11/30/2005 - 10:00 am - 11:30 am Room:N. Hemispheres (Salon B/C) (Dolphin)

AUTODESK UNIVERSITY 2005
Walt Disney World Swan and Dolphin Resort
Orlando, Florida

Basics of the Autodesk® Revit® Building API

Danny Polkinhorn - WATG

BD32-2 Use the power of building information modeling (BIM) to drive your own third-party applications. This class will cover the basics of using Visual Basic, VB.NET, and the new Revit API to link to a Revit model and get information from it. You will learn how to enable your application in Revit, and develop the hooks needed to get to the vast information already in your Revit models.

About the Speaker:

Danny is CAD manager at Wimberly Allison Tong & Goo in Honolulu, where he provides training and support for Revit and AutoCAD. Previously, he was CAD manager at Perkins + Will in Atlanta, where he successfully implemented Architectural Desktop. Danny received his Masters of Architecture from Georgia Tech.

dpolkinhorn@watg.com

Introduction

The release of Revit 8.0 in April, marked the first release that included a .NET Application Programming Interface (API). This had been a long-standing request from those who migrated from AutoCAD and desired the power to personalize Revit for their companies in a way that they were accustomed to with AutoCAD. Revit is a more integrated solution for the AEC industry than AutoCAD's more general 'backbone' approach that serves a variety of industries. As such, Revit developers felt that specific tools should be made part of the product, freeing users to concentrate on buildings and not on developing customization. While this has resulted in a very complete product, there remained a segment of users who needed to integrate Revit with the custom database applications that might be required of the specialized types of building their offices produced. Integrating Revit with external applications remains the focus of the API, while Revit itself concentrates on the design and documentation of the buildings. This approach is fundamental to the goal of 'Building Information Modeling', having a central location for information that is accessible to other applications.

This class will show you how to gain access to the vast amount of information in your models, as well as how to integrate external applications with Revit for two-way transfer of data. The power of Building Information Modeling is now in your hands.

Note: Some terms which might not be familiar to Revit users who aren't developers will be marked with an asterisk and are defined at the end of this document.

Planning

There are a couple important points I should mention before we jump in. First, do you even need the Revit API?* You might not need a custom application using the API if you can work with the exports that are already available from Revit. The ODBC export includes most of the relevant information about the model in separate tables by element type. Unfortunately, the transfer is one way: out of Revit. You cannot subsequently bring data modifications back into your model (without the API). Also, the scheduling features within Revit are very powerful, and users have been using this since the first release to generate tabular data views. Schedule views can be exported to Excel for further manipulation and formatting using tools that most non-Revit users are familiar with. Again, without the API this data transfer is one way out of Revit. IFC support in Revit allows you to export in an industry standard open format that can be used across the building life-cycle. You might want to consider IFC file support if your application needs to work with models generated by other BIM solutions.

Most (probably all) database developers will tell you that trying to coordinate two databases of the same information adds a tremendous amount of complexity to a project. How do you synchronize the data between the two databases when there are changes in one of them? Is the data updated when you save the one you changed, or when you open the second one? What happens if two users change the same data in both databases at the same time? Because of these complex questions, I suggest you use the Revit model to store information as it will eliminate redundant/out-of-sync data. The Revit project file is a very powerful database in its own right. Use your external application to present the information in the format you need, but leave the data itself in Revit.

The Revit API for .NET* will be much easier for you to use than COM because it supports both overloaded functions* and Intellisense* in the editor. While the COM* version is identical in content, it will be difficult to use during development, and may not be supported in the future. For that reason, I suggest using a .NET language and development tool.

Remember that Revit 8.1 includes version 1.1 of the API. While there is a lot of information you can get from your model, the API is still in its infancy. Take the time to familiarize yourself with the Object Model* before you try to tackle a large application. Some critical piece you need may not be in the API yet. If you are a commercial developer, I strongly suggest you become a member of the Autodesk Developer Network (ADN). There you will have access to Revit developers who can provide the interfaces you need in subsequent releases of the API, as well as the support of fellow developers in private newsgroups.

Like any other .NET application, the .NET Framework needs to be installed on every computer that uses your application. The Revit API requires version 1.1 of the .NET Framework or later.

