**A Closer Look at the Revit® Database with the Revit API**

Mikako Harada – Autodesk, Inc.

**DE305-1** Do you work with the Revit API but find the internal structure of the Revit database unclear? Are you often frustrated by not being able to find the information that you are looking for? If so, this class is for you. In this presentation, we’ll take a close look at the internal structure of the Revit database, and analyze and describe it in a more structured manner. The class uses a tool called RvtMgdDbg -- a useful utility for learning and debugging the Revit API -- and explores what’s in the Revit Document and Elements. After attending this class, you’ll have a clear picture of Revit's database structure and be more productive when working with the Revit API.

**About the Speaker:**Mikako works as a consultant for the Developer Technical Services Team at Autodesk, providing API technical support to Autodesk Developer Network members worldwide for AEC products -- primarily AutoCAD Architecture and Revit Architecture. Prior to joining Autodesk, Mikako was a researcher at universities and private research labs and worked in the areas of interactive techniques, optimization and layout synthesis.

****

**Introduction**

Once you have learned the ABC of Revit API and have gone through the samples in SDK, the next question you may have is most likely about the structure of Revit objects and how they are organized in the project database. Even if you have been using the Revit API for some time, you may still be wondering what really is in the *elements* list.

In Revit, all the elements are bundled together as a single list of elements and are accessible through Document’s Element Iterator. Elements that represent graphical objects, such as walls, windows and doors, family types, views such as plans and sections, site, location, and other settings specific to a current project or document are all put together under the single list of *elements*. One of the main tasks for a Revit programmer is often to find out a way to identify the object that you are looking for among thousands of elements in the list.

In this presentation, we will take a closer look at the *elements* list in the attempt to give you a better picture of structure of Revit internals for future programming in Revit API.

**Elements list in Revit API**

The starting point to access a Revit model is through an ExternalCommandData object passed through as a the first argument of the Execute() method of an IExternalCommand derived class. As you have learned by now, there are two ways to access Revit elements from an external command:

* One way is to let the user select objects in UI first, then obtain a list of elements through ExternalCommandData.Application.ActiveDocument.Selection.Elements
* The other is to go through ExternalCommandData.Application.ActiveDocument.Elements

The figure 1 shows the location of Elements in the Revit Object Model. The second is the one we want to take a look here.

Figure 1. Elements in the Object Model

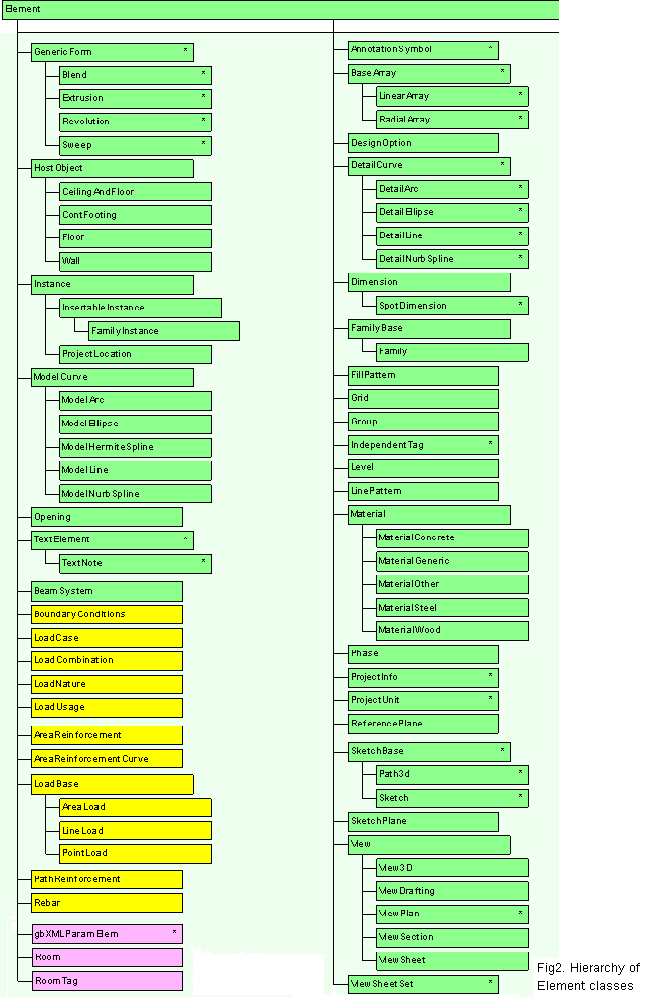
One thing to note is that these two “Elements” are returned in different data types:

* Document.Selection.Elements returns SelElementSet, which implements IEnumerable. We can use ForEach in VB.NET (or foreach in C#), Size, Contains test with this class.
* Document.Elements, returns ElementIterator, which implements IEnumerator. To access each element, we go through by iterator.MoveNext() and iterator.Current(). In real work programming, we often save obtained elements in the ElementSet for easier manipulation later.

A list of elements that you obtain from the document or Document.Elements contains thousands of elements. Even with a seemingly empty document that you have just opened using a default template, you will probably see more than 2000 elements in the Elements list. In typical programming, we can determine objects by their types or classes. Does this apply to Revit? Unfortunately, with Revit API, object’s class is not enough to identify what the given object represents in the Revit project database. Before we go further, let's first look at the classes defined Revit API.

