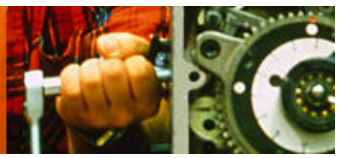


Unleashing Hidden Powers of Inventor with the API

Part 3. Uncovering the hidden documents with Inventor API

Brian Ekins – Autodesk, Inc.

This article introduces the basic concepts you need to understand to access Inventor documents.

Key Topics:

- Understand how Inventor provides a programming interface
- Object-oriented programming
- Inventor's object model
- Basics of Documents portion of the object model

Target Audience:

Autodesk Inventor users who want to increase their productivity with Inventor by writing programs . It assumes that you are already familiar with programming and know basics of how to use VBA, (Please see “Getting started with Inventor VBA” for tips on using VBA)

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.

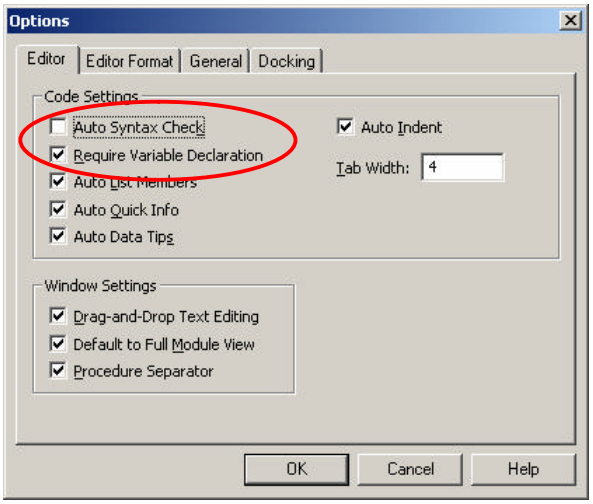
brian.ekins@autodesk.com

Introduction

The goal of the third article in this four-part series is to get you excited about exploring below the surface of Inventor by writing small programs. We'll begin by looking briefly at VBA. Then we'll discuss the concepts around Inventor's programming interface. Next we'll look at the interface for programming iProperties. Finally, we'll look at several examples ...

VBA Revisited

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. 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.



The Basics of Inventor's Programming Interface

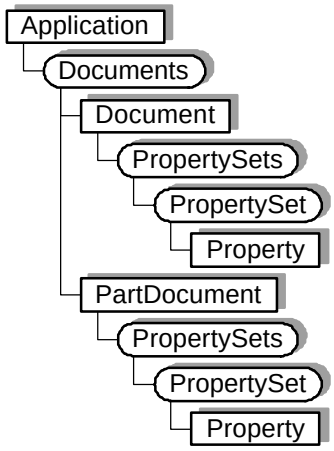
Inventor supports the ability for you to automate tasks by writing programs. Inventor does this using some Microsoft technology called COM Automation. This is the same technology used when you write macros for Word or Excel. The technology itself isn't important, but what is important is what it allows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

Objects

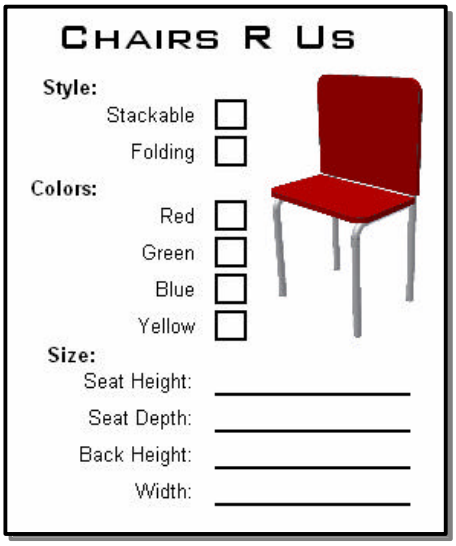
A COM Automation interface exposes it functions through a set of *objects*. A programming object has many similarities to real-world objects. Let's look at a simple example to understand how object oriented programming terminology can be used to describe a physical object.