Originally the API was developed for and introduced in Revit Structural. Revit needed to integrate with a variety of well-established structural analysis tools. Developing an API allowed the Structural product to work with these tools in a very robust way, so the structural features of the API are much more developed than those elements that were originally part of Revit Building. You will have more luck adding and modifying structural columns, for instance, than if you needed to add a wall and sweep a profile on it (which you can't do at all). Currently with Revit Building 8.1, the only element type you can add is a family instance, a group or a symbol (family type). You can also load a family into a project along with some or all of its types (symbols). Interfaces for the elements in Revit Building will certainly be added in the near future.

API files

The Revit API consists of only three files. You can find all three in the Program directory where you have Revit installed, C:\Program Files\Autodesk Revit Building 8.1\Program is the default for Revit Building.

RevitAPI.dll	Reference this file if you're using .NET. This is the complete API for Revit Building and Revit Structural. While the API includes methods for all Revit verticals, you can only use those functions that are part of the installed product. For example, you can't place structural loads in your project if your application is running in Revit Building.
RevitAPI.tlb	Reference this file if you're using COM. It contains the same methods as the DLL, but is formatted for COM. For example, overloaded functions in .NET will appear as separate methods in COM because COM doesn't support overloading.
RevitAPI.chm	A compiled help file for the API. All objects, methods, events, and properties* are listed here with some additional documentation. No examples are included.

Revit API Object Model

The object model includes several base classes* that all objects in the API are derived from. These base classes provide a majority of the functionality, with specific objects inheriting from them. See the RevitAPI.chm help file in the Revit Program directory for more information about specific objects and the complete object model.

First, I need to clarify the term 'Element' in this document. In the Revit Object Model, all things in the model are Element objects. Because of this I'll use the term Element for things in the model instead of the more generic term Object.

Here are some of the objects available in the Revit API and some descriptions for their usage.

- Revit – The major base classes. All of the remaining objects, such as Revit.Collections are derived from these base classes. I'll highlight some of the most important ones.

Application	With the application object, you can obtain the ActiveDocument and change some application options.
Category	The category will tell you what type of element something is, such as Door, Symbol, or WallType. When using ODBC export, a separate table is setup for each category. You can also access the parent category or sub-categories.
Document	The Document object (the Revit Project) allows you to access specific elements in the model, such as a particular door. You can move or rotate elements, load new families, determine the file name and path of the project, or get the active selection with this object.
Element	All physical objects in your model inherit from this class so they can share common properties. Some of these properties are Category, Level, Location, Parameters, and PhaseCreated.
Location, LocationCurve, LocationPoint	These objects give you the physical location or the start point and end point of an element in the project.
Parameter	This object gives you access to most of the parameters of an element that are available in the Properties dialog, both Instance and Type parameters. For a door, these would include the Width, Height, Mark, Fire Rating, etc. A majority of your applications will use this object type to obtain information from the model.

- Revit.Collections – Different collection objects to allow you to iterate through groups of objects in the model. Each collection type has a corresponding iterator* that you should use for that collection.

Array	Ordered set of elements. Insertion of elements at specific points in the array is possible. Use this collection when the order of elements is important, such as layers of a wall, phases of a project, or lines/arcs in a profile.
DefinitionBindingMap	A mapping of parameter definitions to the element type. Use this to determine if an element contains a specific parameter.
Map	Generic map collection. You can map any object type to any other object type for relational comparison.
Set	A Set is a generic unordered collection. The fastest type of collection to iterate.
StringStringMap	A mapping of two strings for comparison.

- Revit.Creation – This base class allows you to add elements to the model. Currently with Revit Building 8.1, the only element type you can add is a family instance or a group. With Revit Structural 2, different load conditions can also be added.

Application	The application object contains a series of creation tools for items such as new collections (Arrays, Maps, and Sets).
Document	The Document object allows you to create actual elements in the current project, such as a new family instance.

- Revit.Elements – This class contains different elements that are part of your model, for example Grids, Levels, and Walls. Many of the Load cases in Revit Structural can be found here. Because of the vast number of elements here, I'll highlight only a few of the most important, and some of their properties.

