



Unleashing Hidden Powers of Inventor with the API Part 2 of 4: Using iProperties with Inventor VBA

Brian Ekins – Autodesk, Inc.

The second article in this four-part series provides an introduction to the basic concepts you need to understand when programming Inventor. These concepts are put into practice as we use API to query and modify iProperties using VBA.

Key Topics:

- Understand how Inventor provides a programming interface
- Inventor's object model
- Using API to query and modify iProperties of a document

Target Audience:

Autodesk Inventor users who want to increase their productivity with Inventor by writing programs

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

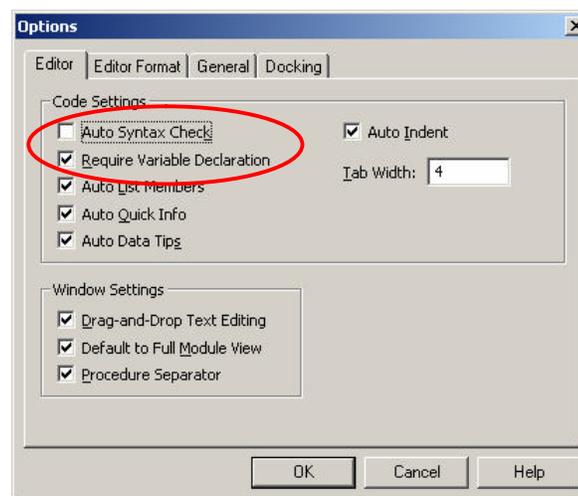
Introduction

The goal of this class is to provide enough information that anyone will be able to write a program to automate working with Inventor's iProperties. The class is not intended to be a comprehensive coverage of VBA programming or even iProperties, but focuses on the features that are most commonly used. We'll begin by looking briefly at VBA and the steps required to begin writing a program. 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 that demonstrate all of these concepts.

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. If you are unfamiliar with VBA please see the companion article, "Getting started with Inventor VBA".

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.

Unleashing hidden powers of Inventor with the API

The topics below discuss the basic concepts you'll need to understand when working with Inventor (or any COM Automation programming interface).

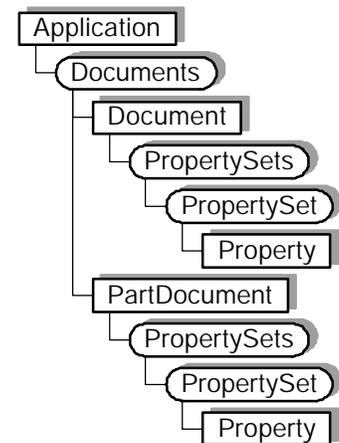
Objects

A COM Automation interface exposes its 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 object's methods. The first step in manipulating any object is getting access to the object you want. This is explained in the next section.



CHAIRS R Us

Style:

Stackable

Folding

Colors:

Red

Green

Blue

Yellow

Size:

Seat Height: _____

Seat Depth: _____

Back Height: _____

Width: _____



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.

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.

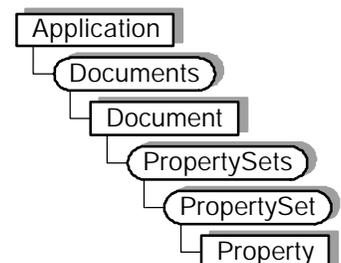
The primary job of a collection object is to provide access to a group of related objects (the set of children for that parent). For example, the Documents object provides access to all of the documents that are currently open in Inventor. A collection provides access to its contents through two properties; *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.

