Unleashing Hidden Powers of Inventor with the API

Part 1. Getting Started with Inventor VBA – “Hello Inventor!”

Brian Ekins – Autodesk, Inc.

This article provides an introduction to Inventor's VBA programming environment and some of the basic concepts you need to understand when programming Inventor.

Key Topics:

❑ The basics of Visual Basic for Applications (VBA)

❑ Understand how Inventor provides a programming interface

Target Audience:

Autodesk Inventor users who want to increase their productivity with Inventor by writing programs, but have never used VBA included with Inventor.

About the Author:
Brian is an Autodesk Inventor API evangelist and works with professionals and companies to help make using Inventor's programming interface easier. Brian began working in the CAD industry over 25 years ago. He has worked in CAD administration, as an application engineer, CAD API designer, and consultant. Brian was the original designer of Inventor's API and has presented at conferences and taught classes throughout the world to thousands of users and programmers.

brian.ekins at autodesk.com

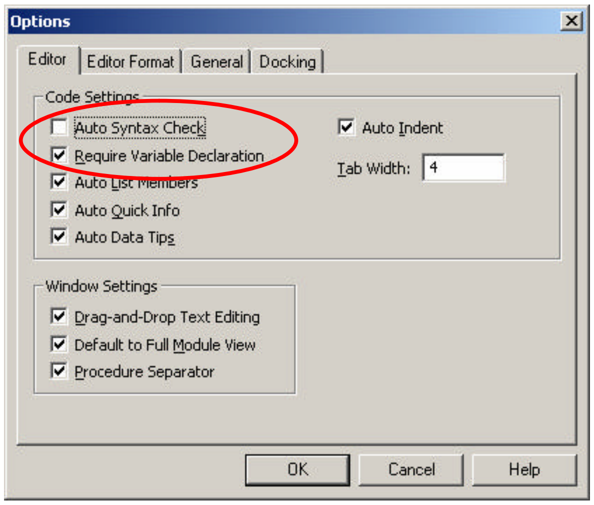
Introduction

This article series is not intended to be a comprehensive coverage of VBA programming, but looks briefly at VBA and the steps required to begin writing a program.

VBA Basics

Visual Basic for Applications (VBA) is a programming environment developed by Microsoft and licensed by other companies to integrate into their applications. Autodesk licenses VBA and includes it in AutoCAD and Inventor.. This provides all customers of Inventor with a free development environment. Below are the minimal steps to begin using VBA and writing a simple macro.

You access VBA through Inventor using the **Macro | Visual Basic Editor** command in the Tools menu, or by pressing Alt-F11. Once the VBA environment is open, the first thing I recommend you do is change some of the VBA settings. In the VBA environment run the **Options** command from the Tools menu. Change the **Auto Syntax Check** and **Require Variable Declaration** settings to those shown below.

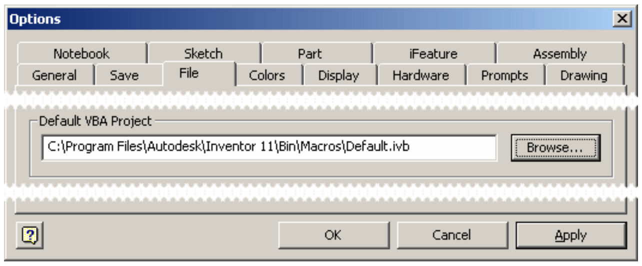


Code Settings
- [ ] Auto Syntax Check
- [✓] Require Variable Declaration
- [✓] Auto List Members
- [✓] Auto Quick Info
- [✓] Auto Data Tips
- [✓] Auto Indent