**Revit Element Classes**

The following two figures (Figures 2 and 3) show the hierarchy of classes derived from Revit.Element. This is taken from "Revit API Diagram.rvt" included in the Revit SDK folder. There are 109 classes derived from Revit.Element currently. Elements in green show the common elements in all three Revit verticals (e.g., Wall, FamilyInstance, and FamilySymbol). Classes in yellow represent elements that are specific to Revit Structure (e.g., BoundaryConditions, various types of loads and rebar classes). Similarly pink for Revit Architecture (e.g., Room and RoomTag). (As of Revit 2008 releases, there is no Revit MEP specific API’s exposed.) Please notice that classes, such as, Level, ProjectInfo and Views, are also derived from Element class.



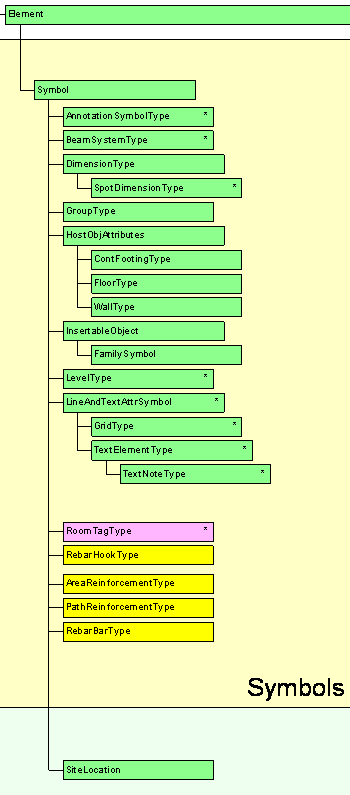


Fig.3 Hierarchy of element classes (cont.)

There is also a significant portion of element classes derived further from Symbols (Figure 3). As the name suggests, these are the classes representing family symbols or types. (Family types and symbols are used interchangeably here.) You can see here that SiteLocation is actually a special type. As you can see, the collection is rather large and a little difficult to read. So let us look at a subset of interesting classes and highlight some of them.

Figure 4 shows yet another view of a subset of classes, and some of interesting (or often used) classes are highlighted:

Figure 4. Subset of element classes in hierarchy

In Revit, APIObject is the base class of all the Revit API objects (with exception of few supporting classes, such as UV and XYZ in geometry name space). All the BIM objects are derived from Element. Wall and Floor (on the right) are HostObject, which means that they can host other objects such as windows and openings. FamilyInstance (on the left) is the class that represents an instance of Component Family. Columns, braces and furniture, such as desks, are examples of this class. WallType and FloorType (at the bottom) are classes representing types specifically for Wall and Floor object respectively. FamilySymbol is a class that defines a type of any kinds of component family. Roof (lower right) is shown is green with a dotted line here. There is no Roof class in the current Revit API. But it is also a HostObject.

**Built-in Category**

In Revit, doors, windows, columns, beams and furniture like desks are component families. The class defines these are FamilyInstance. If all the component family is an instance of FamilyInstance, then how do we distinguish some of architectural components? In Revit API, Element class has a property called “Category”. You can use this property to distinguish elements representing different family objects. Category is predefined. You can find the list of available categories in the RevitAPI Help file under Autodesk.Revit 🡪 “BuiltInCategory Enumeration”. There are 598 categories listed in total. You can access a specific category object passing built-in category id to Document.Settings.Categories.Item(). (Our experiment shows 397 out of 598 has actually associated categories in document categories.) However, not all the element has the category property defined. Some element may return *nothing/null* when you try to access it (e.g., Opening, Section.). You will always need to check to see if it is defined before you try to get the category object.

Basically, class names and category, these two are the bases of identifying an element from the massive list of element. All clear by now? Not really, right? What we have described so far is from the view of classes and property defined in Revit API. In the following sections, we want to step back a little bit and look at what’s in the element list more closely, and form a bigger picture of what’s in there and how elements are structured in Revit. The motivation to achieve such a goal was availability of a tool called RvtMgdDbg. We will talk about this tool a bit next.

**RvtMgdDbg – “Snoop” Tools**

RvtMgdDbg is a utility similar to ArxDbg and MgdDbg utilities (well-known among AutoCAD ObjectARX and .NET API developer community). RvtMgdDbg is written by the same author and his team. It is an external application which defines its own menu and a toolbar, and solely written using Revit API. It provides comprehensive tests of the Revit API, sample code and utility classes, scaffolding for quick tests. It is the powerful exploration tool for learning Revit internals. Figure 5 shows the main menu of RvtMgdDbg:

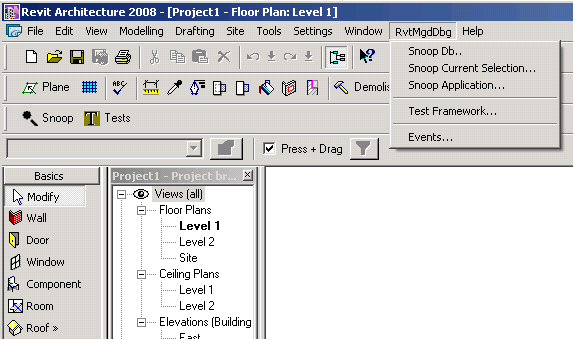


Figure 5 RvtMgdDvg menu

Currently five commands are available with RvtMgdDbg:

* Snoop Db --- this command allows you to view the Elements list under the current active document or ExternalCommandData.Application.ActiveDocument.Elements. It sorts elements by class name and display them in an alphabetical order. For each element selected, it displays its properties. Figure 6 shows the image from a sample usage of his command. Blue lines indicated the names of class as well as derived base classes. Bold letter in the item indicates the availability of the further “drill down” information. By click it, you can view more information.
* Snoop Current Selection --- this command allows you to view the elements you have selected to prior to running this command. The information displayed for each element is same as above.
* Snoop Application --- this command allows you to view the properties of application object.
* Test Framework --- this command consolidates various test commands including ones found in the SDK samples. You can also find more samples here.
* Events --- this command allows you to turn on/off event notifications currently available through Revit API.