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 you don't need to do anything but use this variable.

Apprentice

There's one other technique of accessing some of Inventor's functionality. There's a utility component referred to as *Apprentice* that provides a programming interface to a small portion of the information in an Inventor document. Apprentice doesn't have a user-interface and is exclusively a programming component. Design Assistant and Inventor View are both programs that use Apprentice to access document information. iProperties are one of the things that you have access to through Apprentice.

There are a couple of reasons to use Apprentice when possible. First, programs using Apprentice will run much faster than the equivalent program in Inventor. This is because Apprentice doesn't have a user-interface and it only provides partial access to information in the document so it doesn't need to load as much of the document as Inventor does. Second, Apprentice is freely available by installing Inventor View, (which includes Apprentice), from autodesk.com.

Derived Objects

The idea of *derived* and *base class* objects is usually a new concept for most end-users wanting to use Inventor's API. It's not a critical idea to understand but is a useful concept that we'll take advantage of when writing many of the following samples. To help describe this concept let's look at a close parallel; animal taxonomy or classification. For example, within the Animal kingdom you have insects, birds, mammals, etc. Within the mammal classification there are many different species but all of them share the same characteristics of a mammal; have hair, produce milk, etc. Even though an elephant and a dolphin are distinctly different they can both still be called mammals and share those same traits. This same idea can be used to understand the concept of derived and base class objects.

Let's look at how this concept applies to Inventor. Inventor has base class objects and derived objects. An example is the Document, PartDocument, AssemblyDocument, and DrawingDocument objects. The base class object is the Document object. The Document object supports all of the common traits that all documents share. The PartDocument, AssemblyDocument, and DrawingDocument objects are derived from the Document object and inherit these traits. They support everything the Document object supports plus they have additional methods and properties that are specific to that particular document type. For example, from the Document object you can get the filename, referenced documents, and iProperty information. From a PartDocument object you can get all of that, plus you can get sketches, features, and parameters.

The usefulness of having derived objects is demonstrated in the previous sample code. Notice that the variable invDocument is declared to be type Document. The program iterates through the contents of the Documents collection and assigns each item to this variable. It doesn't matter if the document is a part, drawing, or an assembly since they are all derived from the Document object.

Exploring Inventor documents

From the Document object, you can access the contents, properties, and other objects associated with that particular document. Before accessing any of those objects, you need to first access the document. Lets look at some different ways of accessing documents using the programming interface. This will let you look at the documents are not even visible or are hard to find otherwise.

Accessing Documents

There are a few different ways you might want to access a document:

- Accessing through the Documents collection (which we've already seen in the previous samples).
- Access the document the end-user is currently working on in Inventor.
- Open a specific document that's been saved on disk.
- Create a new document
- Through a reference from another document.
- Open a document in Apprentice.

These are demonstrated in the code samples below. All of these samples, except for the Apprentice sample are shown using Inventor's VBA, and take advantage of the ThisApplication global variable it provides.

Accessing the Active Document

This sample gets the document currently being edited by the end-user. It uses the ActiveDocument property of the Application object.

```
Public Sub ActiveDocumentSample()
    ' Declare a variable to handle a reference to a document.
    Dim invDoc As Document

    ' Set a reference to the active document.
    Set invDoc = ThisApplication.ActiveDocument
    MsgBox "Got document: " & invDoc.DisplayName
End Sub
```

Accessing all the open documents

This sample gets the documents that are currently open – including the ones that are hidden. It uses the Documents property of the Application object.

```
Public Sub AllDocumentsSample()
    ' Declare a variable to handle a reference to a document.
    Dim invDoc As Document

    ' Look at all the open documents
    For each invDoc in ThisApplication.Documents
         MsgBox "Found a document: " & invDoc.DisplayName
     Next
End Sub
```