Tab Width: 4

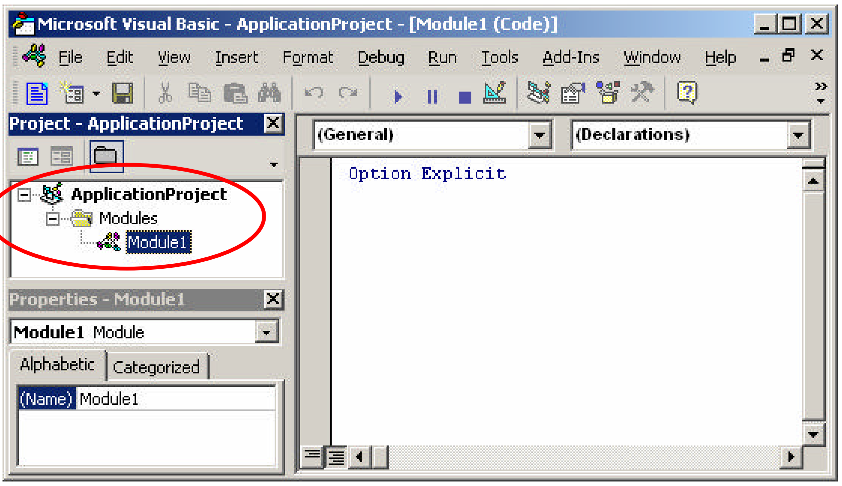
Window Settings
- [✓] Drag-and-Drop Text Editing
- [✓] Default to Full Module View
- [✓] Procedure Separator

The VBA programs you write can be saved in Inventor documents or in external files. For 99% of the cases, saving it in an external file is best, so that's what we'll cover here. This external file is referred to as a VBA project and Inventor saves it as an .ivb file. A single project can contain any number of macros (programs). There is a default VBA project that Inventor loads at startup. The default VBA project is defined in the File tab of the Inventor application options, as shown below. This is where your programs will be saved.



Default VBA Project: C:\Program Files\Autodesk\Inventor 11\Bin\Macros\Default.ivb

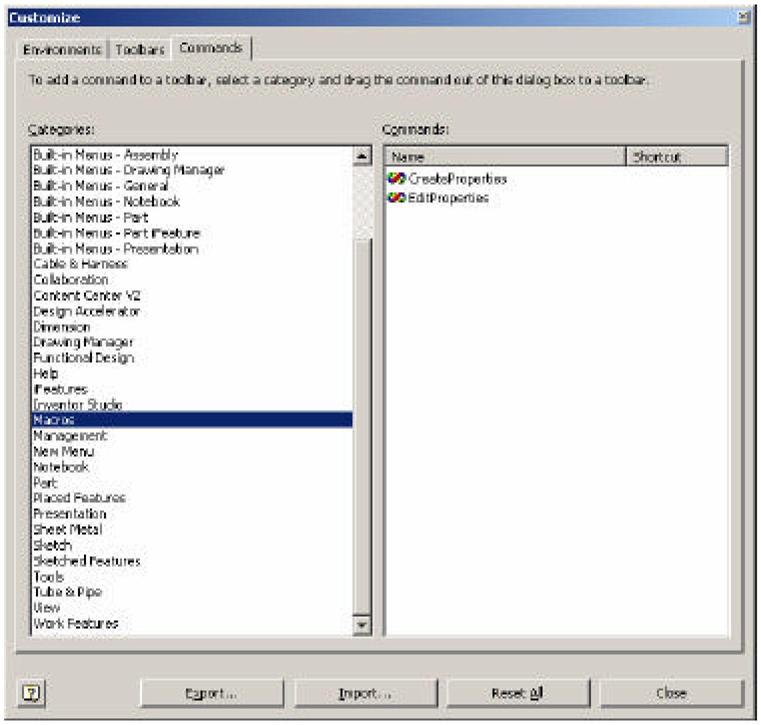
A project can consist of many different types of modules. Macros are written in a code module within the VBA project. There is a code module automatically created for any new VBA project named “Module1”. To add code to this module you can double-click on the module in the Project Explorer window, as shown below. This will cause the code window for that module to be displayed.


Microsoft Visual Basic - ApplicationProject - [Module1 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Project - ApplicationProject
ApplicationProject
Modules
Module1
Properties - Module1
Module1 Module
Alphabetic Categorized
(General)
(Declarations)

(Name) Module1

```
Option Explicit
```

There are several ways to run a VBA macro. From the VBA editor you can position the cursor in the code window within the macro you want to run and execute the **Run Macro** command on the main toolbar. From within Inventor you can run the **Macro | Macros…** command from the Tools menu (or Alt+F8) which brings up a dialog where you can choose a macro and run it. Finally, you can use the Inventor **Customize** command to create a button that will run a macro. For frequently used macros this provides the most convenient interface for the end-user. This is done using the **Commands** tab of the Customize dialog, as shown below. Select "Macros" as the category. The available macros are listed on the right-hand side of the dialog. To create a button to execute the macro, just drag and drop the desired macro onto any existing toolbar.