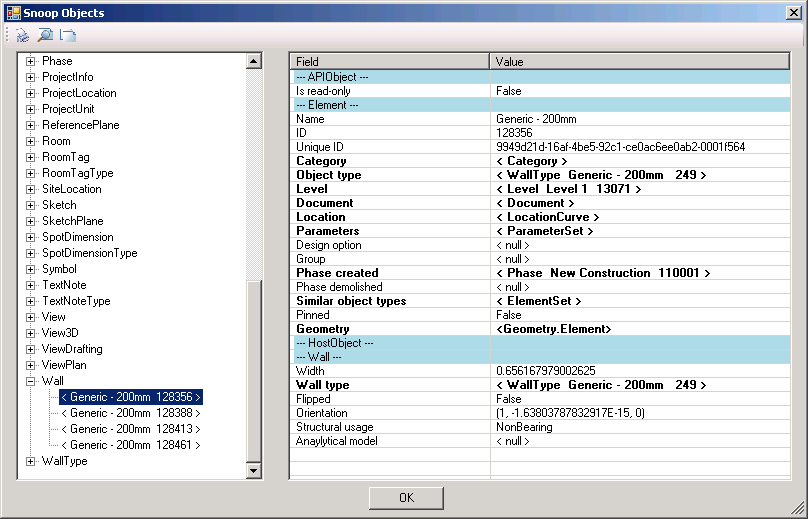


Figure 6. a sample usage of Snoop Db

You can find a word .doc file in Documentation folder. Please refer to it for the more description about this tool.

*Note:* The tool should be available for the participant of this class. If you are a member of ADN (Autodesk Developer Network), you can also download it from the ADN site.

**Closer Look at Element List**

If you look at Elements list using Snoop Db tool, soon you learn there are many elements in a Revit project drawing even when the drawing is just opened from a default template and it is seemingly empty. Draw a couple of walls, for example, then “snoop it”. You will now see a wall object is added under the “Wall” class name in the Snoop Object dialog window. Still you may wonder how many elements are there and what they are representing.

To answer these questions for myself, I have done a little experiment; taking a small (rather trivial minimum) sample .rvt project file of 20~30 basic elements added, “dumped” Elements list into a .txt file, and analyzed this “raw” data. Figure 7 shows the images of LittleHouse.rvt:

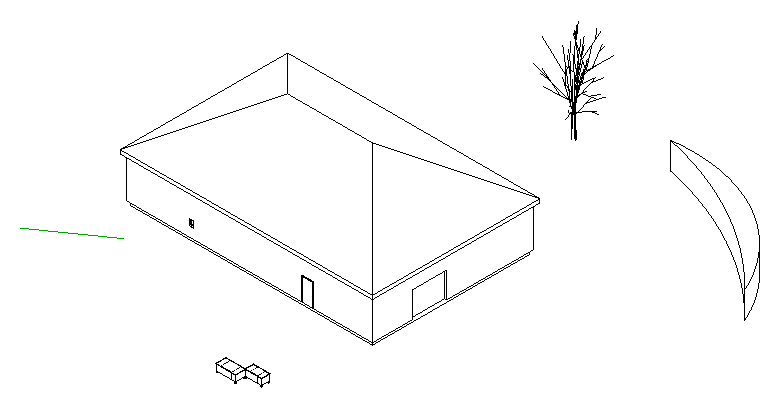


Figure 7. LittleHouse.rvt sample drawing

Information dumped into the txt file is:

* Element.Id,
* Element.GetType.Name
* Element.Category.name
* Element.Name

Figure 8 shows the image of the dump file:

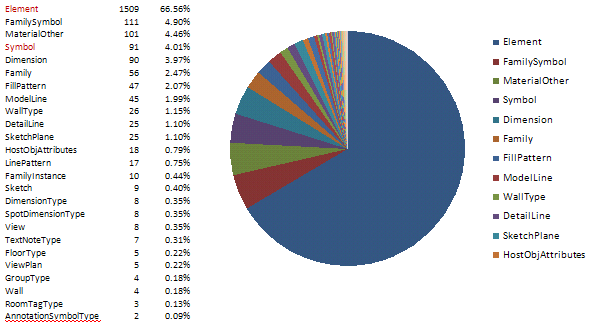


Figure 8 Elements list “dumped” in a .txt file

I then imported this “raw” txt file into an Excel file, explored with various sorting and observed the following:

* There are 2267 elements in total
* Every element has an Id
* Not every element has Category defined
* Not every element has Name defined. But sometime it has some meaningful string.

Further, figure 9 shows the number of each class in the list. Toward the top, you can see about 2/3 of elements are class Element, which means that they are not exposed. (Remember Element is a base class of all the elements in Revit.) A default template normally includes a number of typical family types. FamilySymbol and Symbol represent those family types. You see quite a few of pre-loaded family types in the list. If you go toward the bottom, you see more specific class names, such as FamilyInstance, Wall and various types.

