integer
Dim intB As Integer
Dim intC As Integer

'Valuate two of the variables
intA = 2
intB = 3

'Set the value of the third as a sum of the first two
intC = intA + intB

'Create a string to be displayed as output. The function Str() converts the integer intC into a String
Dim strOut As String
strOut = "The sum is " + Str(intC)

'Call a message box to display the output string
MsgBox strOut

End Sub
```
Step 3

- Close the VBA editor.
- Open the Macros menu through **Tools > Macro > Macros**.
- Select the **WorkShop2_01** macro on the list and click the Run button.
- Note that the value “5” is displayed in the message box.
Step 4

- Right-click on the myVBA project on the Project tree and select **Insert > Module**.
- Rename the newly created module “Workshop2_02”.
- In the code window, type the code shown below.

```vba
Sub CATMain()
    Dim intA As Integer
    Dim intB As Integer
    Dim intC As Integer

    intA = 2
    intB = 3

    DisplaySum intA, intB
End Sub

Sub DisplaySum(ByVal int1 As Integer, ByVal int2 As Integer)
    Dim int3 as integer
    int3 = int1 + int2

    MsgBox “The sum is “ + Str(int3)
End Sub
```
Step 5

• Click **Save** on the **File** menu to save your changes.
• Run the macro, this time by using the “Run” button in the VBA editor. This requires that the cursor in the code window is sitting within the “CATMain” Sub.

```vba
Sub CATMain()
    Dim intA As Integer
    Dim intB As Integer
    Dim intC As Integer
    intA = 2
    intB = 3
    DisplaySum intA, intB
End Sub

Sub DisplaySum(ByVal int1 As Integer, ByVal int2 As Integer)
    Dim int3 As Integer
    int3 = int1 + int2
    MsgBox “The sum is “ + Str(int3)
End Sub
```
Step 6

- Right-click on the myVBA project on the Project tree and select **Insert > Module**.
- Rename the newly created module “Workshop2_03”
- In the code window, type the code shown below
- Click **Save** on the **File** menu to save your changes.
- Run the macro, this time by using the “Run” button in the VBA editor. This requires that the cursor in the code window is sitting within the “CATMain” Sub.

```vba
Sub CATMain()
    Dim intA As Integer
    Dim intB As Integer
    Dim intC As Integer

    intA = 2
    intB = 3

    intC = DisplaySum(intA, intB)

    MsgBox “The sum is “ + Str(intC)
End Sub

Function DisplaySum(ByVal int1 As Integer, ByVal int2 As Integer) As Integer
    DisplaySum = int1 + int2
End Function
```
Step 7

- Right-click on the myVBA project on the Project tree and select **Insert > Module**.
- Rename the newly created module “Workshop2_04”
- In the code window, type the code shown to the right
- Click **Save** on the **File** menu to save the your changes.
- Run the macro.

```vba
Sub CATMain()
    Dim intA(3) As Integer
    Dim intB(3) As Integer
    Dim intC As Integer

    intA(1) = 1
    intA(2) = 2
    intA(3) = 3

    intB(1) = 1
    intB(2) = 2
    intB(3) = 3

    Dim i As Integer
    For i = 1 To 3
        intC = DisplaySum(intA(i), intB(i))
    Next
    MsgBox intC
End Sub

Function DisplaySum(ByVal int1 As Integer, ByVal int2 As Integer) As Integer
    DisplaySum = int1 + int2
End Function
```
This concludes Workshop 3. In this workshop you have learned to:

• Create a CATIA VBA program from scratch

• Use “primitive” variable types such as integers and strings

• Use a Sub (subroutine) to encapsulate code that is to be called multiple times throughout a program

• Use a Function in a way similar to a Sub, but when a return value is desired

• Use an array to store a list of primitives (integers in this case) and then iterate through that array using a For loop
WORKSHOP 4

Objects in CATIA VBA
Description

In this workshop you will explore the basics of objects, their properties, and their methods in CATIA VBA programming.

Outline

1. Create a new module in the macro library myVBA_01.
2. Start an “empty” CATMain() Sub.
3. Observe the properties and methods of the CATIA object.
4. Grab the Documents collection and display its Count property.
Step 1

- Create a new module in the myVBA macro library called “CATIAbasicObjects”

- In the code window type “Sub CATMain()” and hit Enter.

- In the CATMain() Sub, type “CATIA.” (notice the period) and note that the “Intellisense” drop-down appears, displaying a list of properties and methods of the CATIA object.

- Note that the “Documents” property of the CATIA object is on the list. This is what is depicted in the CATIA object model diagram below.

NOTE: the object called “Application” is the same as the CATIA object in the VBA view.
Step 2

• Scroll down the Intellisense drop-down and note that the “Windows” property is there as well.
• So “Documents” and “Windows” are both properties of the CATIA object, but they are also objects themselves. Moreover, they are a special type of object: they are collections.
• Erase the text “CATIA.” that you just entered.
Step 3

- Open 3 new parts in CATIA, accepting their default names and save them anywhere on disk.
- Enter the code below into the code window.
- Run the macro using the “Run” button. The count of open documents should be displayed.