Customize
Environments Toolbars Commands
Export... Import... Reset All Close

To add a command to a toolbar, select a category and drag the command out of this dialog box to a toolbar.

Categories:
Built-in Menus - Assembly
Built-in Menus - Drawing Manager
Built-in Menus - General
Built-in Menus - Notebook
Built-in Menus - Part
Built-in Menus - Part Feature
Built-in Menus - Presentation
Cable & Harness
Collaboration
Content Center V2
Design Accelerator
Dimension
Drawing Manager
Functional Design
Help
Features
Inventor Studio
Macros
Management
New Menu
Notebook
Part
Placed Features
Presentation
Sheet Metal
Sketch
Sketched Features
Tools
Tube & Pipe
View
Work Features

Commands:

Name	Shortcut
CreateProperties	
EditProperties	

The Basics of Inventor's Programming Interface

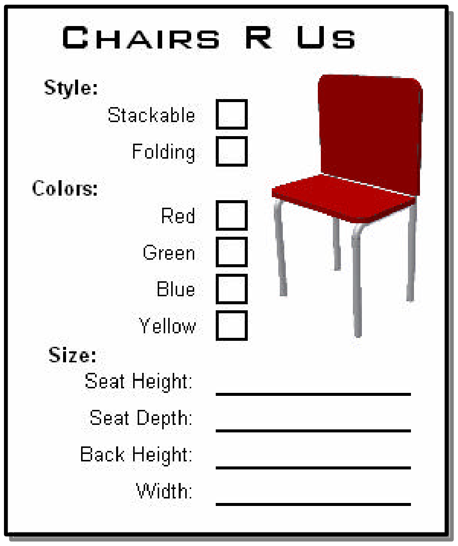
Inventor supports the ability for you to automate tasks by writing programs. Inventor does this using some Microsoft technology called COM Automation. This is the same technology used when you write macros for Word or Excel. The technology itself isn't important, but what is important is what it allows you to do.

COM Automation allows applications, like Inventor, to provide a programming interface in a fairly standard structure that can be used by most of the popular programming languages available today. This provides two benefits to you. First, if you've programmed other applications that use COM Automation (Word, Excel, AutoCAD) Inventor will have a lot of similarities. Second, it's very flexible in how you access the programming interface so you can choose your favorite programming language and integrate with other applications.

The topics below discuss the basic concepts you'll need to understand when working with Inventor. You can apply these concepts to any COM Automation programming interface.

Objects

A COM Automation interface exposes it functions through a set of *objects*. A programming object has many similarities to real-world objects. Let's look at a simple example to understand how object oriented programming terminology can be used to describe a physical object.



A company that sells chairs might allow a customer to design their own chair by filling out an order form, like the one shown to the right. The options on the order form define the various *properties* of the desired chair. By setting the properties to the desired values the customer is able to describe the specific chair they want.

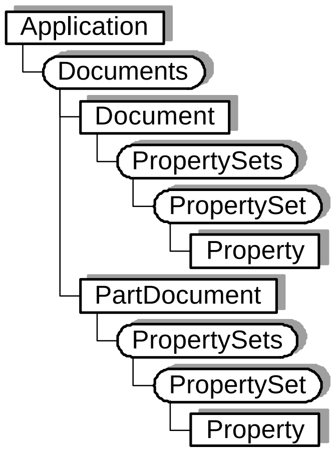
In addition to properties, objects also support *methods*. Methods are basically instructions that the object understands. In the real world, these are actions you would perform with the chair. For example, you could move the chair, cause it to fold up, or throw it in the trash. In the programming world the objects are smart and instead of you performing the action you tell the object to perform the action itself; move, fold, and delete.