FamilyInstance	This element is a particular family instance in your model. Here you can determine the instances location, what room it's in, and the symbol it's using. If it's hosted you can also obtain the hosting element, such as obtaining the wall that's hosting a door instance.
Grid	Get/Set the location of a grid line in the model.
Level	Get/Set the elevation of a level.
PhaseArray	With this element, you can see the phases in the project as well as insert new ones.
Parameter, Parameters	These are the key methods for obtaining data from an element. All parameters, system, type, and instance are found here.
Room	You can get the physical boundary of a room with this object.
Wall	Get/Set the Structural Usage, WallType, or Width of a wall.

- Revit.Events – This class has only one usable event, OnDialogBox. This event occurs whenever Revit displays a dialog box while your code is running. You can overload this event and return specific responses to the dialog without user intervention.
- Revit.Geometry – This base class contains all the objects needed to describe the geometry of an element in your model, such as XYZ of a point, a parametric curve, or a face. Using these objects, you could potentially write a tool to export a particular file type by grabbing all the coordinate data, either 2D or 3D.

Edge, Face, Solid	These objects are used for massing elements. From the solid, you can obtain all the edges and all the faces. This interface could be used for generating an export tool to other 3d formats.
Element, Instance	Used for obtaining the individual geometry objects from elements or instances in your model.

Curve, Line, Arc	Linear objects Line and Arc are derived from the Curve class. The individual lines and arcs can be obtained from a Profile and are returned in a CurveArray.
Options	For obtaining the geometry in elements and instances, you'll need to tell Revit which detail level to return. The Options object is used in this situation.
UV, UVArray	2d points or vectors on a surface.
XYZ, XYZArray	3d points in space, for example start points and end points on a curve.

- Revit.Parameters – Access to all parameter items and their values can be found within this base class.

Binding, TypeBinding, InstanceBinding	Bindings tell Revit which parameter definitions are attached to which objects in the project. For example, a Mark parameter is bound to both door and windows instances.
Definition	The name and type of parameter.
StorageType	This property tells you how the parameter value is stored. You will need this for type conversion among the different variable types.
Type	This property tells you what kind of value it is, Area, Length, etc, and is the same list you see when adding a parameter in your project.

- Revit.Rooms – Provides access to Boundary information for Rooms in your project.
- Revit.Structural – This class is the base class for the Analytical model and Compound Structural components in Revit Structural.
- Revit.Symbols – Symbols are Revit Types. With this base class, you'll have access to all the type information within the project.

FamilySymbol, FamilySymbolSet	A FamilySymbolSet is one family, for example "Single-Flush" for doors. A FamilySymbol is one type within a family, such as 36"x84". The FamilySymbol includes both model elements and annotation elements.
WallType	When you query a wall for its type, this object will be returned giving you more information about the wall type than just the name. Note that if you query a wall element for its Name, Element.Name, you'll get an empty string. All other elements return the Type Name.

Setting up a new project

Let's walk through setting up a "Hello World" project in Visual Studio.NET using the Revit API. In this exercise, you'll learn what the steps are for creating an application using the Revit API and the required components.

First, you don't need to use Visual Studio, any .NET editor will do. A copy of Visual Basic.NET Standard Edition is around $100, and should provide everything you need.

Also, you don't need to use Visual Basic.NET. Any of the .NET languages will work, as will any language that supports COM (or ActiveX). Some of the samples that come with Revit were written with VB.NET, and that's what we'll demonstrate here. The process is similar with other .NET languages. There are some syntax differences with the Revit API in COM, so I suggest you stick with .NET.

Let's start by launching VB.NET and creating a new project.

1. In VB.NET, go to File > New > Project. Under Visual Basic Projects, choose a Class Library. Specify a name for the project, and a location. The name will become the name of the DLL library, and you'll need this name later. For this example, we'll keep the default name of "ClassLibrary1".

   You'll need a reference to the Revit API. This gives you access to the methods and properties available within the API and also enables Intellisense in the editor.

2. Go to the Project menu and select Add Reference. On the .NET tab, click the Browse button in the upper-right. Browse to the directory where you have Revit installed, and find the "Program" directory. For Revit Building, the default location is "C:\Program Files\Autodesk Revit Building 8.1\Program". Find the "RevitAPI.DLL" and select it. Click OK to both dialogs. If you're using COM (ActiveX), select the "RevitAPI.TLB" in the same directory.

