MACRO VS EXTERNAL COMMAND

MACRO MANAGER

Application:

Macro modules available to all opened Revit projects in the current instance of the Revit application.

Active document tab:

The active document tab represents the currently active project in Revit. The project does not necessarily contain embedded macros.

SHARP DEVELOP

Create a module and then as many macro as you need in that module.

HELLO WORLD!

```
public void MyFirstMacro()
    {
    TaskDialog.Show("Dialog Title", "My first Macro!");
    }
```

PUBLIC

Available to all callers with access to the type

VOID

The method does not return anything. For example:

```
void Ok_btnClick(object sender, EventArgs e)
{
    usertext = textBox1.Text;
}
```

This method sets the value of a variable.

RETURN

This method selects all the View template in the project and return them as a list.

```
public static List<View> collectTemplates(Document doc)
{
    IEnumerable<View> fec = new FilteredElementCollector(doc)
                    .OfClass(typeof(View))
                    .Cast<View>();

    List<View> myVT = new List<View>();
    foreach (View v in fec)
    {
        if (v.IsTemplate) {
            myVT.Add(v);   }
    }
return myVT;
}
```

STATIC

No instance is required to be invoked.

```
List<View> viewTemplates = collectTemplates(doc);
```

An instance can be created using the *new* keyword:

```
FilteredElementCollector viewTypes = new FilteredElementCollector(doc)
          .OfClass(typeof(ViewFamilyType));
```

WHY I NEED TO CREATE AN INSTANCE OF SOME CLASSES?

PYTHON TO C#

1. When you declare a variable or constant, you must either specify its type or use the *var* keyword
2. You must end each statement with a semicolon;
3. Double quotes encode a string of multiple characters, single quotes encode a single character (data type *char*)

NAMESPACE

Python

```python
import clr
clr.AddReference('RevitAPI')
clr.AddReference('RevitAPIUI')
from Autodesk.Revit.DB import *
from Autodesk.Revit.UI import *
```

C#

```csharp
using System;
using Autodesk.Revit.UI;
using Autodesk.Revit.DB;
using Autodesk.Revit.UI.Selection;
using System.Collections.Generic;
using System.Linq;
```

DOCUMENT MANAGER

Python

```
doc = DocumentManager.Instance.CurrentDBDocument
uidoc = DocumentManager.Instance.CurrentUIApplication.ActiveUIDocument
```

C#

```
//Access the UI of the currently Revit project opened
UIDocument uidoc = this.ActiveUIDocument;
//The active or top most view of the project
Document doc = uidoc.Document;
```

SELECTION

Python

```python
viewTypes = list(FilteredElementCollector(doc).OfClass(ViewFamilyType))
```

C#

```csharp
FilteredElementCollector viewTypes = new FilteredElementCollector(doc)
    .OfClass(typeof(ViewFamilyType));
```

FILTER

Python

```python
for vt in viewTypes:
    if str(vt.ViewFamily) == 'Drafting':
    viewType = vt
    break
```

C#

```csharp
ViewFamilyType vft = null;
foreach (ViewFamilyType vt in viewTypes) {
    if (vt.FamilyName == "Drafting View"){
    vft = vt;
    }
}
```

TRANSACTION

Python

```
t = Transaction (doc, 'Make new Drafting view')
t.Start()
t.Commit()
```

C#

```
using (Transaction t = new Transaction(doc))
{
t.Start("Make new Drafting view");
t.Commit();
}
```

CALLING A METHOD

Python

```
newDraftingView = ViewDrafting.Create(doc, viewType.Id)
newDraftingView.Name = textBox.Text
```

C#

```
ViewDrafting newDraftingView=ViewDrafting.Create(doc,vft.Id);
newDraftingView.Name = "My New Drafting View";
```

CODE STRUCTURE

1. Store your methods in a separate Class (i.e. Helpers)
2. These methods must be *public static*
3. Add a Form to the project
4. Create an instance of the Form in ThisApplication
5. Call your methods from ThisApplication (i.e. Helpers.MethodName)

HELPERS

```
public static List<View> collectTemplates(Document doc){
    IEnumerable<View> fec = new FilteredElementCollector(doc)
                .OfClass(typeof(View))
                .Cast<View>();
    List<View> myVT = new List<View>();
    foreach (View v in fec)
    {
        if (v.IsTemplate){
        myVT.Add(v);
        }
    }
    return myVT;
}
```

```
public static
```

```
IEnumerable<View>
```

FORM

```
public partial class Form2 : frms.Form {
    public int chosenView;
    public Form2(Document doc) {
        InitializeComponent();
        List<View> viewTemplates = Helpers.collectTemplates(doc);
        foreach (var v in viewTemplates) {
        comboBoxDrop.Items.Add(v.Name);
        }
    }
    void Form2Load(object sender, EventArgs e){ }
    void ComboBox1SelectedIndexChanged(object sender, EventArgs e){
        chosenView = comboBoxDrop.SelectedIndex;}
}
```

AVOID NAMESPACE CONFLICTS

```
using winForm = System.Windows.Forms;
```

COMBOBOX SELECTED INDEX CHANGED EVENT

```
void ComboBox1SelectedIndexChanged(object sender, EventArgs e)
{
    chosenViewTemplate = comboBox1.SelectedIndex;
}
```

ADD THE DOCUMENT AS AN ARGUMENT OF THE FORM

```
public CreateDraftingViewForm(Document doc)
```

THIS APPLICATION

```
public void PopulateDropDown()
{
    UIDocument uidoc = this.ActiveUIDocument;
    Document doc = uidoc.Document;
    List<View> allViewTemplates = Helpers

.collectTemplates(doc);
    using(var forma = new Form2(doc)){
    //use ShowDialog to show the form as a modal dialog box.
    forma.ShowDialog();
    TaskDialog.Show("result",
                allViewTemplates[forma.chosenView].Name);
}
```

```
TaskDialog.Show("ViewTemplateSelected", form.chosenViewTemplate);
```

USE WHILE TO KEEP THE DIALOG BOX OPEN

```
string interrupt = "False";
while(interrupt == "False") {
form.ShowDialog();
if (form.usertext.Length >2) {
        Helpers.AddDraftingView(doc, form.usertext, form.chosenTemplateId);
        interrupt = "True";
    }
else if (form.usertext == "") {
        TaskDialog.Show("Error", "Please specify the view name"); }
else if (form.usertext.Length <2) {
        TaskDialog.Show("Error", "The view name is too short");   }
else { TaskDialog.Show("Error", "I don't know what went wrong");  }
}
```