```vba
Sub CATMain()
    Dim documents1 As Documents
    Set documents1 = CATIA.Documents
    MsgBox "The number of documents is " & documents1.Count
End Sub
```
Add the code shown to the right and re-run the macro:

Note: The method Item() was used on the Documents object “documents1” in order to retrieve a member of the collection. Then the “Name” property and “FullName” property of the Document object “doc1” were used. These properties return Strings, which were displayed in the message boxes.

Sub CATMain()
    Dim documents1 As Documents
    Set documents1 = CATIA.Documents
    MsgBox "The number of documents is " & documents1.Count
    Dim doc1 As Document
    Set doc1 = documents1.Item(1)
    MsgBox doc1.Name
    MsgBox doc1.FullName
End Sub
Step 5

- Modify the previous code as shown in the code window to the right

- Run the new code using the “Run” button.

Note that the “For each” loop iterates through the Documents collection “documents1”. The same effect could be achieved with the below code, but a “For Each” loop is simpler:

```vba
Sub CATMain()
    Dim documents1 As Documents
    Set documents1 = CATIA.Documents
    MsgBox "The number of documents is " & documents1.Count
    Dim doc1 As Document
    'Set doc1 = documents1.Item(1)
    'MsgBox doc1.Name
    'MsgBox doc1.FullName
    For Each doc1 In documents1
        MsgBox doc1.Name
    Next
    Dim i As Integer
    For i = 1 To documents1.Count
        Set doc1 = documents1.Item(i)
        MsgBox doc1.Name
    Next
End Sub
```
Modify the previous code as shown in the code window to the right:

1) Uncomment a couple lines
2) Comment the “For Each” loop out completely
3) Add more code

Run the new code using the “Run” button.
Note that the PartDocument object “partDoc1” is “Set” to the Document object “doc1”. The setting of a variable to an object of a different type is possible in this case because the PartDocument class “inherits” from the Document class. This is depicted in the CATIA object diagram below. Not only does PartDocument inherit from Document, but so does ProductDocument, DrawingDocument, etc.
Modify the previous code as shown in the code window to the right by adding the line:

```
partDoc1.Close
```

Note in the Intellisense drop-down that the “Close” method has a green icon to indicate that it is a method as opposed to a property.

The icon for a property can be seen for the property “FullName”

Run the new code using the “Run” button

Note that the calling of this “Close” method performs the action of closing the document

```
Sub CATMain()
    Dim documents1 As Documents
    Set documents1 = CATIA.Documents
    MsgBox "The number of documents is " & documents1.Count
    Dim doc1 As Document
    Set doc1 = documents1.Item(1)
    MsgBox doc1.Name
    MsgBox doc1.FullName
    For Each doc1 in documents1
        MsgBox doc1.Name
    Next
    Dim partDoc1 As PartDocument
    Set partDoc1 = doc1
    MsgBox partDoc1.Name
    partDoc1.Close
End Sub
```
This concludes Workshop 4. In this workshop you have learned to:

- Access the properties of objects
- Iterate through the items of a collection
- Set a variable of one type to an object of another type that it inherits from
- Call the methods of objects
WORKSHOP 5

Navigating a Part Document with Error Handling
Description

In this workshop you will learn to “drill down” the object structure a part document while using error handling to deal with certain programmatic challenges.

Outline

1. Create a new CATPart in CATIA and create a new CATIA VBA library and module.
2. Use error handling in your code to check that the active document is a part document
3. Have the program count the number of sketch based features and datum planes in the part using error handling
• First, we are going to create some geometry to test our macro on. Units and names do not matter for this exercise.
• Create a new part
• In the part design workbench, draw a sketch of a square on any plane
• Pad the sketch (to any length)
• Under Geometrical Set 1, create an offset plane and a datum (explicit) plane
Step 2

- Create a new module in the myVBA macro library called NavigatePart
- Enter the code shown on the right
- With the new part open, run the code using either the play button or the Tools > Macro > Macros menu.
  
  NOTE: Nothing happens

- Now, create a new product document and leave it open in CATIA as the active document. Run the code again and the message box should appear