   On the View menu, choose Object Browser. Click the plus-sign next to Revit API and from there you can see all the methods and properties available to you. This section of the Object Browser will be critical for you to explore and familiarize yourself with. If you need to know what the API can and cannot do, this is where you find it.

   Also, in the Add Reference dialog, add any other components you need for your project. For example, if you're working with Excel, you'll also need to reference the Office Core library and the Excel library (both of these can be found on the COM tab).

   Switch back to the Class1.vb file. You can also use Solution Explorer to get back (View > Solution Explorer).

3. In Class1.vb, above the "Public Class Class1" line, add the line "Imports Autodesk.Revit". Additional Imports statements can also be placed there.

   This allows you to eliminate the fully qualified name for an object and will save you a ton of typing. So, instead of "Autodesk.Revit.Element", all you need is "Element".

4. Just beneath the "Public Class Class1" line, add the line "Implements IExternalCommand".

   IExternalCommand is an interface that you must reference in order for Revit to call your application. The code that was automatically added to your class is your key to the Revit model. The interface is the interaction point with Revit. All other code you write must pass through this interface. Within this section of code, you can reference other classes as you see fit, but this is where it all starts.

   Your code so far should look like this:

```
Imports Autodesk.Revit

Public Class Class1

    Implements IExternalCommand

    Public Function Execute(ByVal application As Autodesk.Revit.Application,
       ByRef message As String, ByVal elements As Autodesk.Revit.ElementSet) As
       Autodesk.Revit.IExternalCommand.Result Implements
       Autodesk.Revit.IExternalCommand.Execute

    End Function
End Class
```

The function "Execute" is the function that Revit runs when your tool is loaded and executed. Any function that Implements IExternalCommand.Execute can be substituted for the default "Execute" function. There are three parameters for the Execute function, Application, Message, and Elements. The return value is Result.

Application	The Application parameter gives you access to the Revit Application, any documents that are loaded, and any elements that are in those documents.
Message	The message parameter is actually something that you return to Revit. If your application fails, the Message will be displayed in an Error dialog after completion.
Elements	This is also something you return to Revit. If you return a Failed result, these elements will be highlighted to provide the user feedback. While this is Autodesk's intended result, I haven't been able to get this to work, nothing highlights. You won't receive any errors. I also haven't been able to obtain confirmation from Autodesk that this is supposed to work.
Result	The Result is where you tell Revit if your function completed successfully, was cancelled, or failed. If Successful, any changes you made to the Revit database will be kept. If Cancelled, any changes you made during execution will be removed from the database. The user will receive no intrinsic notification from Revit if the Message parameter is empty. If you return Failed, changes will be removed from the database, and the user will receive a "Cannot be ignored" error in Revit along with the text you return in the Message parameter.

Somewhere in your function you should set the Return value. If you don't use a Try…Catch block, set the Return value to Failed, and then set it again to Successful if your code finished properly. That way if an error occurs, any changes will be undone.

A better way is like this:

```
Try
   'Your code here instead
   MsgBox("Hello World")
   Return IExternalCommand.Result.Successful
Catch ex As Exception
   'Your error code and message here
   message = ex.Message
   Return IExternalCommand.Result.Failed
End Try
```

That's it; the rest of the code is up to you! Add your remaining code in the Try block and compile it.

Debugging options

There are some settings you can use in Visual Studio to make debugging your application a little easier. You can make Revit launch when you compile and even open a project. This allows you to create a test Revit project that's populated with elements to test your code on. It keeps you from having to draw a few walls each time you open Revit. Any break points you add in Visual Studio will automatically stop execution and switch to the Visual Studio screen when they are reached.

1. Go to the Project menu and select "ClassLibrary1 Properties".
2. Click on "Configuration Properties" in the left pane, then "Debugging".
3. Click the "Start External Program" radio button, then the Browse button. Browse to the Revit.exe file in the installation directory.
4. Under "Start Options", "Command Line Arguments", add the path to a Revit project you want to open.

