View and Schedule API

Steven Mycynek – with notes by Harry Mattison
Principal Engineer

Autodesk University

Class Summary

- What’s changed and new
- View Creation
- Schedule Views
- Sheet Views
- Other enhancements

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

What’s change and what’s new

What’s Changed and New…

- In Revit 2012…
  - ItemFactoryBase.NewView3D -- no way to choose between iso and perspective
  - Document.NewViewDrafting
  - ItemFactoryBase.NewViewPlan
  - ItemFactoryBase.NewViewSection
  - Document.NewViewSheet
  - No creation for Schedule views
  - Not many methods or properties in View subclasses

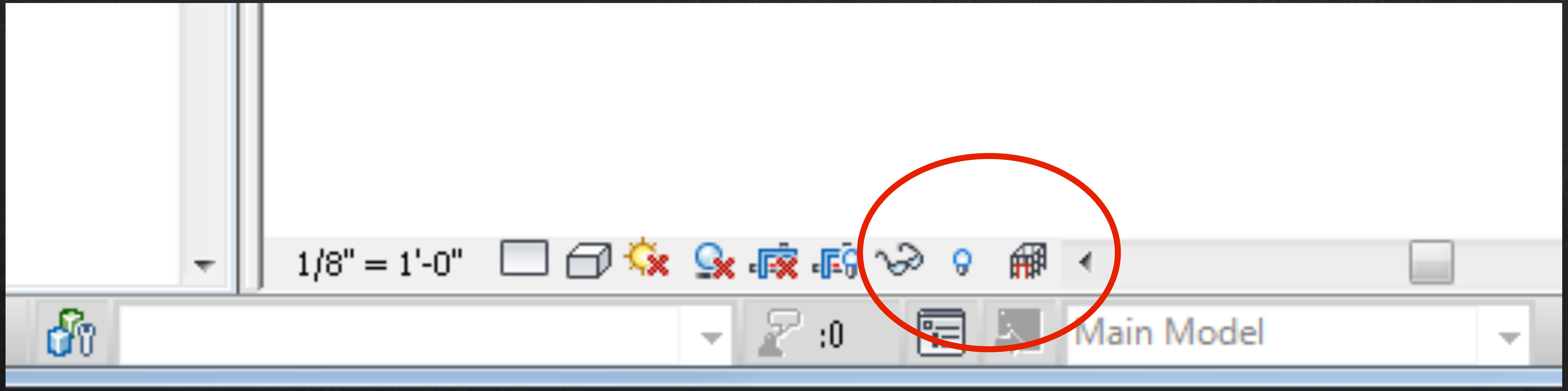
In Revit 2013…

- Creation methods move to static factory methods on View and View subclasses.
- Much more property data on View subclasses
- Support for Schedule Views and Perspective views
- Support for UI view control
- Support for Assembly Views

