REVIT RTC TECHNOLOGY CONFERENCE AUSTRALASIA 2014

Session 35 & 36 Masterclass

Learn to write Addons no coding experience required

Adam Sheather, YTL Corporation

Class Description

This course is for Expert users of Revit who wish to learn how to code. This will cut a lot of not essential information from standard and other coding classes to get users quickly up to speed to write basic add-ons and give them tools to discover and develop independently.

About the Speaker:

I enable teams to bridge the gap between software tools and design construction and operation lifecycle processes.
It's taking a while.

Basic C# Principals

OOP Tips

- Classes are like Revit Families, templates creating types and instances.
- Objects are like actual instances of families in a project.
- Data fields are like Parameters each is made of a DataType like integer or string, custom ones made by programmers are called Object Types.
- Methods are the modification actions like changing objects position or height.
- Constructors are the "Creation Methods" for creating objects, like wall by line, picked line or face.

You all should have received a cheat sheet for actual C# coding. This one is the best I have found and is free to distribute so big thanks to the THECODINGGUYS 2013.

The rest of the guide will focus on Revit API specific commands in combination with the C# commands.

The following is a few of the key things I believe that handout missed but you will need to know.

References

References are all the libraries your code needs in order to work. The most specific example is making sure you have access to RevitAPI.dll and RevitUIAPI.dll. These contain all the classes, methods and data that we use to talk to Revit and our projects.

Namespaces

Namespaces are a logical way of organising our projects. Namespaces separate classes, methods and fields from one namespace to another.

Classes

Classes are our templates, this is where we describe what data our object contains and what it can do with that data.

Scope

The Code blocks in all C# are the current scope of fields, methods and options a command can access.
This means if you cannot access local code in your current code in outer code areas.

```
MyCode()
{
int number = 1;

        if (number == 1)
        {
                int result = number + 3;
        }
        Result.ToString(); - CANNOT ACCESS RESULT LIKE THIS
}

CORRECT METHOD

MyCode()
{
int result;
int number = 1;

        if (number == 1)
        {
                result = number + 3;
        }
        result.ToString(); IT CAN ACCESS THE CODE
}
```

Public/Private

Public DataTypes are private by default, if you make them public it means other methods in class will be able to access them.

Accessing methods in an object

Accessing data from methods can be a little bit harder.
Some methods require arguments which is data so the method can return a result, the result might also be in a particular format.
Example;
int result = convert.ToInt32("25");
This is a conversion method, it returns an integer(int) in order to work it requires a string argument(the "25"), in order to work.

There are also methods which do not require arguments.
Example;
int number = 1;
string string = number.ToString();
In this example another conversion is taking place, but because the variable number is 1, a simple ToString() method can work directly on the variable value.

Visual Studio Tips

- F12 is your friend, it will check show you methods, fields and options for any class in VS without looking at SDK
- Error List is a great way to check what errors you have in your syntax to allow you to debug your application.
- Breakpoints are your best friends in the debugging world so you them!!!
- Local which appears when you are debugging is CRITICAL, it shows you all the current variables, classes, collections and other items you have accessed.
- Whenever your coding and still trying things, just comment out your old code and new code instead of copy/paste or delete
- ALWAYS COMMENT YOUR COMPLETED CODE you'll be surprised how much it helps you if you need to update it, let alone others.

Revit Cheat Sheet

Do's and Don'ts

- Before you do anything google what you trying to do, then check out these Datasets, Building Coder, Spiderinnet and Harry Mathison's blog, then go through the Revit samples then if you still can't find anything go through the SDK, chances are someone has done something like it or portions.
- Dynamo is like coding for newbies, if you know dynamo it's a small jump to the API, and the reverse is even easier.
- I always find it best to make simple mockups in VSTA then bring the code into Visual Studio class, if it doesn't need a UI then leave it as a macro.
- Whenever you change the Revit Database you must do it inside a **Transaction**
- You may need to pass the active document as a parameter to custom classes in order to manipulate objects
- I can swear that practice makes perfect, you can read all you want but creating code and debugging is the only to become comfortable with a lot of logical and abstract ideas.
- Don't get discouraged, sometimes 2-3 lines of code can take 8 hours of research patience and an ability to not destroy office equipment is a key skill to have.
- Like BIM and Revit, coding is big world outside (much bigger then BIM), so keep trying to improve and learning new features and skills not only on Revit but the dotnet API, and the other languages beyond, you'll be surprised how easy it is an things you can do!!