Unleashing hidden powers of Inventor with the API

```

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.

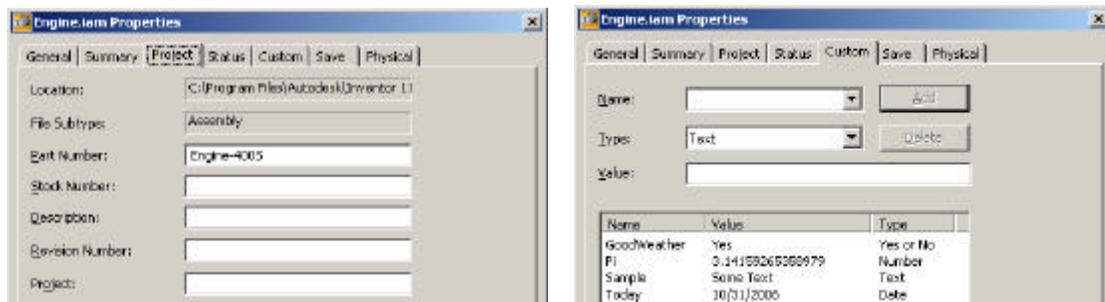
Another important feature of many collections is the ability to create new objects. For example the Documents collection supports the Add method which is used to create new documents. It also supports the Open method which is used to open existing documents from the disk. These will be used in some of the samples that follow.

Programming Inventor's iProperties

iProperties in the User-Interface

Before looking at the programming interface for iProperties let's do a quick review of iProperties from an end-user perspective. iProperties are accessed through the iProperties dialog. There are several tabs at the top that organize the available properties. There are also some tabs on this dialog that are not actually iProperties. For example, the General tab provides access to information about the document, the Save tab provides access to options that control how the thumbnail image is captured, and the Physical tab provides access to the various physical properties of the part or assembly document. If you need access to this information it is available through programming objects other than properties.

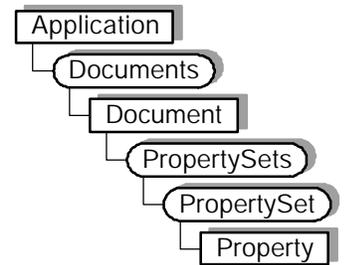
The picture below on the left illustrates a typical tab of the iProperties dialog where you have access to a specific set of iProperties. Through the dialog, you can view and edit the values of iProperties. The set of iProperties shown on each tab are predetermined by Inventor and cannot be changed. However, using the Custom tab (shown in the picture below on the right) you can add additional iProperties to the document. This allows you to associate any information you want with the document.



Using iProperties with Inventor VBA

iProperties in Inventor's Programming Interface

The object model for iProperties is shown to the right. The diagram starts with the Application object and goes through the Documents collection to get the Document object. Remember that the Document object is the base class for all document types so it can represent any type of Inventor document. From the Document object we then access the PropertySets object which is the top-level property related object and provides access to the various iProperties associated with that particular document. Before we look at accessing the iProperties lets look at some different ways of accessing documents using the programming interface.



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
  
```

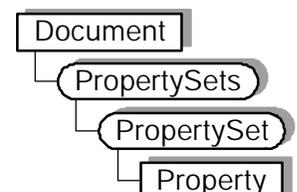
Accessing iProperties

Accessing iProperties through Inventor's programming interface is reasonably simple. The first thing to notice is that they're accessed through the Document object. Properties are owned by documents and to get access to properties you need to go through the document that owns them. The PropertySets object serves as the access point to iProperties. The PropertySets object itself doesn't have a name but is simply a utility collection object that provides access to the various PropertySet objects. Using the Item method of the PropertySets object you'll specify which PropertySet object you want. The Item method accepts an Integer value indicating the index of the PropertySet object you want, but more important, it also accepts the name of the PropertySet object you want.

PropertySet Objects

The PropertySet object is a collection object and provides access to a set of iProperties. The PropertySet object is roughly equivalent to a tab on the iProperties dialog. The Summary, Project, Status, and Custom tabs of the dialog contain the iProperties that are exposed through the programming interface. There are four PropertySet objects in an Inventor document; Summary Information, Document Summary Information, Design Tracking Properties, and User Defined Properties.