Figure 9 The number of each classes Elements list

Next thing I did was to check those classes if they are further derived from Symbol or not, then grouped together under the associated UI (i.e., Modeling, Drafting, View, Families, Settings, etc.). Figures 10 and 11 show the results.

Figure 10 shows class names that represent non Symbol element. Figure 11 shows one for Symbols. As you can see, if you break down the element to this level, you can get a reasonable understanding what kind of elements are in there. In the figure, items in brown letters indicate elements that may worth additional comments. For example, Family class is actually a “container” class rather than an instance or definition of a type. It groups together component families. It is still a class derived from Element. We will come to this later.

Please note that this is by all means not an exhaustive list. This is merely a case study with the small sample mentioned above and to get the basic idea of elements list. For example, I have not included structural elements here. You should be able to easily extend the same idea to experiment with structural samples.

|  |  |  |
| --- | --- | --- |
| Non Symbol Element | View | View |
| View3D |
| ViewDrafting |
| ViewPlan |
| Settings | FillPattern |
| gbXMLParamElem |
| LinePattern |
| MaterialOther |
| MaterialSteel |
| Phase |
| ProjectInfo |
| ProjectLocation |
| ProjectUnit |
| Modeling | FamilyInstance |
| Floor |
| ModelLine |
| Opening |
| Wall |
| Group | Group |
| Families | Family |
| Drafting | AnnotationSymbol |
| DetailArc |
| DetailLine |
| DetailNurbSpline |
| Dimension |
| Grid |
| IndependentTag |
| Level |
| ReferencePlane |
| Room |
| RoomTag |
| SpotDimension |
| TextNote |
| Not Exposed | Element |
| Supporting Element | Sketch |
| SketchPlane |

Fingure 10. Non symbol element classes

|  |  |  |
| --- | --- | --- |
| Symbol | System Family Type (end with type) | AnnotationSymbolType |
| BeamSystemType |
| ContFootingType |
| DimensionType |
| FloorType |
| GridType |
| GroupType |
| LevelType |
| RoomTagType |
| SpotDimensionType |
| TextNoteType |
| WallType |
| Settings Type | SiteLocation |
| Component Family type | FamilySymbol |
| Not exposed yet | HostObjAttributes |
| Symbol |

Figure11 Symbol classes

**Elements, Classes and Categories**

As we have discussed earlier, to identify an element among others, we would typically need to check the type of class plus Category property. But how reliable is it? Can we always count on category? If we know there is a Wall class, can we simply look for Wall class? To answer this question, let’s take a look at another experiment.

The next thing we did was that we placed the basic Revit modeling objects and compared their internal class names and its category value and viewed it in a spreadsheet. Figure 12 shows the part of the result. Classes for walls and floor created through Wall/Floor UI menu are HostObject/Wall and Floor. Doors and windows are both FamilyInstance, but category clearly indicated; Doors for door object and Windows for window object. Some objects are not exposed yet; their classes show as Element (in green in the figure). Those are as expected. Now look at category column. One thing you notice is that a wall created as an in-place family (i.e., you create using “Create…” command) is internally an instance of FamilyInstance class and not a Wall class (in blue). (This is shown as a crescent shape wall in the figure 7 earlier.) Category is not defined for some objects, such as opening, section and model group.