```vba
Sub CATMain()
    On Error Resume Next
    Dim partDoc1 As PartDocument
    Set partDoc1 = CATIA.ActiveDocument
    If Err.Number <> 0 Then
        MsgBox "Active document is not a part document!"
    End If
End Sub
```
Step 3

Open the V5Automation.chm file and go to the object diagram for part documents. The first goal is to add code that will count the number of sketch-based-features. These feature objects are all different kinds of "shapes". The diagram shows that we will have to “drill down” like so: Part > Bodies > Body > Shapes. Click on the red arrow next to the “Shape” box to see a more detailed view of Shape objects.
As you can see, there are many classes that inherit from the more general Shape class. Here you can see SketchBasedShape, BooleanShape, DressUpShape, and TransformationShape. We want to count the number of SketchBasedShape objects that are in the part.
Step 4: Add Code to Count SketchBasedShape Objects

- Add the code shown to the right to the previously entered code. Note that this first section of code “grabs” the Shapes collection underneath the first part body.

- This next section of code prepares the variables we will use to “grab” individual objects from the Shapes collection.

- The for each loop “grabs” each Shape in the Shapes collection, then attempts to set the more specific SketchBasedShape object to the more abstract Shape object. If the Shape object is in fact a SketchBasedShape, then no error is generated. If no error is generated, then the “sketchShapeCount” is increased by 1.

- Note that the Error object “Err” must be cleared each time so that errors don’t “carry over” to the next iteration of the loop. Run the code, you should see this message box:

```vba
Dim body1 As Body
Set body1 = partDoc1.Part.Bodies.Item(1)

Dim shapes1 As Shapes
Set shapes1 = body1.Shapes

Dim sketchShape As SketchBasedShape
Dim count1 As Integer
count1 = 0

For Each shape1 In shapes1
    Set sketchShape = shape1
    If Err.Number = 0 Then
        count1 = count1 + 1
    End If
    Err.Clear
Next

MsgBox "Number of sketch based shapes is " & count1
End Sub
```
Step 5: Add Code to Count BooleanShape Objects

Another method for counting the number of sketch based features is the For Next loop.

Note the Item(1) in the code. This means we will only look at the first geometrical set within a part file. If you want to expand the code you could add another loop to go through all geometrical sets within a catpart.

```vba
Dim body1 As Body
Set body1 = partDoc1.Part.Bodies.Item(1)

Dim shapes1 As Shapes
Set shapes1 = body1.Shapes

shapecount = shapes1.count

Dim shape1 As Shape

Dim sketchShape As SketchBasedShape

Dim count1 As Integer
count1 = 0

For i = 1 To shapecount

    Set sketchShape = shapes1.Item(i)
    If Err.Number = 0 Then
        count1 = count1 + 1
    End If