More of what's new

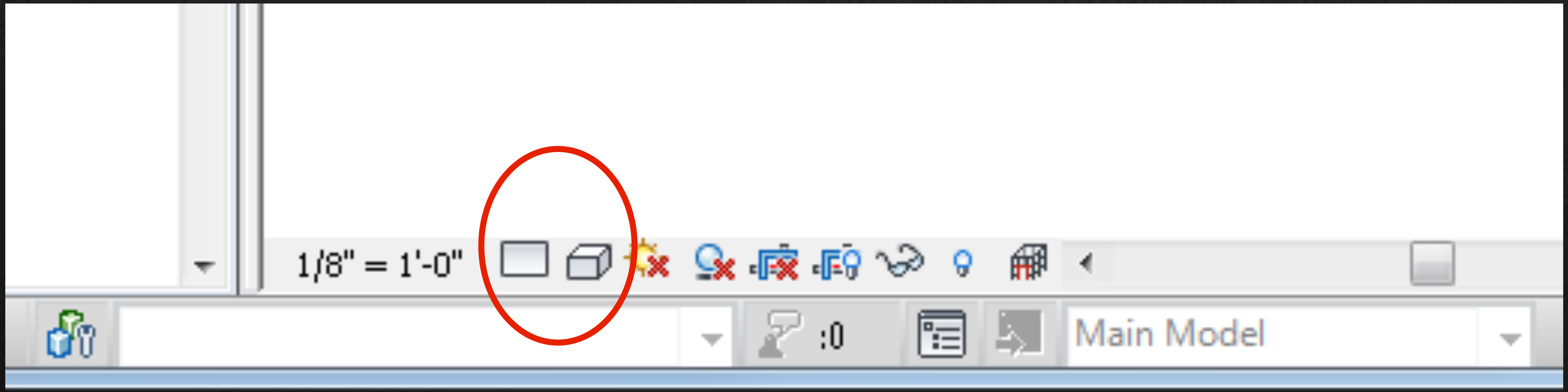
Temporary view modes

- View.EnableRevealHiddenMode()
- View.EnableTemporaryAnalyticalDisplayMode()
- View.DisableTemporaryViewMode(TemporaryViewMode mode)
- View.IsTemporaryViewPropertiesModeEnabled()



More of what's new

- View.DisplayStyle and DisplayStyle enum (get/set Wireframe, HLR, Shading)
- View.DetailLevel and ViewDetailLevel enum (get/set Coarse, Medium, Fine)
- ViewRange can be manipulated via ViewPlan.Get/ SetViewRange()
- 3D View Locking
  - View3D.SaveOrientationAndLock, View3D.RestoreOrientationAndLock, and View3D.Unlock
  - View3D.IsLocked indicates if a view is currently locked
  - View3D.NewTag can be used in locked 3D Views
- View.Duplicate
- StartingViewSettings (StartingViewSettings.GetStartingViewSettings(Document doc)
  - The view that will be open when the model is opened



More of what's new

UIView class

- Represents view windows in the Revit user interface
- UIView.GetOpenUIViews
  - A list of all open views
- UIView.GetWindowRectangle
  - A rectangle that describes the size and placement of the UIView window
- UIView.ZoomAndCenterRectangle(XYZ viewCorner1, XYZ viewCorner2)
  - Ability to zoom and pan the active view
- UIView.GetZoomCorners
  - Two points that define the corners of the view's rectangle

More of what’s new

- ViewFamilyType.PlanViewDirection
  Get/set the view direction to Up or Down for StructuralPlan views
- Schedule export
- Running add-in commands when a schedule view is active

A few API tips

Tips

How comfortable is everyone with?

- C#
- The SharpDevelop environment?

