Autodesk University 2012

View and Schedule API

Steven Mycynek - Autodesk
(Assistant/Co-presenter optional) [Arial 10]

CP3133 – Using the Autodesk Revit Schedule and View APIs

Learning Objectives

At the end of this class, you will be able to:

- Create a variety of view types using the API
- Understand the data that makes up a View
- Perform UI view customizations
- Work with schedules via the API
- Use a few general API tips

About the Speaker

Steven Mycynek is a principal engineer on the Autodesk Revit team and has been with Autodesk since 2010. Prior to this, he worked for eight years at Dassault Systems SolidWorks corporation in the API support and development group.

Steve holds a B.S. in computer science from the University of Illinois at Urbana, IL.

Create a variety of View Types/Understanding the data in a View

Calculating a direction "up" vector

While (0,0,1) will be the global "up" vector in Revit most of the time, when specifying a 3D view, you will need to specify an Up-vector relative to your direction. That is, you will need a vector that is both normal to your vector and is oriented in a way that is intuitively "up" for that point of view. How do we do this?

English

1. Decide what direction is globally "up" (In Revit, this is usually (0,0,1), the opposite direction of a plumb bob).
2. Calculate your direction vector by subtracting your eye point from your target point.
3. GroupGet a local "right" vector normal to your direction and global up via a cross product
4. Get a local "up" vector normal to that right vector via another cross product.

PseudoCode

1. globalUp = (0,0,1) (or XYZ.BasisZ)
2. direction = targetPoint – eyePoint
3. up = direction.Cross(globalUp).Cross(direction)

C# Code:

```
public static void CalculateDirectionAndUp
(XYZ eyePoint, XYZ targetPoint, out XYZ direction, out XYZ up)
{
 direction =  targetPoint.Subtract(eyePoint);
 up =  direction.CrossProduct(NamedLocations.GlobalUp).CrossProduct(direction);
}
```

Turning a view over and upside down

If you have a 2d ortho-plan view or detail view, but it is backwards and/or upside down (e.g, you have a reflected ceiling plan when you want a floor plan), perform 180 degree (Pi radians) rotations to adjust.

```
detailSectionBox.Transform
//rotate about Z axis to turn upside down.
 = (Transform.get_Rotation(new XYZ(0, 0, 0), new XYZ(0, 1, 0), System.Math.PI) *
//rotate about y axis to flip over.
Transform.get_Rotation(new XYZ(0, 0, 0), new XYZ(0, 0, 1), System.Math.PI)) ;
```


Comments and code have been trimmed for print readability – please see the source code download for complete documentation.

Section View Transform

Creating a bounding box for a section view requires a few vector operations. One way to start is by finding the midpoint of a wall (or other curve-based-object) and getting the direction vectors at that point using ComputeDerivatives().

```
LocationCurve wallLocation = wallToCut.Location as LocationCurve;  //Get the line that the wall follows.

//Center location, normalized midpoint.
Transform wallLocationTransform = wallLocation.Curve.ComputeDerivatives(0.5, true);

//Get vectors that show direction of wall and its normals.
 XYZ wallOrigin = wallLocationTransform.Origin;

//Tangent vector -- section view will be mirrored away from this vector.
XYZ wallTangentDirection = wallLocationTransform.BasisX.Normalize();

//Normal to Tangent
XYZ wallNormal = wallTangentDirection.CrossProduct(XYZ.BasisZ).Normalize();
Transform wallSectionTransform = Transform.Identity;

wallSectionTransform.Origin = wallOrigin;
wallSectionTransform.BasisX = wallNormal; //In this case, "pointing out from the wall" is "X/Right."
wallSectionTransform.BasisY = XYZ.BasisZ;  //In this case, "Y" is "Up." --- only works for Vertical walls.

//In this case, "Z" is depth.
wallSectionTransform.BasisZ  = wallNormal.CrossProduct(XYZ.BasisZ);


BoundingBoxXYZ wallSectionBox = new BoundingBoxXYZ();  //View Into your section
// +Z = depth beyond section plane, -Z = depth behind section plane

//Set up a bounding box aligned with the wall midpoint.
wallSectionBox.Transform = wallSectionTransform;
wallSectionBox.Min = clipNear;
wallSectionBox.Max = clipFar;
```


Comments and code have been trimmed for print readability – please see the source code download for complete documentation.