    Err.Clear

Next
```
Step 6: Use the Part Document Object Diagram

Return to the part document object diagram of the .chm documentation (picture on the following page). Note that the Part object has a collection called HybridBodies. This is a collection of all the geometrical sets of the part, but the programmatic name of a geometrical set is HybridBody. We want to add code to count the number of datum planes. A datum plane is a hybrid shape, but we don’t know for sure what it’s programmatic name is. Click on the red arrow indicated below to go to the object diagram for the HybridShape object.
Step 7: Use the HybridShape Object Diagram

There are many classes that inherit from the abstract class HybridShape, as can be seen below. We are looking for a class that is the equivalent of a datum plane. The class HybridShapePlaneExplicit looks like the right one.
Step 8: Add Code to Count Datum Planes

Insert the following code below what we already have. You should get the resulting message box.

Set part1 = partDoc1.Part
Dim hybridShapeFactory1 As HybridShapeFactory
Set hybridShapeFactory1 = part1.HybridShapeFactory
Dim hybridBodies1 As HybridBodies
Set hybridBodies1 = part1.HybridBodies
Dim hybridBody1 As HybridBody
Set hybridBody1 = hybridBodies1.Item(1)
geocount = hybridBody1.HybridShapes.count
Dim k, count2 As Integer
Dim datumPlane As HybridShapePlaneExplicit
count2 = 0
For k = 1 To geocount
    Set datumPlane = hybridBody1.HybridShapes.Item(k)
    If Err.Number = 0 Then
        count2 = count2 + 1
    End If
    Err.Clear
Next
MsgBox "Number of sketch based shapes is: " & count1 & vbNewLine & _
    "Number of datum planes is : " & count2
End Sub
This concludes Workshop 5. In this workshop you have learned to:

• Use the object diagrams found in the “.chm” file to understand CATIA objects

• Navigate the structure of a CATIA part document by iterating through the various collections that a part contains

• Access the Shapes collection under a part Body object in order to iterate through features of the part

• Access the HybridShapes collection under a HybridBody (Geometrical Set) to iterate through various 3D geometry
WORKSHOP 6
Creating Sketch Geometry
Description

In this workshop you will explore the creation of sketch geometry in a part document. The purpose of this is to illustrate the use of “factories” to create objects in a part document.

Outline

1. Start a new macro recording.

2. Create a sketch on the Y-Z plane that consists of a single line.

3. Open the recorded macro and observe what has been recorded.

4. Modify the macro to create additional geometry and rerun it on a new part document.
Step 1: Start Recording a Macro

- Create a new part document
- Start recording a new macro.
- Name it “Create_Sketch_01”
Step 2. Start the sketch

• Start a sketch with the “Sketch” button

• Select the yz plane as the sketch Plane

• Create a line away from the origin that isn’t parallel to the “V” axis or “H” axis

• Complete the sketch with the “Exit Workbench” button

• The resulting should similar to what is pictured to the right. It is important that there are no constraints in the sketch that was created.
Step 3: Start Commenting the Code

Open the CATIA VBA editor to view the code that was recorded (see left). A good first step after recording a macro is to add comments to clearly call-out the sequence of “stages” of the code. To start, add the comments seen below to call-out the first step, which is the getting of the part body.

```
Sub CATMain()
'===========================================
' Get the part body
'===========================================
Dim partDocument1 As PartDocument
Set partDocument1 = CATIA.ActiveDocument
Dim part1 As Part
Set part1 = partDocument1.Part
Dim bodies1 As Bodies
Set bodies1 = part1.Bodies
Dim body1 As Body
Set body1 = bodies1.Item("PartBody")
Dim sketches1 As Sketches
Set sketches1 = body1.Sketches
Dim originElements1 As OriginElements
Set originElements1 = part1.OriginElements
Dim reference1 As Reference
Set reference1 = originElements1.PlaneYZ
Dim sketch1 As Sketch
Set sketch1 = sketches1.Add(reference1)
Dim arrayOfVariantOfDouble1(8)
arrayOfVariantOfDouble1(0) = 0#
arrayOfVariantOfDouble1(1) = 0#
arrayOfVariantOfDouble1(2) = 0#
arrayOfVariantOfDouble1(3) = 0#
arrayOfVariantOfDouble1(4) = 1#
arrayOfVariantOfDouble1(5) = 0#
arrayOfVariantOfDouble1(6) = 0#
arrayOfVariantOfDouble1(7) = 0#
arrayOfVariantOfDouble1(8) = 1#
Set sketch1Variant = sketch1
sketch1Variant.SetAbsoluteAxisData arrayOfVariantOfDouble1
```
Step 4: Comment and test the code

Sub CATMain()
    '===============================================
    ' Get the part body
    '===============================================
    Dim partDocument1 As PartDocument
    Set partDocument1 = CATIA.ActiveDocument

    Dim part1 As Part
    Set part1 = partDocument1.Part

    Dim bodies1 As Bodies
    Set bodies1 = part1.Bodies

    Dim body1 As Body
    Set body1 = bodies1.Item("PartBody")

    '===============================================
    ' Add a sketch on the Y-Z Plane
    '===============================================
    Dim sketches1 As Sketches
    Set sketches1 = body1.Sketches

    Dim originElements1 As OriginElements
    Set originElements1 = part1.OriginElements

    Dim reference1 As Reference
    Set reference1 = originElements1.PlaneYZ

    Dim sketch1 As Sketch
    Set sketch1 = sketches1.Add(reference1)

    '===============================================
    End

• Add these comments in order to call-out the next “stage” in the code: the addition of a sketch to the Y-Z plane

• Add this “End” command here. This will cause the macro to stop here when it is run.

• Open a new part document and run the macro. Note that the result is that a new sketch is created.
Step 5: Comment and test the code

- Remove the “End” command and add a comment to call-out the next stage:

- This next stage of code can be thought of as containing two parts:
  1. The creation of an 9 member array of variants
  2. The calling of the “SetAbsoluteAxisData” method.

The array contains the numerical information that describes a 2D axis system that is positioned in 3D space.

Note that a variable that isn’t dimmed (sketch1Variant) is set to the sketch object, then “SetAbsoluteAxisData” is called on it. This is a requirement: calling “SetAbsoluteAxisData” on the actual sketch object like so would cause a compile error:

```plaintext
sketch1.SetAbsoluteAxisData arrayOfVariantOfDouble1
```

- Add the “End” command as shown.
- Open a new part document and run the macro. Note that the result is indistinguishable from the last time the macro was run. This is because the position of the axis system that is described by the array is the same as the default position.
Step 6: Replace code with simpler code

- Remove the “End” command and add a comment to call-out the next stage: creating sketch geometry.

- The first command sets the “In Work Object”. Leave this command as-is.

- Note that the GeometricElements object is only used to get this Axis2D object. Simplify the code by replacing this code with a more direct approach. You can think of this as “collapsing” two sections of code into one (two lines replace the original four).

**REPLACEMENT CODE**

```vba
dim axis2d1 as axis2d
set axis2d1 = sketch1.geometricelements.item("absoluteaxis")
```

---

```vba
dim axis2d1 as axis2d
set axis2d1 = sketch1.geometricelements.item("absoluteaxis")
dim point2d1 as point2d
set point2d1 = factory2d1.createpoint(20#, -20#)
point2d1.reportname = 3
dim point2d2 as point2d
set point2d2 = factory2d1.createpoint(60#, -50#)
point2d2.reportname = 4
dim line2d1 as line2d
set line2d1 = axis2d1.getitem("hdirection")
line2d1.reportname = 1
dim line2d2 as line2d
set line2d2 = axis2d1.getitem("vdirection")
line2d2.reportname = 2
```

---

WS-75
Step 7: Identify code for removal (1/3)

- The commands such as “line2D1.ReportName = 1” and “look peculiar. They seem to serve no purpose. See the next slide for tips on how to investigate the purpose of this command.

- Note that although the recorded macro shows that a line2D1 and line2D2 were created, these objects’ only use is to have their “ReportName” properties set. If we determine that the setting of these values isn’t necessary, we can get rid of this section of code all together.

```vba
Dim line2D1 As Line2D
Set line2D1 = axis2D1.GetItem("HDirection")
line2D1.ReportName = 1

Dim line2D2 As Line2D
Set line2D2 = axis2D1.GetItem("VDirection")
line2D2.ReportName = 2

Dim point2D1 As Point2D
Set point2D1 = factory2D1.CreatePoint(20#, -20#)
point2D1.ReportName = 3

Dim point2D2 As Point2D
Set point2D2 = factory2D1.CreatePoint(60#, -50#)
point2D2.ReportName = 4

Dim line2D3 As Line2D
Set line2D3 = factory2D1.CreateLine(20#, -20#, 60#, -50#)
line2D3.ReportName = 5
line2D3.StartPoint = point2D1
line2D3.EndPoint = point2D2