Opening a Document from Disk

This sample opens a document on disk and returns a reference to the open document. It uses the Open method of the Documents collection object.

```
Public Sub OpenDocumentSample()
    ' Declare a variable to handle a reference to a document.
    Dim invDoc As Document

    ' Open a document.
    Set invDoc = ThisApplication.Documents.Open("C:\Temp\Part1.ipt")
    MsgBox "Got document: " & invDoc.DisplayName
End Sub
```

Creating a New Document

This sample creates a new part document. It uses the Add method of the Documents collection and uses the FindTemplateFile method to get the filename of the standard part template.

```
Public Sub CreateDocumentSample()
    ' Declare a variable to handle a reference to a document.
    Dim invDoc As Document

    ' Create a new part document using the standard part template.
    Set invDoc = ThisApplication.Documents.Add(kPartDocumentObject, _
                  ThisApplication.FileManager.GetTemplateFile(kPartDocumentObject))
    MsgBox "Created: " & invDoc.DisplayName
End Sub
```

Through a Reference from another Document

When you open an Inventor document, Inventor may open many other documents behind the scenes. It is possible to access documents that are referenced from the active document. This is a very simple sample that assumes that you did not skip or suppress any of the documents.

```
Public Sub AllReferencedDocumentsSample()
    ' Declare a variable to handle a reference to a document.
    Dim invDoc As Document

    ' Set a reference to the active document.
    Set invDoc = ThisApplication.ActiveDocument
    MsgBox "Got active document: " & invDoc.DisplayName

    Dim refDoc As Document

    ' Look at all the open documents
    For each refDoc in invDoc.AllReferencedDocuments
         MsgBox "Found a referenceddocument: " & refDoc.DisplayName
     Next
End Sub
```

Opening a Document using Apprentice

This example illustrates opening a document using Apprentice. Apprentice cannot be used within Inventor's VBA, so this sample must be run from either Visual Basic or VBA within another product, i.e.

Excel. Within Apprentice, documents are represented by the ApprenticeServerDocument object. It is slightly different than the Document object but serves the same purpose and provides access to the iProperties of the document.

```
Public Sub ApprenticeOpen()
    ' Declare a variable for Apprentice.  Notice that this uses the "New"
    ' keyword which will cause a new instance of Apprentice to be created.
    Dim invApprentice As New ApprenticeServerComponent

    ' Open a document using Apprentice.
    Dim invDoc As ApprenticeServerDocument
    Set invDoc = invApprentice.Open("C:\Temp\Part1.ipt")
    MsgBox "Opened: " & invDoc.DisplayName

    ' Close the document and release all references.
    Set invDoc = Nothing
    invApprentice.Close
    Set invApprentice = Nothing
End Sub
```

Changing Properties using Apprentice

Now that we are on the topic of Apprentice, let us quickly look at how it is possible to only save the iProperty changes to the document with Apprentice, which is much faster than saving the entire document. The sample below demonstrates this by opening a document using Apprentice, changing a property value, saving the change, and closing everything. The FlushToFile method of the PropertySets object saves any iProperty changes. Remember that Apprentice cannot be used from within Inventor's VBA. It must be from another application's VBA or from Visual Basic.

```
Public Sub ApprenticeUpdate()
    ' Declare a variable for Apprentice.
    Dim invApprentice As New ApprenticeServerComponent

    ' Open a document using Apprentice.
    Dim invDoc As ApprenticeServerDocument
    Set invDoc = invApprentice.Open("C:\Temp\Part1.ipt")

    ' Get the design tracking property set.
    Dim invDTProperties As PropertySet
    Set invDTProperties = invDoc.PropertySets.Item("Design Tracking Properties")

    ' Edit the values of a couple of properties.
    invDTProperties.Item("Checked By").Value = "Bob"
    invDTProperties.Item("Date Checked").Value = Now

    ' Save the changes.
    invDoc.PropertySets.FlushToFile

    ' Close the document and release all references.
    Set invDoc = Nothing
    invApprentice.Close
    Set invApprentice = Nothing
End Sub
```