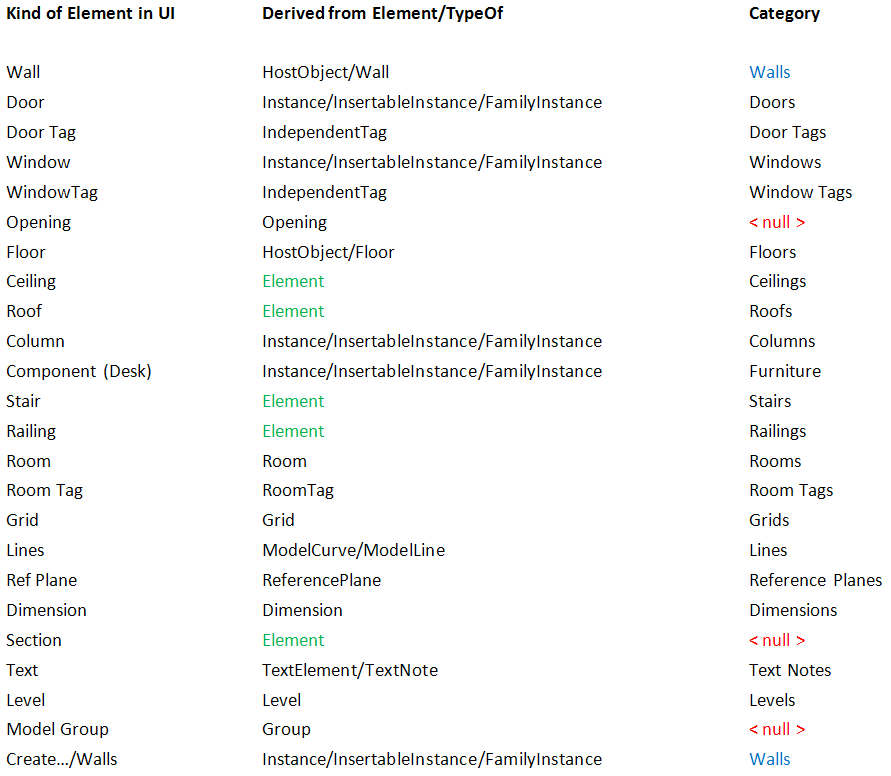


Figure 12 Elements in UI and their classes and categories

Figure13 shows the similar list that actually extends previous one by adding the corresponding symbols on the right. You can see that elements and symbols basically share the same category names. There are some exceptions in grids, text notes, levels, dimensions. There is no corresponding symbol for an opening. You can see correspondence in most of the naming of element/symbol classes: e.g., Wall/WallType, Floor/FloorType, Grip/GridType, TextNote/TextNoteType, and FamilyInstance/FamilySymbol. System family symbol has designated classes while the class of standard or component family symbol is all FamilySymbol.

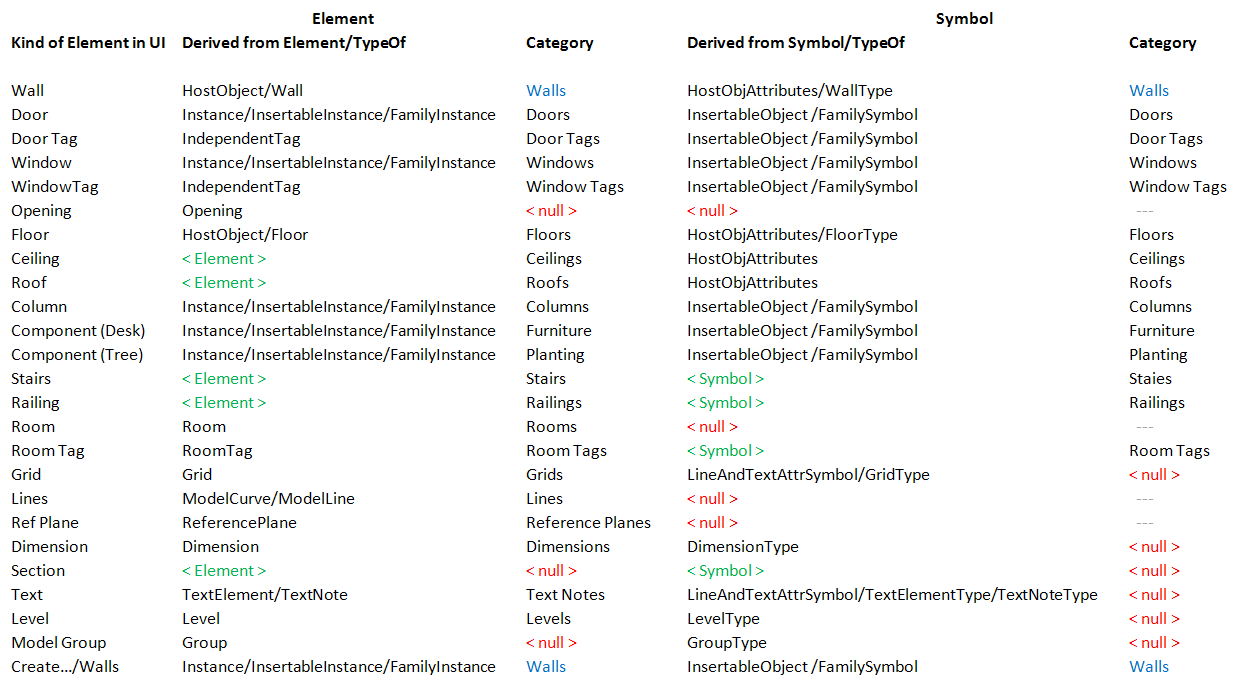


Figure 13 Elements and corresponding symbols in UI and their classes and categories

**Identifying Revit Elements – Revisited**

Identifying Revit elements could be brain twisting. (At least it has been for me for a long time.) Now that we have a little better view of elements in the list, let’s take a look at a way we determine the element once again. An object can be determined using:

* Object Types (i.e., class) or TypeOf
* Category (not always defined).
* Derived from Element or Element/Symbol

We will need to use combination of above. Sometimes additional checks are needed (e.g., Family and Symbols, geometry). Element.Name property has some meaningful string in a specific context, such as component family names, but not always defined (i.e. shows as "???").

If you encounter a new object which you need to handle using Revit API, draw that object in Revit, and then use “Snoop Current Selection” tool to identify the object class and the class for its type. Check the category. If you see the class is Element, unfortunately its property is not fully exposed. (Some generic property such as geometry can be still useful.)

**Snooping Around**…

I have been playing around with this RvtMgdDbg for some time. It is truly a great tool and helps our daily life of supporting Revit API. In the rest of this section, we will point to a few tips about using this tool, which you might find useful later on.

Component Families and Types

Earlier I mentioned that the class called Family is like a container class. A set of family types is grouped under a family. In Revit UI, you can view it under Families in the project browser. Take a “M\_Keynote Tag” (this is in a metric template), for example, there are three types defined in this family. In the Snoop Db tool, you can locate this under Family tree node. If you check the element named “M\_Keynote Tag”, you can find the member types under Symbols property. One thing to note is that the class Family has no category defined. A workaround is to check the category of one of the member FamilySymbol under Symbols property. (Figure 14.)