UI View Customization

The new UIVew class in Revit 2013 has a convenient "ZoomAndCenterRectangle" method to easily take bounding values and temporarily set a view to those bounds.

```
/// <summary>
/// Sets all open views to a zoom-to-fit view of a given element.
/// </summary>
private void ZoomElementToViews(Element element)
    {
     if (element == null)
         throw new ArgumentNullException("element");

    foreach (UIView uiView in ThisApplication.DocumentUtility.UiDoc().GetOpenUIViews())
      {
        View dbView = ThisApplication.DocumentUtility.Doc().GetElement(uiView.ViewId) as View;
        if (dbView.ViewType != ViewType.Schedule)
        {
          BoundingBoxXYZ bbox = element.get_BoundingBox(dbView);
          uiView.ZoomAndCenterRectangle(bbox.Min, bbox.Max);
          ThisApplication.DocumentUtility.UiDoc().ActiveView = dbView;
          ThisApplication.DocumentUtility.UiDoc().RefreshActiveView();
        }
      }
    }
```


Comments and code have been trimmed for print readability – please see the source code download for complete documentation.

Working with View Schedules

View schedules need fields, and optionally, they can contain sorting and filtering rules. See below how to take text representations of "ALL_MODEL_MARK" and "FAMILY_WIDTH_PARAM" and convert them to BuiltInParameter Enums and then into fields to be scheduled.

```
string field1 = "ALL_MODEL_MARK"
string field4 = "FAMILY_WIDTH_PARAM"
ScheduleSortOrder.Ascending = true;
int greaterThanValue = 3;

bool field1ParseSuccess = Enum.TryParse<BuiltInParameter>(field1, out field1Param); //ALL_MODEL_MARK
bool field4ParseSuccess = Enum.TryParse<BuiltInParameter>(field4, out field4Param); //FAMILY_WIDTH_PARAM

ScheduleField sf1 = vs.Definition.AddField(new SchedulableField
(ScheduleFieldType.Instance, new ElementId(field1Param)));

ScheduleField sf4 = vs.Definition.AddField(new SchedulableField
(ScheduleFieldType.ElementType, new ElementId(field4Param)));

//Sort field 1 in either ascending or decending order, depending on input, by adding //"ScheduleSortGroupField" to the
ViewSchedule.

//In this case, we will be sorting by ALL_MODEL_MARK (Door Tag).
ScheduleSortGroupField markSortField= new
ScheduleSortGroupField(sf1.FieldId, isAscending ? ScheduleSortOrder.Ascending : ScheduleSortOrder.Descending);

markSortField.ShowBlankLine = true;
vs.Definition.AddSortGroupField(markSortField);

//Filter items in Field4 greater than "greaterThanValue"
vs.Definition.AddFilter(
new ScheduleFilter(sf4.FieldId, ScheduleFilterType.GreaterThan , greaterThanValue));
```


Comments and code have been trimmed for print readability – please see the source code download for complete documentation.

General API Tips

Getting a selected element and verifying it of the type expected can take a bit more code than you may expect. These methods below help manage getting a selected object of a given type and prompting the user to select one if one was not already selected. Note the use of generics, default arguments, and PickObject().

Selection Utility

```
public RevitElement GetSelectedElementOrPromptForSelection<RevitElement>
(bool promptForSelection = true)  where RevitElement : Element
{
        return GetSelectedElementOrPromptForSelection<RevitElement>
("Select an element of type: " + typeof(RevitElement).Name, promptForSelection);
}

private RevitElement GetSelectedElementOrPromptForSelection<RevitElement>
(string promptText, bool promptForSelection)  where RevitElement : Element
{
 RevitElement retval = null;
  if ((!(UiDoc().Selection== null)) && (!(UiDoc().Selection.GetElementIds() == null))
 && (!(UiDoc().Selection.GetElementIds().Count ==0)))//If something was pre-selected
   {
    //Try to get the first selected object.
     ElementId first = UiDoc().Selection.GetElementIds().First();
  if (first == null)
    retval = null;
   else
      retval= Doc().GetElement(UiDoc().Selection.GetElementIds().FirstOrDefault()) as RevitElement;
     }
 //Return the pre-selected object if it exists.
  if ((retval == null) && promptForSelection)
    retval = PickRevitElement<RevitElement>(promptText);
 return retval;
 }
```


Comments and code have been trimmed for print readability – please see the source code download for complete documentation.