sketch1.CloseEdition
part1.InWorkObject = body1
part1.Update
End Sub
```
Step 7: Identify code for removal (2/3)

- The programming object in question is the class “Line2D”. Hit F2 to enter the object browser and search “CATIALine2D”

- All the “Members” of CATIALine2D (Line2D) are listed. Right-click on “ReportName” and select “Help”
Step 7: Identify code for removal (3/3)

- On the documentation page that comes up, note that “Geometry2D” is the class. This is because although “ReportName” is a property of “Line2D”, this property is inherited from “Geometry2D”

- The description of the ReportName property doesn’t tell us much. For one, it only states that this property returns the report name, but we can see in the recorded macro that it “sets” the report name.

- For now, although the macro does use this property to “set” these report name values, assume that this is an unnecessary step, and remove these lines of code.
Step 8: Renaming and more code removal

// Creating Sketch Geometry

part1.InWorkObject = sketch1

Dim factory2D1 As Factory2D
Set factory2D1 = sketch1.OpenEdition()

Dim axis2D1 As Axis2D
Set axis2D1 = sketch1.GeometricElements.Item("AbsoluteAxis")

Dim point2D1 As Point2D
Set point2D1 = factory2D1.CreatePoint(20#, -20#)

Dim point2D2 As Point2D
Set point2D2 = factory2D1.CreatePoint(60#, -50#)

Dim line2D3 As Line2D
Set line2D3 = factory2D1.CreateLine(20#, -20#, 60#, -50#)

line2D3.StartPoint = point2D1
line2D3.EndPoint = point2D2

sketch1.CloseEdition
part1.InWorkObject = body1
part1.Update
End Sub

- This stage of code, after modifications, is shown to the left
- Note that first two points are created, and it appears as though the line gets created by using the same coordinates of the points.
- The “StartPoint” property and “EndPoint” property need to be explicitly set to the two points that were created.
- In order to increase the clarity of the code, many objects could be renamed in a more descriptive manner. The Point2D object “point2D1” could be named “pt2Dstart”, and “point2D2” could be named “pt2Dend”.
- These couple lines of code now serve no purpose, and may be removed.
This concludes Workshop 6. In this workshop you have learned to:

- Start a CATIA VBA program by first recording a macro
- Record the creation of a single line in a sketch
- How to edit a recorded macro by using comments, by renaming variables, and by deleting unnecessary lines, for the purpose of increasing clarity and simplicity in the code
- How to use the object browser and help documentation to investigate unknown API properties and methods
WORKSHOP 7

Using Forms in CATIA VBA

 VB Scripting for CATIA V5 – Workshop 7
 Nick Weisenberger, February 2012
**Description**

In this workshop you will learn how to use VBA forms in order to provide users with a user interface from which they can launch CATIA VBA programs.

**Outline**

1. Create a form in the macro library

2. Create various buttons and fields in the form.

3. Re-use code from an earlier workshop in order to add CATIA functionality and interactivity
Step 1a: Create a new form in the macro library

- Open the CATIA VBA editor so that the myVBA macro library is visible. Make sure the “Create_Sketch_01” module is present.

- Right-click on the macro library and select “Insert > UserForm”
Step 1b

- Name it the form “frmCreateLine” by editing the “(Name)” property.

- Edit the caption of the form by using the “Caption” property.
Step 2: View the Code of the Form

- With “frmCreateLine” selected in the project tree, click the “View Code” button.

- Note that a code window appears. There are two drop-down menus, one that shows which object’s code you are looking at, the other showing the “event” subroutine you are editing. By default the “Click” event is being edited, and the subroutine for this object and event is “UserForm_Click”.

- As a quick experiment, enter the command featured to the right in this “Click” sub.
Step 3a: View the Code of the Form

- You need to create a module that displays the form you just created. Create the new module and name it “Show_CreateLine_Form”