Active Application and Document

The application is the current open Revit application, and the Document is a project or family file open in the application. The project or family you have open at the time you run an API is called the Active Document.

To call up the active document use the following;

```
Document doc = ExternalCommand.Application.ActiveUIDocument.Document;
```

This accesses the Revit Database and objects inside. To access the user interface commands you need to use the UIDocument.

```
UIDocument uidoc = ExternalCommand.Application.ActiveUIDocument;
```

I suggest you keep the naming similar so you can quickly copy/paste code between VSTA and Visual Studio.

Get Elements Family instances, symbols and System objects.

Easiest way to get an element is by its element id.

```
Element e = GetElement(eid);
```

To get a Wall element as a family instance just use the *as* keywords.

```
FamilyInstance fi = GetElement(eid) as FamilyInstance;
```

The same thing works for a wall.

```
Wall w = GetElement(ied) as Wall;
```

To get the type

Filtered Collections

To get elements from your project file your best to create a collection. Below is a breakdown of a Collection of Wall Elements.

```
ICollection<Element> elem = new FilteredElementCollector(doc)
```

Creates a new empty collection of elements from the active project.

```
.OfCategory(BuiltInCategory.OST_Walls)
```

Uses categories method to get access to the Walls

```
.WhereElementIsElementType()
```

Uses the Type method to check if the elements are types and filters out instances

```
.ToElements();
```

Stores the items as element objects in the collection.

LINQ and LAMBDA Revit Queries

```
var doorColl = new FilteredElementCollector(doc)
```
Creates a new empty collection of elements from the active project. Notice the use of var (it stands for variable and instead of writing int, string or others you can just write var), be very careful with this option it can make writing code easier but for programmers it can make it harder to read your intentions and catch errors.
```
.OfCategory(BuiltInCategory.OST_Doors)
```
Uses categories method to get access to the Doors
```
.OfClass(typeof(FamilyInstance));
```
Gets the FamilyInstances in your Collection and stores them as family instance objects.

Now we can use a LINQ query to further refine our collection going through values, or other options rather than using foreach, this is preferred for me if you are searching for items not manipulating them. This is just an easy readable way of using filters.
```
//Uses linq queries to select the door that have a ToRoom Value
IEnumerable<FamilyInstance> doors =
```
Creates a new empty Enumerable that can contain FamilyInstances for now just assume IEnumerables are another option like collections.
```
from FamilyInstance f in doorColl
```
Gets the FamilyInstances from your door's collection up above
```
where f.ToRoom != null
```
Check if the doors have a ToRoom parameter that does not equal null
```
select f;
```
Selects the doors if that is the case in your new IEnumerable

Now from here we can use Lambda which is a type of fancy math that allows you write functions without naming them.

```
ElementId[] doorIds = doors.Select(door => door.Id).ToArray();
```
Creates an ElementId array, then has an equation to access the door objects then get the Id's from the door and saves them to the array. See that I used the variable door, but did not declare or initialise it anywhere the same with the rest of the function as it's not a method.

Transactions

Transactions are a key part of Revit, you can read any Revit data you want, but as soon as you want to change something in a Document via API even if it's a parameter it needs to be in a transaction.
The safest way to use a transaction is by the "using" keyword.

```
//Start the transaction CRITICAL without transactions Revit cannot update
using (Transaction t = new Transaction(doc, "Door Data Update"))
{
//Starts the transaction
t.Start();

//Your code does something here

//Commits the changes to the Revit File
t.Commit();
}
```

Links

Building Coder
Building Coder Samples
RevitNETAddinWizard
Boost Your BIM
https://github.com/DynamoDS/Dynamo