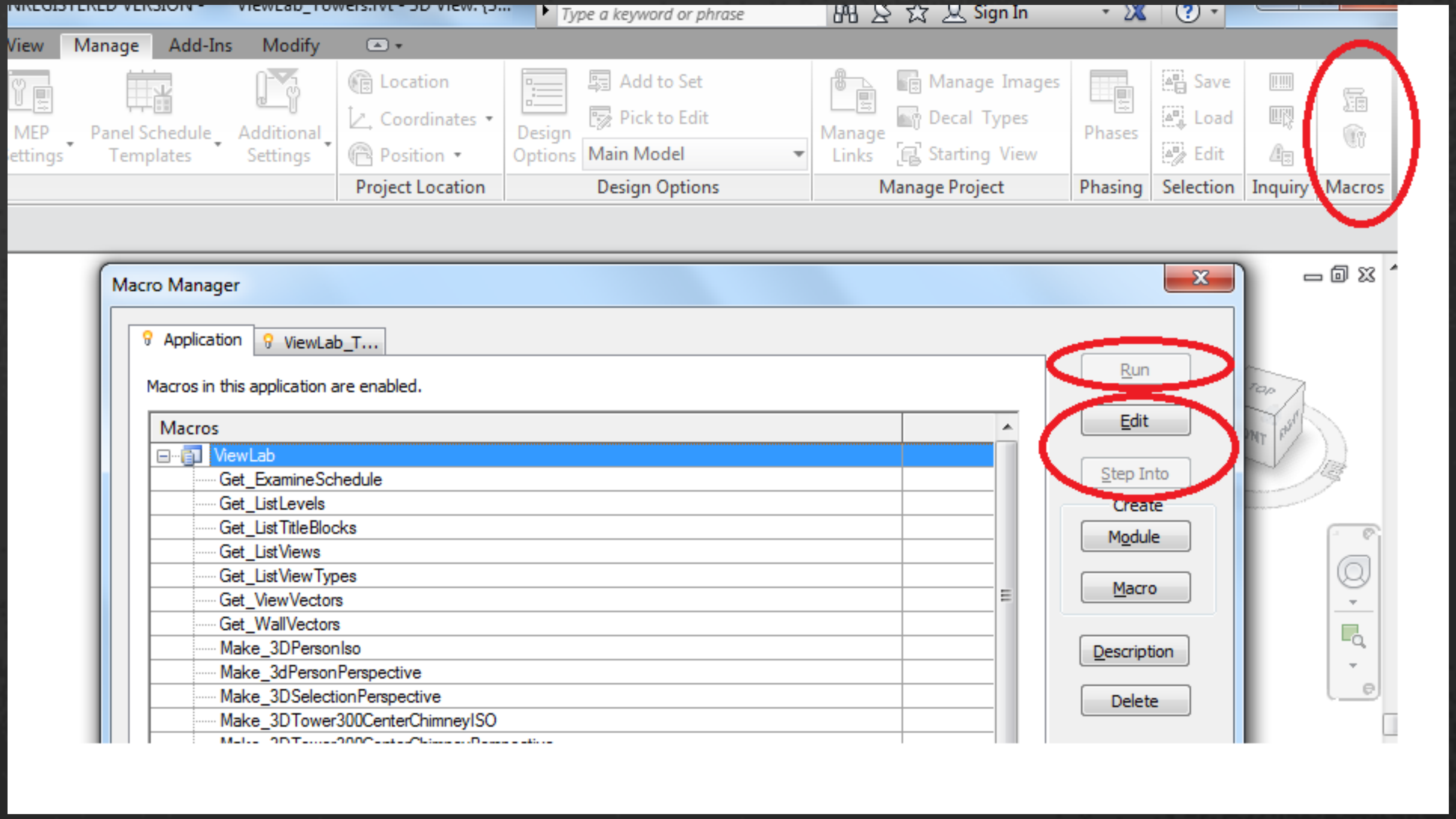
Recommended: Liberal use of

- Partial classes
- Region Codes
- Helper Classes
- Generic Utility methods

Tips

Inside the sample code:

- ThisApplication
  - Entrypoint into UI
  - Access to ViewOperations and DocumentUtility
- ViewOperations
  - View API specific code
- DocumentUtility
  - Document Access
  - Selection
  - Naming and existence checking



Macro Manager

Application | ViewLab_T...

Macros in this application are enabled.

Macros
- ViewLab
  - Get_ExamineSchedule
  - Get_ListLevels
  - Get_ListTitleBlocks
  - Get_ListViews
  - Get_ListViewTypes
  - Get_ViewVectors
  - Get_WallVectors
  - Make_3DPersonIso
  - Make_3dPersonPerspective
  - Make_3DSelectionPerspective
  - Make_3DTower300CenterChimneyISO

Tips

1. DocumentUtility Pattern – consolidate all your document access, selection, and naming code
2. Selection utility – handle pre-selection, post-selection, and selection filtering all at once.
3. Dialog Pattern -- Create a simple data collection shell you can implement quickly.
4. Debug Pattern – a standard, reusable way to get diagnostic info.
5. SuffixBuilder -- A tool for generating unique names necessary for Revit elements.
6. Enum.Parse – Taking string enum names from the keyboard and passing them to Revit as enums
7. Named locations – See the XYZ class, and create a constants class for other ones.
8. Exceptions – Use them to make your error handling simple and consistent.

- We will point out these tips along the way.
- Search for "Tip – " in the PowerPoint and source code.

View Creation

Getting started with View creation

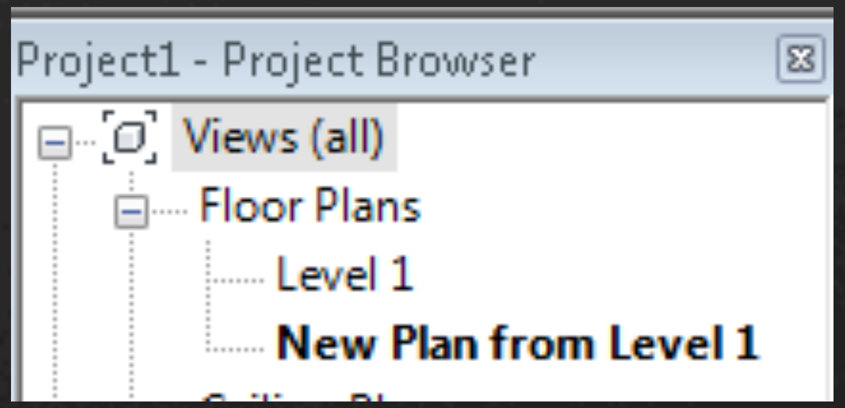
- Most view creation methods require a ViewFamilyType.
- Often, documents have only one ViewFamilyType for each ViewType.
- API Demo – show View Family types.
- Tip – DocumentUtility
- Tip - DebugPattern