In Inventor, some examples of objects are extrude features, work planes, constraints, windows, and iProperties. Inventor's programming interface defines the set of available objects and their associated methods and properties. By obtaining the object you're interested in you can find out information about it by looking at the values of its properties and edit the object by changing these values or by calling the objects methods. The first step in manipulating any object is getting access to the object you want. This is explained in the next section.

The Object Model

As discussed above, Inventor's API is exposed as a set of objects. By gaining access to these objects through the API you can use their various methods and properties to create, query, and modify them. Let's look at how the *object model* allows you to access these objects. Understanding the concept of the object model is a critical concept to using Inventor programming interface.

The object model is a hierarchical diagram that illustrates the relationships between objects. A small portion of Inventor's object model is shown in the figure to the right. Only the objects relating to iProperties are shown. In most cases you can view these as parent-child relationships. For example, the Application is the parent of everything. The Document object is a parent for the various property related objects. To obtain a specific object within the object model you typically start at the top and then go down child-by-child to the object you want. For example, you would start with the Application object and from it get the Document that contains the iProperty you're interested in. From the Document you then get the PropertySet that is the parent of the iProperty and finally you get the desired Property object.



The Application Object

One of the most important objects in the object model is the *Application* object. This object represents the Inventor application and is the top-most object in the hierarchy. The Application object supports methods and properties that affect all of Inventor but its most important trait is that through it you can access any other Inventor object. You just need to know how to traverse the object model to get to the specific object you want. We'll look at how to write a program to do this and some shortcuts you can take to make it a little bit easier.

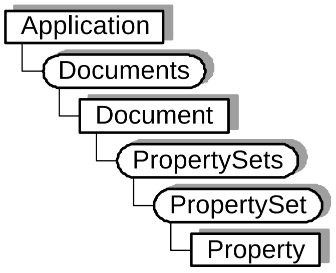
When using Inventor's VBA there is the global variable called ThisApplication that always contains the Application object. So with Inventor's VBA

Collection Objects

Another concept that's important to understand when working with Inventor's programming interface is *collection* objects. Collection objects are represented in the object model diagram by boxes with rounded corners. In the object model picture to the right the Documents, PropertySets, and PropertySet objects are collection objects.

The primary job of a collection object is to provide access to a group of related objects (the set of children for that parent). For example, the Documents object provides access to all of the documents that are currently open in Inventor. A collection provides access to its contents through two properties; *Count* and *Item*. The Count property tells you how many items are in the collection and the Item property returns a specific item. All collections support these two properties.

Typically, when using the Item property you specify the index of the item you want from the collection (i.e. Item 1, 2, 3 ...). For example, the code below prints out the filenames of all of the documents currently open by using the Count and Item properties of the Documents collection object. (This and most of the following samples use Inventor's VBA and take advantage of the ThisApplication global variable.)



```
Public Sub ShowDocuments()
   ' Get the Documents collection object.
   Dim invDocs As Documents
   Set invDocs = ThisApplication.Documents

   ' Iterate through the contents of the Documents
collection.
   Dim i As Integer
   For i = 1 To invDocs.Count
      ' Get a specific item from the Documents collection.
      Dim invDocument As Document
      Set invDocument = invDocs.Item(i)

      ' Display the full filename of the document in the Immediate window.
      Debug.Print invDocument.FullFileName
   Next
End Sub
```

Another technique of going through the contents of a collection is to use the For Each statement. This can also be more efficient and results in a faster program in many cases. The macro below accomplishes exactly the same task as the previous macro but uses the For Each statement.

```
Public Sub ShowDocuments2()
   ' Get the Documents collection object.
   Dim invDocs As Documents
   Set invDocs = ThisApplication.Documents

   ' Iterate through the contents of the Documents collection.
    Dim invDocument As Document
    For Each invDocument In invDocs
      ' Display the full filename of the document in the Immediate window.
      Debug.Print invDocument.FullFileName
   Next
End Sub
```

When the Item property is used with a value indicating the index of the item, the first item in the collection is 1 and the last item is the value returned by the collection's Count property. For some collections the Item property also supports specifying the name of the item you want. Instead of specifying the index of the item you can supply a String that specifies the name of the object. We'll see this later when working with iProperties.

As you try out more things, you will see other important feature of many collections such as their ability to create new objects. For example the Documents collection supports the Add method which is used to create new documents.