PropertySet objects are named so that you can find a particular PropertySet object. The Name property cannot be changed, is guaranteed to be consistent over time, and is an English human readable string. The Display Name is not guaranteed to remain constant. The Display Name is the localized version of the name and will change for each language. Below is an example of obtaining one of the PropertySet objects. In this case the summary information set of iProperties.



```

Public Sub GetPropertySetSample()
  
```

Unleashing hidden powers of Inventor with the API

```

' Get the active document.
Dim invDoc As Document
Set invDoc = ThisApplication.ActiveDocument

' Get the summary information property set.
Dim invSummaryInfo As PropertySet
Set invSummaryInfo = invDoc.PropertySets.Item("Inventor Summary Information")
End Sub

```

Property Objects

A Property object represents an individual property. The Name is an English string that is guaranteed to remain constant. The following code gets the iProperty that represents the part number.

```

Public Sub GetPropertySample()
' Get the active document.
Dim invDoc As Document
Set invDoc = ThisApplication.ActiveDocument

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

' Get the part number property.
Dim invPartNumberProperty As Property
Set invPartNumberProperty = invDesignInfo.Item("Part Number")
End Sub

```

Something else you'll see in a lot of sample code is where several property calls are combined into a single line. The code below does the same thing as the previous sample but it's getting the PropertySet object returned by the PropertySets Item property and immediately calling the Item property on it to get the desired property.

```

Public Sub GetPropertySample()
' Get the active document.
Dim invDoc As Document
Set invDoc = ThisApplication.ActiveDocument

' Get the part number property.
Dim invPartNumberProperty As Property
Set invPartNumberProperty = invDoc. _
    PropertySets.Item("Design Tracking Properties").Item("Part Number")
End Sub

```

Now that we have a reference to a specific property we can use its programming properties to get and set the value. The Property object supports a property called Value that provides this capability. For example, the line below can be added to the previous sample to display the current part number.

```
MsgBox "The part number is: " & invPartNumberProperty.Value
```

This next line sets the value of the part number iProperty.

```
invPartNumberProperty.Value = "Part-001"
```

Creating Properties

Interactively, as shown in the previous picture, you specify the type of property you're going to create; Text, Date, Number, or Yes or No. When you create them using Inventor's programming interface you don't explicitly specify the type but it is implicitly determined based on the type of variable you input.

New iProperties can only be created within the Custom (user defined) set of properties. New iProperties are created using the Add method of the PropertySet object. The sample below illustrates creating four new iProperties, one of each type.

```
Public Sub CreateCustomProperties()
    ' Get the active document.
    Dim invDoc As Document
    Set invDoc = ThisApplication.ActiveDocument

    ' Get the user defined (custom) property set.
    Dim invCustomPropertySet As PropertySet
    Set invCustomPropertySet = invDoc.PropertySets.Item( _
        "Inventor User Defined Properties")

    ' Declare some variables that will contain the various values.
    Dim strText As String
    Dim dblValue As Double
    Dim dtDate As Date
    Dim blYesOrNo As Boolean

    ' Set values for the variables.
    strText = "Some sample text."
    dblValue = 3.14159
    dtDate = Now
    blYesOrNo = True

    ' Create the properties.
    Dim invProperty As Property
    Set invProperty = invCustomPropertySet.Add(strText, "Test Test")
    Set invProperty = invCustomPropertySet.Add(dblValue, "Test Value")
    Set invProperty = invCustomPropertySet.Add(dtDate, "Test Date")
    Set invProperty = invCustomPropertySet.Add(blYesOrNo, "Test Yes or No")
End Sub
```

Putting it all together...

Let us change the Designer property of all the open documents to John Doe Inventor.

```
public Sub AddPropertiesToAllOpenDocuments()
    ' 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

        ' Get the Designer property.
        Dim invDesignerProperty As Property
```

Unleashing hidden powers of Inventor with the API

```
Set invDesignerProperty = invDoc. _  
    PropertySets.Item("Design Tracking Properties").Item("Designer")  
invDesignerProperty.Value = "John Doe Inventor"  
  
Next  
End Sub
```

Saving Properties

To save any changes you make to properties you need to save the document. The Save method of the Document object does this.

```
invDocument.Save
```

What next...

Read the next part of the article on "Working with properties..." to understand the many nuances and details of properties that we glossed over in this introduction.