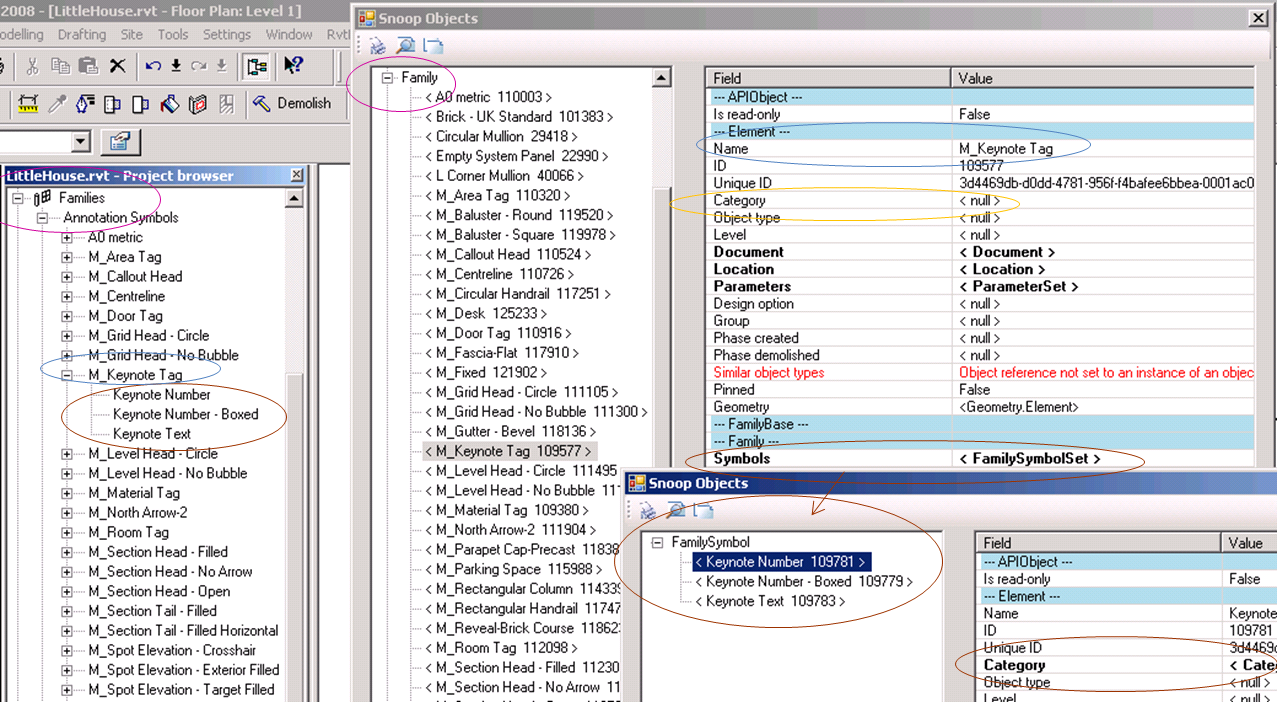
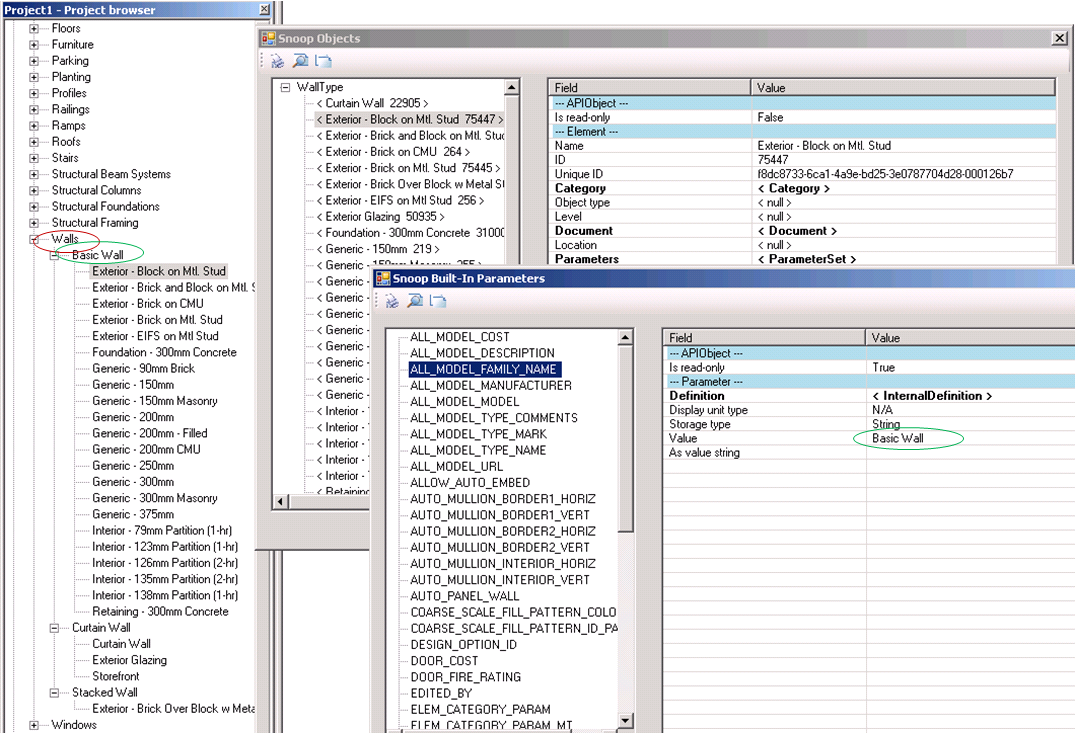
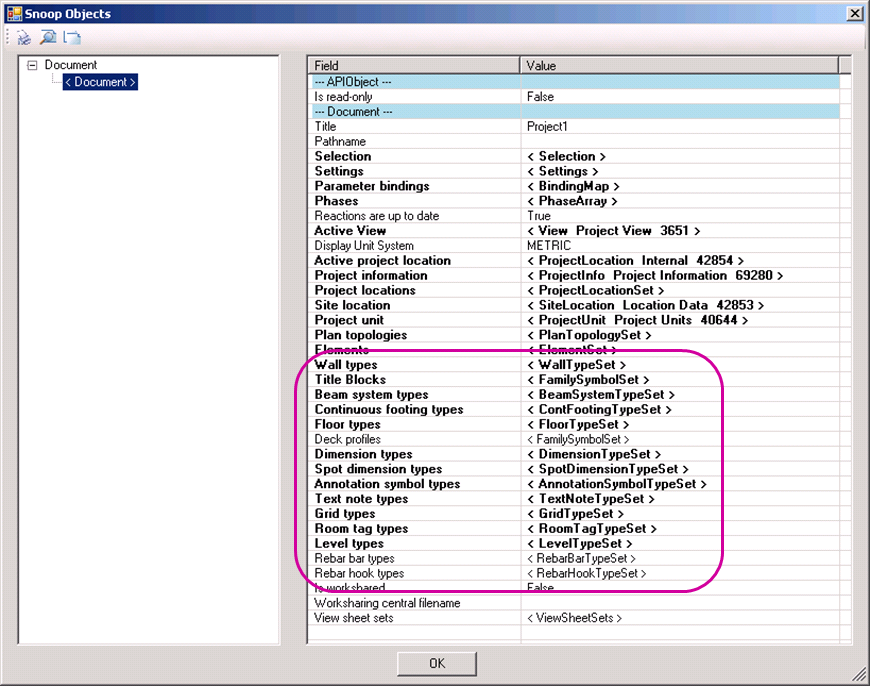