VB.Net (Visual Basic 2005 Express Edition)

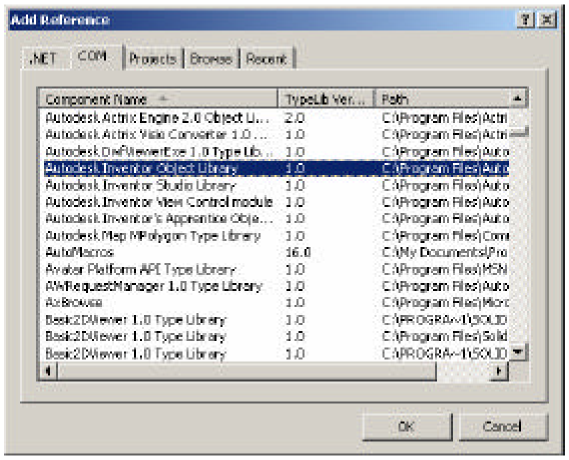
In this detour from the main topic, let's look at how to connect to Inventor and get the application object using VB.Net. To add a reference to the Inventor type library use the **Add Reference** command found under the Project menu. In the dialog, shown to the right, select the "COM" tab, select the "Autodesk Inventor Object Library" item and click the "OK" button.

The VB.Net code is shown below. It's identical to the VBA/VB code above except it uses a different style of error handling that only VB.Net supports.

```
Public Sub InventorConnectSample()
    ' Declare a variable as Inventor's Application type.
    Dim invApp As Inventor.Application

    ' Enable error handling.
    Try
        ' Try to connect to a running instance of Inventor.
        invApp = GetObject(, "Inventor.Application")
    Catch ex As Exception
        ' Connecting to a running instance failed so try to start Inventor.
        Try
            ' Try starting Inventor.
            invApp = CreateObject("Inventor.Application")

            ' When Inventor is started using CreateObject it is started
            ' invisibly.  This will make it visible.
            invApp.Visible = True
        Catch ex2 As Exception
            ' Unable to start Inventor.
            MsgBox("Unable to connect to or start Inventor.")
            Exit Sub
        End Try
    End Try
End Sub
```



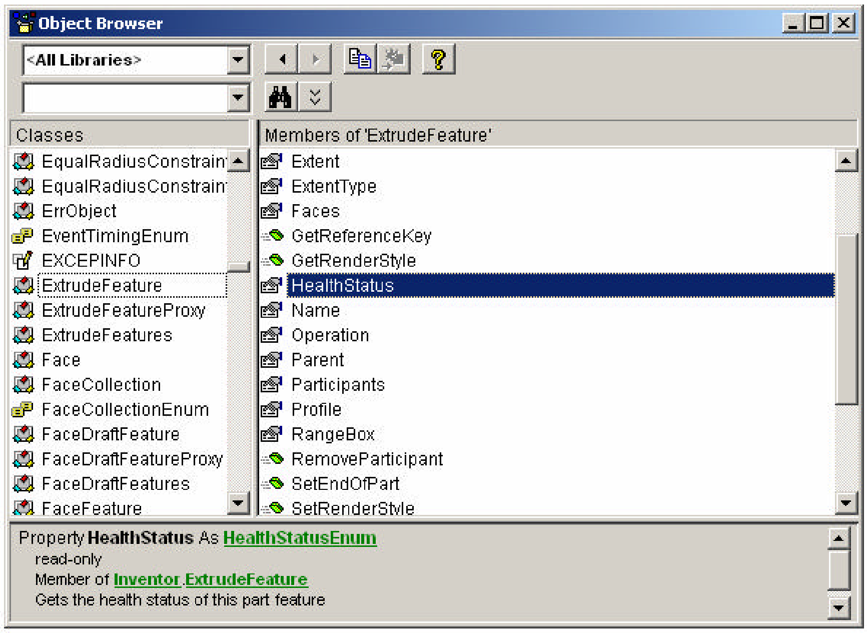
Component Name	TypeLib Ver...	Path
Autodesk.Actrix Engine 2.0 Object Li...	2.0	C:\Program Files\Actri
Autodesk.Actrix Visio Converter 1.0 ...	1.0	C:\Program Files\Actri
Autodesk.DwfViewerExe 1.0 Type Lib...	1.0	C:\Program Files\Auto
Autodesk.Inventor Object Library	1.0	C:\Program Files\Auto
Autodesk.Inventor Studio Library	1.0	C:\Program Files\Auto
Autodesk.Inventor View Control module	1.0	C:\Program Files\Auto
Autodesk.Inventor's Apprentice Obje...	1.0	C:\Program Files\Auto
Autodesk.Map MPolygon Type Library	1.0	C:\Program Files\Com
AutoMacros	16.0	C:\My Documents\Pro
Avatar Platform API Type Library	1.0	C:\Program Files\MSN
AWRequestManager 1.0 Type Library	1.0	C:\Program Files\Auto
AxBrowse	1.0	C:\Program Files\Micr
Basic2DViewer 1.0 Type Library	1.0	C:\PROGRA~1\SOLID
Basic2DViewer 1.0 Type Library	1.0	C:\Program Files\Solid
Basic2DViewer 1.0 Type Library	1.0	C:\PROGRA~1\SOLID