Telling Revit about your application

You need to let Revit know about your app so that you can execute it from within Revit. Edit the Revit.ini file in the Program directory to give Revit the information it needs to display and execute your application properly. All external programs will be displayed in the menu under Tools > External Tools in Revit.

1. Open the Revit.ini from the Program Directory.
2. Add one header for all external applications. If the header already exists, add your application under the existing header.
   [ExternalCommands]
3. Add a parameter, ECCount, and set it equal to the number of applications you're adding. We'll use 1 for our Hello World app.
   ECCount=1

The remaining parameters should be repeated for each app, incrementing the number at the end for each one.

4. Add a parameter for the Name that appears in the menu.
   ECName1="Hello World"
5. Add a parameter for the Description that appears on the status bar at the bottom of the screen. This description is optional.
   ECDescription1="Say hello to the world."

6. Add a parameter for the Assembly which tells Revit where your DLL is located. The name of the DLL will match the name of your project, “ClassLibrary1” in our example. This can be a mapped location or a UNC path, as long as Revit can resolve it.
   ECAssembly1=”C:\Temp\ClassLibrary1.dll”
7. Finally, add a parameter for the Class name, or the name of the component. This should include the assembly name.
   ECClassName1=ClassLibrary1.Class1
8. Save the Revit.ini file.

So the completed section should look like this:

```
[ExternalCommands]
ECCount=1
ECName1=”Hello World”
ECDescription1=”Say hello to the world.”
ECAssembly1=”C:\Temp\ClassLibrary1.dll”
ECClassName1=ClassLibrary1.Class1
```

Additional apps would be added like this, incrementing the number for each one:

```
[ExternalCommands]
ECCount=3
ECName1=”Hello World”
…
ECName2=”Other app”
…
ECName3=”Third app”
…
```

You can also add a keyboard shortcut for your application by editing the KeyboardShortcuts.txt file in the Program directory. Follow the established syntax using the ECName as the menu item.

```
"HW"     menu:"Tools-External Tools-Hello World"
```

Some Things to Note for AutoCAD Developers

There are some procedures that developers may utilize when customizing AutoCAD that are not available in the Revit API for various reasons. I’ll let you know about some of them so you’ll have an idea about what to expect before you begin.

- There is no functionality that allows you to access a Revit project outside of Revit, similar to ObjectDBX or RealDWG. All Revit API applications must run while Revit is running, and must be executed by Revit from the menu. This ensures that the database isn’t corrupted during your program’s execution by relying on Revit’s internal check system.
- You cannot have the user interact with Revit during your program’s execution. Your application must complete before Revit will allow editing. Again, this is for database integrity.
- There are no filtered selection sets in Revit, and since the user can’t interact with Revit during your program’s execution, you’ll need to use one of two approaches to select elements. 1. Have the user select the target elements, then run your application. You can access the current selection set with Application.ActiveDocument.Selection. 2. Walk the database and grab the elements you’re interested in. For large Revit projects, this could take some time. You should consider checking the active Selection to see if the user has already done some work for you before you iterate the database.
- Because your app must finish execution before Revit will continue, there are no reactors or events you can trap.

- There are no database transactions you need to deal with since Revit handles these for you based on the return value of the Execute function.
- You cannot create custom objects in Revit like you can with ObjectARX and AutoCAD. All elements your application creates must be one of the built-in element types. Also, unlike AutoCAD-based vertical applications, Revit models do not contain objects that are proprietary to a particular vertical. This means that there will never be a need for object enablers, and you can share models among the different Revit verticals without issue. Your application will not need to be present on a machine in order for Revit to open the model and view any elements you've added.
- You will not be able to edit the User Interface in any way, which is probably a good thing. This is consistent with Revit practice since the beginning. At least you'll always know where to find your custom app.
- Instead of xData or Dictionaries, Revit uses Parameters. You can add hidden parameters to any element if you want to hide the data from the user. To add a parameter that isn't specific to any element, add it to the Project Information element.
- ElementIDs for elements in the model are subject to change during the course of a Revit session, unlike Handles in AutoCAD. Therefore, do not store ElementIDs in your external applications and use them for repeated calls to the database. Instead, add your application data or identifying information as a parameter to the element, then search for it when you need it.
- There is no property available which will tell you if your app launched in Revit Building or Revit Structural. There is a simple workaround. If you attempt to access a property that's not available, it will return 'Nothing'. For example, if you attempt to access the AnalyticalModel from Revit Building, it will return nothing instead of the actual AnalyticalModel that you would get from Revit Structural. This should be addressed in a future version of the API.