Creating Plan Views

Inputs: document, ViewFamilyType, Level

```
Transaction transaction = new Transaction(document, "Create Plan View");
transaction.Start();
  ViewPlan viewPlan = ViewPlan.Create(document, viewFamilyTypes.First().Id, levels.First().Id);
  viewPlan.Name = "New Plan from " + levels.First().Name;
transaction.Commit();

UIDocument uidoc = new UIDocument(document);
// make the new view the active view
uidoc.ActiveView = viewPlan;
```



We can also set the plan view cut plane height

- Typically 3-4 feet from floor.

- API Demo – create floor plan views
- Tip - DialogPattern

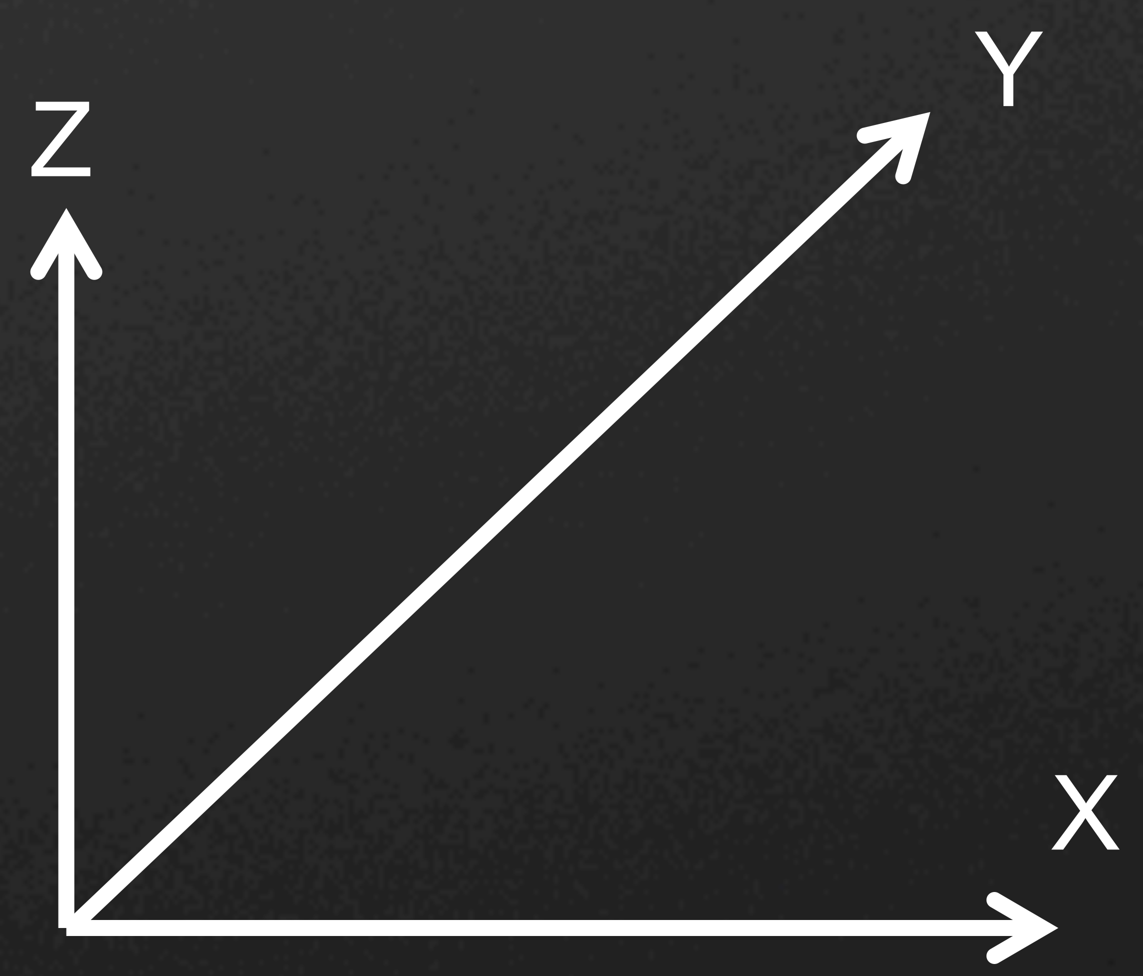
3D Views: Revit Coordinate System

Most of the time (in Revit)
+Z is up towards the sky
+X is East (to the right)
+Y is North (into the screen)
(Right handed rule)

Assuming you’re looking at a map where
North is pointing away from you.