```
Sub CATMain()
    frmCreateLine.Show
End Sub
```

- In the blank code window for the newly created module enter the following code:
Step 3b: View the Code of the Form

Run this new macro using the “Run” button. A blank form should appear over the CATIA window. Click on the form with your mouse and a message box should appear displaying “hello”. As you can see, the form recognizes the click “event” and it runs the code each time you click it. Close the form by using the “x” button in the top right corner.
Step 4: Create a Text Box

- Return to the form in the VBA editor, left-click and hold on the “TextBox” button of the form toolbar, and drag and drop a TextBox onto the form.

- Click the “View Code” button to view the code of the form. Note that in the drop-down menu on the left, there are now two “objects” to select from “UserForm” and “TextBox1”. Each of these objects as it’s own set of “events” that can be chosen from the drop-down menu on the right.
Step 5: Create a Text Box (continued)

• Edit the “(Name)” property of the TextBox to “tbPointA”

• Edit the “Text” property of the TextBox to be “0” (zero). This will set the TextBox so that it’s default text that it contains is “0”.

• Edit the code of the Click event of the form like so.

• Note that in the edited code above, tbPointA is an “object” whose “Text” property is being accessed. So the message box will display the contents of the TextBox when the form is clicked.
Step 5: Create a Text Box (continued)

- Run the “Show_CreateLine_Form” Macro

- Click on the form to trigger its Click event. Note that the message box displays the content of the text box.

- Edit the content of the TextBox and click on the form again. Note that the value that gets displayed again matches the content of the TextBox.
Step 6: Add More Form Objects

- Drag and drop the objects pictured below onto the form. Edit their properties as indicated.
Step 7: Edit the “Click” Event of the Command Button

- Double-click the “Create Line” command button on the form.

- The code window will come up with various subroutines visible. The cursor should be blinking within the cbCreateLine_Click() subroutine.

- Also, note that because the cursor is positioned in this subroutine, the object and event that are shown in the drop-down menus match this.
Rather than redevelop code that creates a line in a sketch, we’ll just cut and paste the code previously developed into the Click event of this form. Double-click the “Create_Sketch_01” module to bring its code window up. Select all the code in the code window (other than the “Sub CATMain()” at the beginning and “End Sub” at the end” and hit Ctrl + C.

![Code window of Create_Sketch_01 module]
9b. Insert the Code from the “Create_Sketch_01”

- Double-click on “frmCreateLine” on the project tree to bring up the form.

- Double click on the “Create Line” button to bring up the form’s code window.

- Make sure the cursor resides in the “cbCreateLine_Click()” subroutine and hit Ctrl + v to paste the copied code in.
The goal is to edit the code that has been pasted so that when the “Create Line” button is clicked, a line gets created whose start point has the coordinates of “Point A” and whose end point has the coordinates of “Point B”.

The arrows below show the correspondence between the fields of the form and the parameters of the methods that are called in the code.
The following replacements need to be made. Note that previously, numerical values were “hard coded” in as parameters to the “CreatePoint” methods. Now we are replacing them with the properties of form objects, in this case the “Text” property of the various TextBox objects that are on the form.

```
='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='=
' Creating Sketch Geometry
'='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='='=
part1.InWorkObject = sketch1

Dim factory2D1 As Factory2D
Set factory2D1 = sketch1.OpenEdition()

Dim axis2D1 As Axis2D
Set axis2D1 =
    sketch1.GeometricElements.Item("AbsoluteAxis")

Dim point2D1 As Point2D
Set point2D1 = factory2D1.CreatePoint(20#, -20#)

Dim point2D2 As Point2D
Set point2D2 = factory2D1.CreatePoint(60#, -50#)

Dim line2D3 As Line2D
Set line2D3 = factory2D1.CreateLine(20#, -20#, 60#, -50#)
```

**REPLACEMENT CODE**

```
Dim point2D1 As Point2D
Set point2D1 = factory2D1.CreatePoint(tbPointAx.Text, tbPointAy.Text)

Dim point2D2 As Point2D
Set point2D2 = factory2D1.CreatePoint(tbPointBx.Text, tbPointBy.Text)

Dim line2D3 As Line2D
Set line2D3 = factory2D1.CreateLine(tbPointAx.Text, tbPointAy.Text, tbPointBx.Text, tbPointBy.Text)
```
Step 11. Save the VBA project and run the macro

- Highlight the VBA library on the project tree and save your changes.

- Close the VBA window, open a new part, and open the macros dialog through Tools > Macro > Macros, and run the “Show_CreateLine_Form” macro.
• Click the “Create Line” button and you will see that a sketch appears containing a line whose coordinates match those that are in the form TextBox objects. However, the form remains open, and if you attempt to interact with CATIA you are unable because the form has been shown in what is called “modal” mode. It would be nice to change this so that after a sketch is created, you could zoom in or rotate, expand the spec tree, etc.

• Close the form using the “x” button in the top-right corner.
Step 12b. Create a sketch using the form

- Open the VBA editor and navigate to the “Show_CreateLine_Form” module and edit it as show below. Note that an argument “vbModeless” will be passed to the “Show” subroutine that is called on the form.

```vba
Sub CATMain()
    frmCreateLine.Show vbModeless
End Sub
```

- Save the project, close the VBA editor, and run the “Show_CreateLine_Form” macro. Note that even before you hit the “Create Line” button you are able to interact with CATIA (zoom in, rotate, etc.). Change point coordinate values on the form and hit the “Create Line” button to create another sketch.
This concludes Workshop 7. In this workshop you have learned to:

• Create a form and insert code into it’s “Event” subroutines

• Add form objects to the form such as TextBox objects, Label objects, and CommandButton objects, edit their properties, and insert code into their events as well

• Reference the properties of form objects for use in code. For instance, you used the “Text” properties of the TextBox objects that were created on the form.

• “Show” a form in both “Modal” and “Modeless” scenarios