Figure 14 Component family

System Families and Types

Above mentioned Family class are available only for component families (although from the UI, it looks as if they are under Family class). For system family, there is normally designated, corresponding class for its type. As an example, the class for wall symbol is WallType. To find out the family name, you can check the property of a wall type and look up Built-In Parameter “ALL\_MODEL\_FAMILY\_NAME” (Figure 15). (More on Snoop Parameter later.) Additionally, some system family types are also accessible from document object (but not all) (Figure 16).

Figure 15. Wall system families and types

   
Figure 16. Accessing types from document object

Accessing View Specific Instances

One way to access instances is through View.Elements. This gives only the instances specific to a given view, however. Select a view from project browser, snoop current selection, then click Elements property (Figure 17).

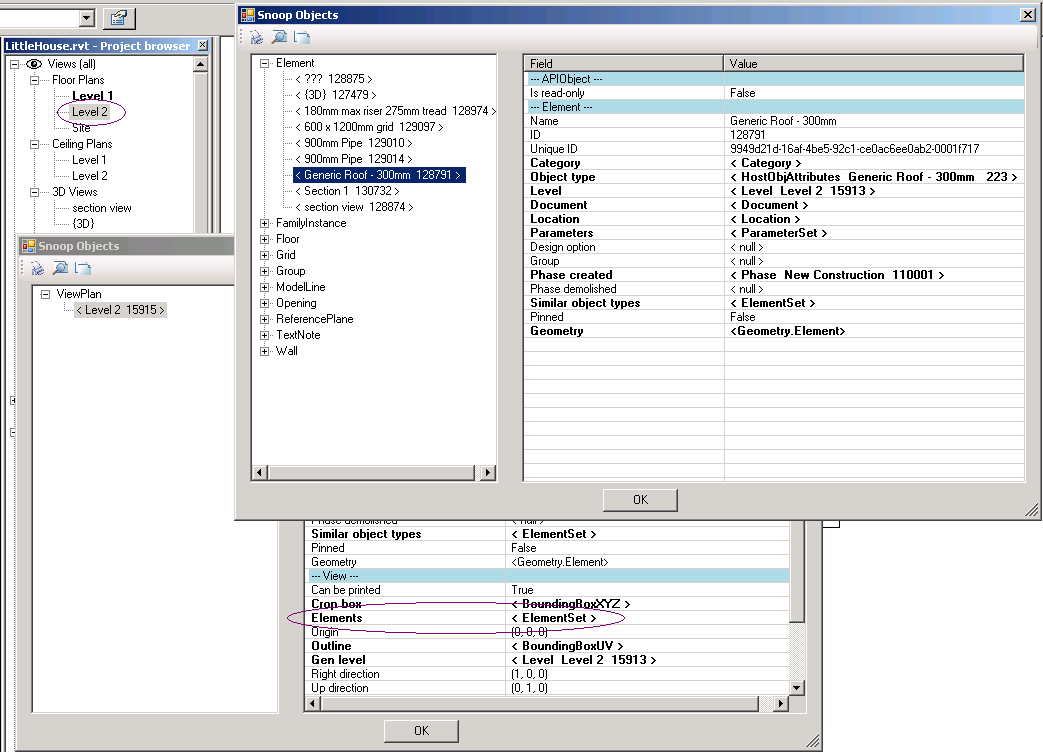


Figure 17. Accessing view-specific instances

Parameters

One last tool I would like to mention is a snoop tool for parameters. Each element has Parameters property, which closely corresponds to the properties values you see in the Element Properties dialog in UI. The parameters you can view here may be built-in or shared parameters, including both visible and invisible ones.