We bend this rule a little bit with Section
Boxes and BoundingBoxXYZ -- more later

When working with 3d Views, you need a direction vector
and an “up” vector **relative to that direction vector.**



Calculating a direction “up” vector (Handout)

English

1. Decide what direction is globally “up” (the opposite direction of a plumb bob).
2. Calculate your direction vector by subtracting your eye point from your target point.
3. Get 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

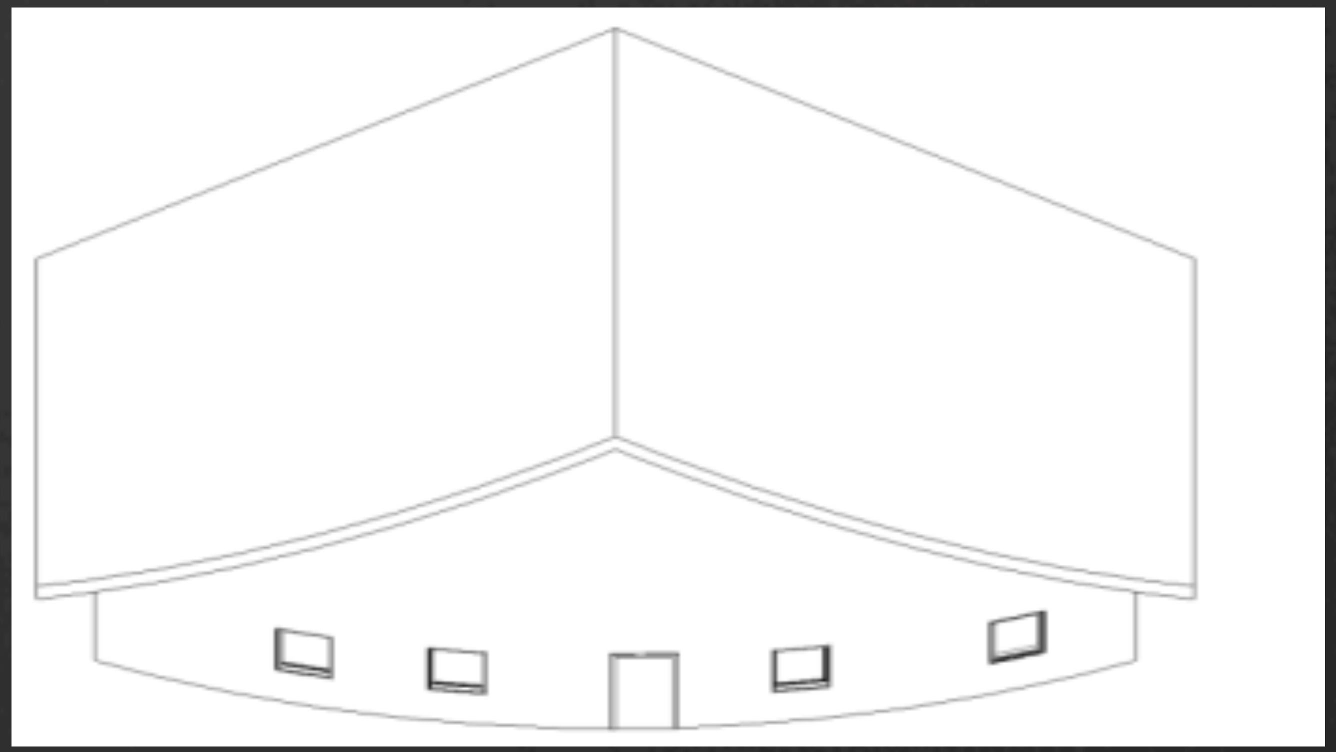
```
1. globalUp = (0,0,1)  (or XYZ.BasisZ)
2. direction = targetPoint – eyePoint
3. localUp = direction.Cross(globalUp).Cross(direction)
```

Recommendation: Make a library of helper functions for things like this.

```
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);
    }
```