Programming Tools

There are tools available to help you when working with Inventor's API. The following describes each of these.

Object Browser

This is the tool that I personally use the most when working with Inventor's API. You access it from the VBA programming environment by pressing the F2 key, the toolbar button, or selecting **Object Browser** from the **View** menu. The Object Browser is shown below.

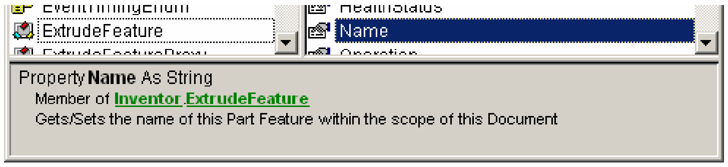


Property **HealthStatus** As HealthStatusEnum
read-only
Member of Inventor.ExtrudeFeature
Gets the health status of this part feature

The left column of the browser, titled "Classes", contains a list of all the objects available in the libraries currently attached. In the drop-down list at the top-left of the browser, "<All Libraries>" in the example above, you can specify a specific library to limit the list to only the objects in that library. In this example, the ExtrudeFeature class has been selected.

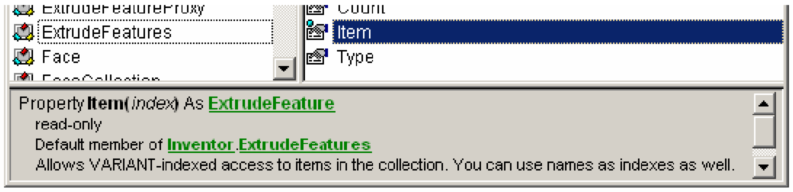
The right column displays a list of the methods, properties, and events supported by the selected class. Selecting one of these will display information about that function at the bottom of the object browser. In the example above the HealthStatus property has been selected. At the bottom of the dialog we learn that this property returns a HealthStatusEnum value. (Enum values are lists of values that are also defined in Inventor's type library.) We also learn that this property is read-only, meaning that we can get the value but we can't change it. A brief description of the selected function is also displayed at the bottom of the browser. The online help is useful in conjunction with the browser because it is context sensitive. Pressing F1 at this point will bring up the help page for the HealthStatus property.

Let's look at a few more examples of the information that's displayed at the bottom of the object browser and how to interpret this information. In the example below, the Name property of the ExtrudeFeature object is selected. The property returns a String. Notice that unlike the previous example, it does not say that it is read-only so this property is read-write. Using this property you can get the name of the feature and you can also assign a String to this property to set the name of the feature.