Other Information

There are no other written sources of information about the Revit API other than the API Help file included with Revit. It covers the complete object model, but it doesn't include any code examples.

Sample applications can be found on the Revit installation CD under Common\Revit SDK. They are commented and should provide a good basis for getting started. There is a mixture of VB.NET and C# samples.

I found "Windows Forms Programming in VB.NET" by Chris Sells to be one of the best reference books on forms programming. It's very comprehensive and should give you all the information you need about coding forms. ISBN# 0-321-12519-3

Also, Dan Appleman's "Moving to VB.NET" is a good guide for those VB programmers wanting to make the jump to .NET. ISBN# 1-893115-97-6

Official Autodesk support for the Revit API is only available from the Autodesk Developers Network. Visit www.autodesk.com/adn for more information.

Like most Revit users, I frequent AUGI's discussion forums because they are a valuable source of information. There's a "Revit – API" forum under AUGI Forums > AEC > Revit > Revit General Support. If you are not an ADN member, you can post questions there and get help from other users. Developers sometimes answer questions there. Please participate as much as you can and share code as well. We'll all benefit.

Glossary

API – Application Programming Interface, a way for one application to control and access another application and its documents.

COM – Component Object Model, an older set of technologies for application development. While COM is still very prevalent, there are some limitations to it that can only be overcome by using .NET. Microsoft is focusing all future development platforms on .NET and will eventually drop support for COM.

Intellisense – The ability for the development environment to present you with available members of an object as you program. It can greatly reduce the amount of typing you do, eliminate typing errors, and ensure that you don't use a feature that's not available on that object.

Iterator – An iterator is an object that allows you to move through a database and inspect (or iterate) each element. Think of it like an assembly line where each widget is stopped in front of you. You can do something to the widget, then move to the next widget, and repeat your procedure on the next one. You repeat the procedure until you've gone through all the widgets. An iterator allows you to iterate the entire Revit model, or just a portion of it. As each element is presented, you can do something to it, get information from it, or add information to it.

Object Model – A map of the available properties, methods, and events in an API.

Objects, Methods, Properties, Events – An Object is a self-contained programming element. Methods, Properties, and Events are 'members' of an object. In the real world let's compare an object to a car, a self-contained thing that we move around in. A Property is simply a property of the object, like Car.Color or Car.Manufacturer. A Method is a function you can perform on the object to make it do something, like Car.GoForward or Car.TurnLeft. An Event is something that occurs to the object, like Car.Stopped. So, if I had a Ferrari, I could order a red one with Car.Color = "Red", a property. To drive fast, use Car.GoForward(160 mph), a method. And when Car.Crashed, an event, then Call Police.

Class – the code that makes up an object. The class contains the code for all its methods, properties, and events. With .NET, Inheritance became possible. Simply, Inheritance allows you to create another class that uses the same methods, properties, and events of an object, AND in which you can add or change functionality without re-writing code. So with our car example from above, we could create a Ferrari class which inherits the Car class. But, in our Ferrari class we could add additional options like Ferrari.RacingMode that wouldn't be available in the basic Car. But, we don't have to re-write the code for Ferrari.Color, Ferrari.GoForward, and Ferrari.Stopped, because they inherit from Car.

Overloading – Allows a function to accept different parameter types. Consider a function named GoToLocation. A Location could be a street address or a set of Longitude and Latitude coordinates. So GoToLocation("SomeAddress") and GoToLocation(Longitude, Latitude) would allow the same function name to accept either parameter and not give you an error. Overloading is a feature of .NET and not COM.

.NET – Microsoft .NET is a set of technologies for developing applications for a variety of target platforms including Windows and the web. The .NET Framework is a component that is installed in Windows and it provides a common set of features for accessing the computer, Windows components, and other services.