To access a specific parameter, you can use the Parameter method by passing in a built-in parameter enum value (note singular form here as opposed to Parameters in plural form previously). Often, the difficulty in using built-in parameters is that there are more than 2000 built-in parameters in Revit, while only a small portion of these apply to a specific element. Unfortunately, there is no documentation, either. (The documentation of built-in parameters is one of most frequent requests, by the way. There is no official document for that to date.) Nevertheless, it has been our experience that occasionally you may find additional information; a family name for a specific wall family type is an example of such case.

In the SnoopObjects dialog, click on Parameters property fields. The Snoop Parameters dialog includes a button labeled “Built-In Enum Snoop”. From there, you can view the list of “available” built-in parameters (Figure 18). Please note that “available” means it is not *nothing/null*. If does not guarantee that it has meaningful value. Built-in parameter is not an officially supported portion of API. In future we expect it will be replaced by data being properly exposed as a property.

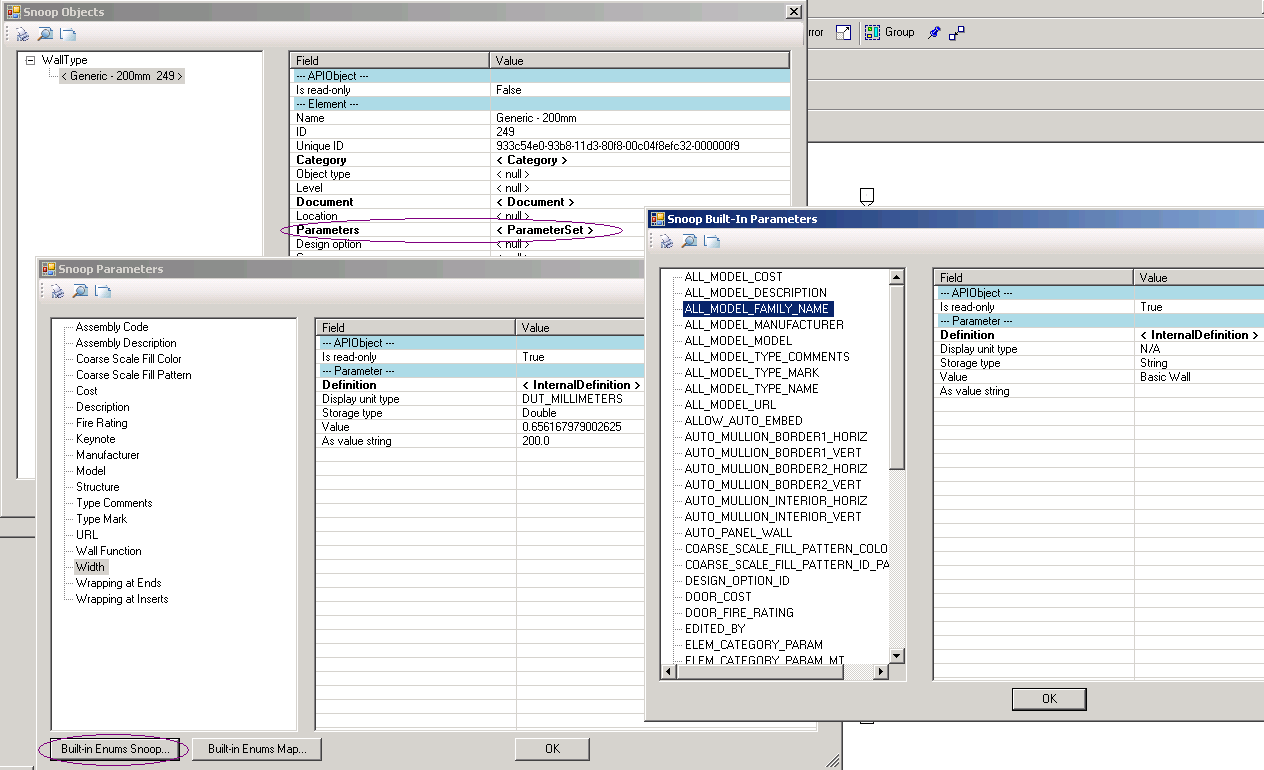


Figure 18. Built-in parameters snoop tool

**Summary**

In Revit, all the elements are bundled together as a single list of elements under Document.Elements and are accessible through Document’s Element Iterator. The element list contains thousands of elements in an unstructured manner. For Revit programmers, a task often becomes to find out a way to identify the object that you are looking for. In this session, we have taken a closer look at the elements list in Revit drawing database. Starting with “raw” data, we have analyzed it and learned how much is exposed currently. We have attempted to group together among those exposed according to the grouping in UI. We introduced a tool called RvtMgdDbg and showed you how to explore the internals using this tool. Through these exercises, I hope you now have better view of Revit API than before.

**Acknowledgement**

Special thanks to Jim Awe and Arjun Ayyar of Advance Development Group of Autodesk. RvtMgdDbg was developed by Jim Awe and his team originally intended for internal use. They have been continuously improving the tool, and making it available for the developer community.