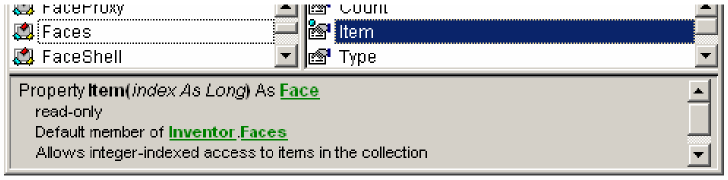


Property **Name** As String
Member of Inventor.ExtrudeFeature
Gets/Sets the name of this Part Feature within the scope of this Document

The Item property of a collection object was discussed earlier. Remember that you can always use a value as the index to get a specific item and sometimes you can also use the name of the item. The two examples below show one technique to determine if using a name is supported. The first example is for the ExtrudeFeatures collection. It shows that the Item property has one argument called index. Notice that it doesn't say what type this argument is. Whenever the type is not specified that means it is a *Variant*. A Variant is a special type that can represent any variable type. In this case index is a Variant to allow you to provide either a Long or a String. Often, as in this case, this is also indicated in the description and in the online help. In the second example, index is specified to be a Long value. In this case you can only supply a value. Supplying a String will cause a failure.

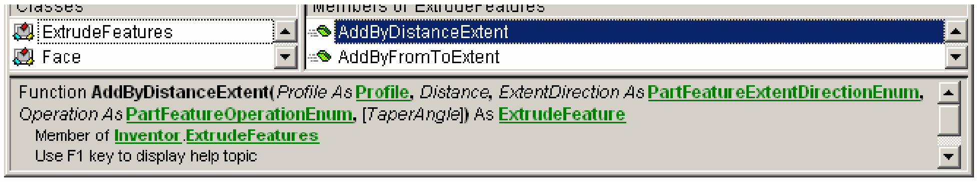


Property **Item**(*index*) As ExtrudeFeature
read-only
Default member of Inventor.ExtrudeFeatures
Allows VARIANT-indexed access to items in the collection. You can use names as indexes as well.



Property **Item**(*index As Long*) As Face
read-only
Default member of Inventor.Faces
Allows integer-indexed access to items in the collection

Let's look at a more complicated signature and what we can learn by looking at the signature. We'll use the AddByDistanceExtent method of the ExtrudeFeatures collection object. This method has several arguments. The first is called Profile and requires a Profile object as input. The second called Distance does not have a type displayed because it is a Variant. For this method the distance can be specified by providing a Double value indicating the distance to extrude or you can supply a String that represents the equation you want to assign to the parameter that controls the distance. The ExtentDirection and Operation arguments expect values from enum lists. The TaperAngle argument is enclosed within brackets. This indicates that this argument is optional. In this case, not providing this argument will default to a taper angle of 0. For optional arguments you should look in the online help to find out what the behavior when you don't supply a value and what the expected type is if you do supply a value.

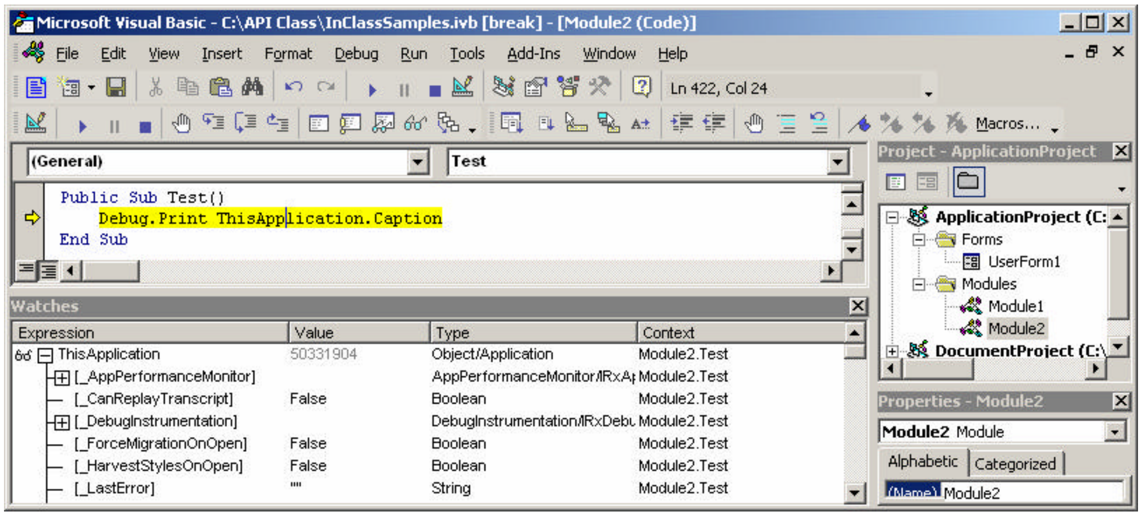


Function **AddByDistanceExtent**(*Profile* As Profile, *Distance*, *ExtentDirection* As PartFeatureExtentDirectionEnum, *Operation* As PartFeatureOperationEnum, [*TaperAngle*]) As ExtrudeFeature
Member of Inventor.ExtrudeFeatures
Use F1 key to display help topic