Selection Utility (Continued)

```
private RevitElement PickRevitElement<RevitElement> (string prompt) where RevitElement : Element
    {
      ISelectionFilter filter = new ElementSelectionFilter<RevitElement>();
      Reference selectedReference = null;
      try
      {
        selectedReference = UiDoc().Selection.PickObject(ObjectType.Element, filter, prompt);
        return Doc().GetElement(selectedReference) as RevitElement;
      }
      catch (Exception)
      {
        return null;
      }
    }
```

Use this selection filter with the selection methods above

```
/// <summary>
/// A simple selection filter that can allow any element type.
/// </summary>
public class ElementSelectionFilter<RevitElementType> : ISelectionFilter
{
        //Tip
        public bool AllowElement(Element element)
        {
                return (element is RevitElementType);
        }

 public bool AllowReference(Reference refer, XYZ point)
        {
return false;
        }

}
```


Comments and code have been trimmed for print readability – please see the source code download for complete documentation.

Name generation

Many Revit APIs will generate a unique name for a new element for you if you do not specify one. However, if you do specify a name, it must be unqiue? What if you want some control over new names but can't interrupt the user to ask for a new name? Try something like this.

```
public bool GenerateName<RevitElement>
    (string originalName, out string newName) where RevitElement: Element
{
        //Tip - SuffixBuilder
        newName = null;
        if (!DoesElementExist<RevitElement>(originalName))
        {
                newName = originalName;
                return true;
        }
        int nameAttempts = 10;
        for (int index = 0; index != nameAttempts; index++)
        {
                string attemptedName = originalName + "_" + SuffixBuilder.Next();
                if (!DoesElementExist<RevitElement>(attemptedName))
                {
                        newName = attemptedName;
                        SuffixBuilder.Reset();
                        return true;
                }
        }
        SuffixBuilder.Reset();
        return false;
}
```


Comments and code have been trimmed for print readability – please see the source code download for complete documentation.

Name generation (continued)

Here's the code that helps build a unique name for your new element.

```
public class SuffixBuilder
  {
    public static void Reset()
    {
      sm_letterIndex = 0;
      sm_numberIndex = 1;
    }
    //Tip - SuffixBuilder
    public static string Next()
    {
      if (sm_letterIndex == 4)
      {
        sm_letterIndex = 0;
        sm_numberIndex++;
      }
      string suffix = sm_letters[sm_letterIndex].ToString() + sm_numberIndex.ToString();
      sm_letterIndex++;
      return suffix;
    }

    #region Data
    private static char[] sm_letters = { 'a', 'b', 'c', 'd' };
    private static int sm_letterIndex = 0;
    private static int sm_numberIndex = 1;
    #endregion

  }
```


Comments and code have been trimmed for print readability – please see the source code download for complete documentation.

Name Generation and element querying

It's helpful to have a few tools available to see if an element exists in your document.

```
private bool DoesElementExist<RevitElement>(string name) where RevitElement: Element
        {
                if (string.IsNullOrEmpty(name))
                        return false;

                FilteredElementCollector fec = new FilteredElementCollector(Doc());
                fec.OfClass(typeof(RevitElement));
                IList<Element> elements = fec.ToElements();
                Element result = null;
                result = GetElementByName(elements, name);
                if (result == null)
                        return false;
                else
                        return true;

        }
```


Comments and code have been trimmed for print readability – please see the source code download for complete documentation.

Name Generation and element querying (continued)

Getting an element by name is also a very convenient utility worth having around.

```
public RevitElement GetElementByName<RevitElement>(string name) where RevitElement: Element
        {
                                if (string.IsNullOrEmpty(name))
                                        return null;
                                FilteredElementCollector fec = new FilteredElementCollector(Doc());
                                fec.OfClass(typeof(RevitElement));
                                IList<Element> elements = fec.ToElements();
                                Element result = null;
                                result = GetElementByName(elements, name);
                                return result as RevitElement;
        }

private Element GetElementByName(IList<Element> elements, string name)
        {
                Element result = null;
                try
                {
                        result = elements.First(element => element.Name == name);
                }
                catch(Exception)
                {
                        return null;
                }
                return result;
        }
```