Using the VBA Debugger

Another very useful tool when working with Inventor's API is the VBA debugger. In this case we're not using it to debug a program but using some features of the debugger to better understand Inventor's object model. To use the debugger we need to be running a program. Fortunately we don't need much of a program to enable access to the Inventor objects. The specific steps to use the debugger are listed below.

- Within any code module, create a sub, like the sub Test shown below.
- Click the mouse anywhere within the sub and press F8 to step into the code. You're now running the sub one line at a time.
- Click anywhere within the word "ThisApplication".
- Run the Quick Watch command from the Debug menu and click "Add" in the Quick Watch dialog.
- The debug window will appear and ThisApplication will appear within the debug window. This represents a live view of the Inventor Application object. You can click on the "+" sign next to ThisApplication to expand it and view its properties and their values. Properties that return other objects will also have the "+" sign next to them allowing you to continue to expand the objects and view their properties. This provides a live view of the object model.



```
Public Sub Test()
    Debug.Print ThisApplication.Caption
End Sub
```

Expression	Value	Type	Context
ThisApplication	50331904	Object/Application	Module2.Test
[_AppPerformanceMonitor]		AppPerformanceMonitor/IRxA	Module2.Test
[_CanReplayTranscript]	False	Boolean	Module2.Test
[_DebugInstrumentation]		DebugInstrumentation/IRxDebu	Module2.Test
[_ForceMigrationOnOpen]	False	Boolean	Module2.Test
[_HarvestStylesOnOpen]	False	Boolean	Module2.Test
[_LastError]	""	String	Module2.Test

Programming Help

Delivered with Inventor is programming help. There are several ways to access help. First, by selecting **Programming Help** from Inventor's Help menu. Second is from within VBA itself. If you select the name of an object or function within the code window and press F1, Inventor will take you to help for that object or function. Also, as discussed earlier, you can press F1 from the Object Browser to go to help for the currently selected object or function.

The online help provides descriptions of each object, method, property, event and enum. In addition there are several overviews that discuss specific topics within the API. Finally there are sample programs that demonstrate a lot of the API. These samples are all written using VBA and can be copied from the help window, pasted into a VBA code module, and executed.

Inventor's SDK

When you install Inventor one of the directories created is the SDK (Software Development Kit) directory. In addition to the online help already described, this directory contains a significant number of programming samples. These samples are written in several different programming languages and demonstrate broader workflows than the samples that are part of the online help. The SDK directory also contains several Add-ins written specifically for VBA developers, specially the AutoCustomize sample.

Additional Resources

There are some additional resources for Inventor's API.

The Autodesk website www.autodesk.com/developinventor contains some additional overview material and the object model chart. This site also contains presentations that were used to illustrate the new API features of each new release.

One of the best resources is the "Autodesk Inventor Customization" newsgroup. By selecting the "Discussion Groups" link on the first page of www.autodesk.com you can navigate to the Inventor Customization group. The newsgroup is primarily for peer support between customers but Autodesk employees also participate in this group.

There are training classes taught by the ADN (Autodesk Developer Network) group. These are primarily intended for ADN members but non-members can also attend. Schedules and pricing can be found at www.autodesk.com by following the "Developers" link on the first page.

Another option for training is on-site consulting. This allows the training to focus on your specific needs while working on issues related to tasks you want to accomplish. Information about consulting can also be found at www.autodesk.com.

Finally, there are also some good resources outside of Autodesk. The following are some websites that have information about the API and provide access to some applications that were written using the API.

http://www.sdotson.com

http://www.cbliss.com

http://www.kwikmcad.com

http://www.mymcad.com