- Tip - NamedLocations

Creating 3D Isometric Views



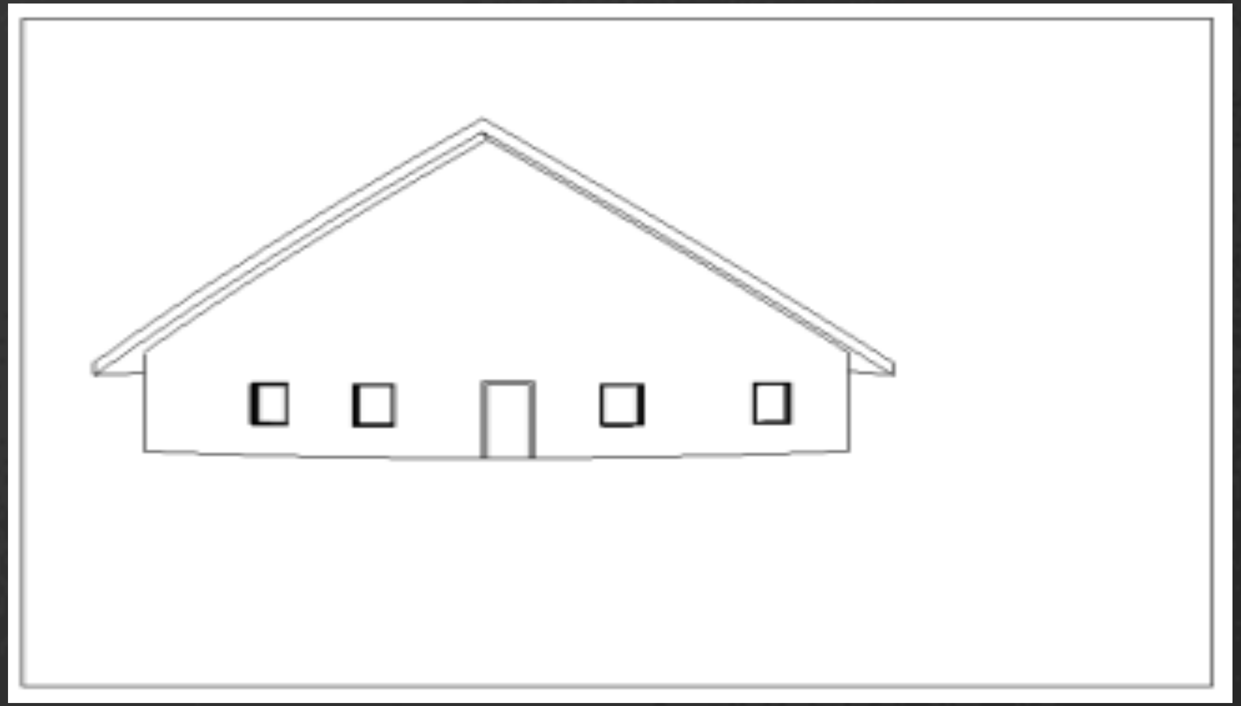
Inputs: document, ViewFamilyType

```
View3D view3D = View3D.CreateIsometric(document, viewFamilyTypes.First().Id);

// By default, the 3D view uses a default orientation.
// Change the orientation by creating and setting a ViewOrientation3D
XYZ eye = new XYZ(10, 10, 10);
XYZ up = new XYZ(0, 1, 1);
XYZ forward = new XYZ(0, 1, -1);
ViewOrientation3D viewOrientation3D = new ViewOrientation3D(eye, up, forward);
view3D.SetOrientation(viewOrientation3D);
```

API Demo – Create 3D Iso Views

Creating 3D Perspective Views



Inputs: document, ViewFamilyType

```
View3D view3D = View3D.CreatePerspective(document, viewFamilyTypes.First().Id);

XYZ eye = new XYZ(0,-100, 10);
XYZ up = new XYZ(0, 0, 1);
XYZ forward = new XYZ(0, 1, 0);

view3D.SetOrientation(new ViewOrientation3D(eye, up, forward));
// turn off the far clip plane with standard parameter API
Parameter farClip = view3D.get_Parameter("Far Clip Active");
farClip.Set(0);
```

- API Demo
  - Query view vectors
  - Create view vectors
  - Create 3D Perspective views

Creating Section Views

Inputs: document, ViewFamilyType, BoundingBoxXYZ

```
ViewSection section = ViewSection.CreateSection (document, viewFamilyTypes.First().Id, sectionBox);
```

How to create the proper section box is where it gets interesting…

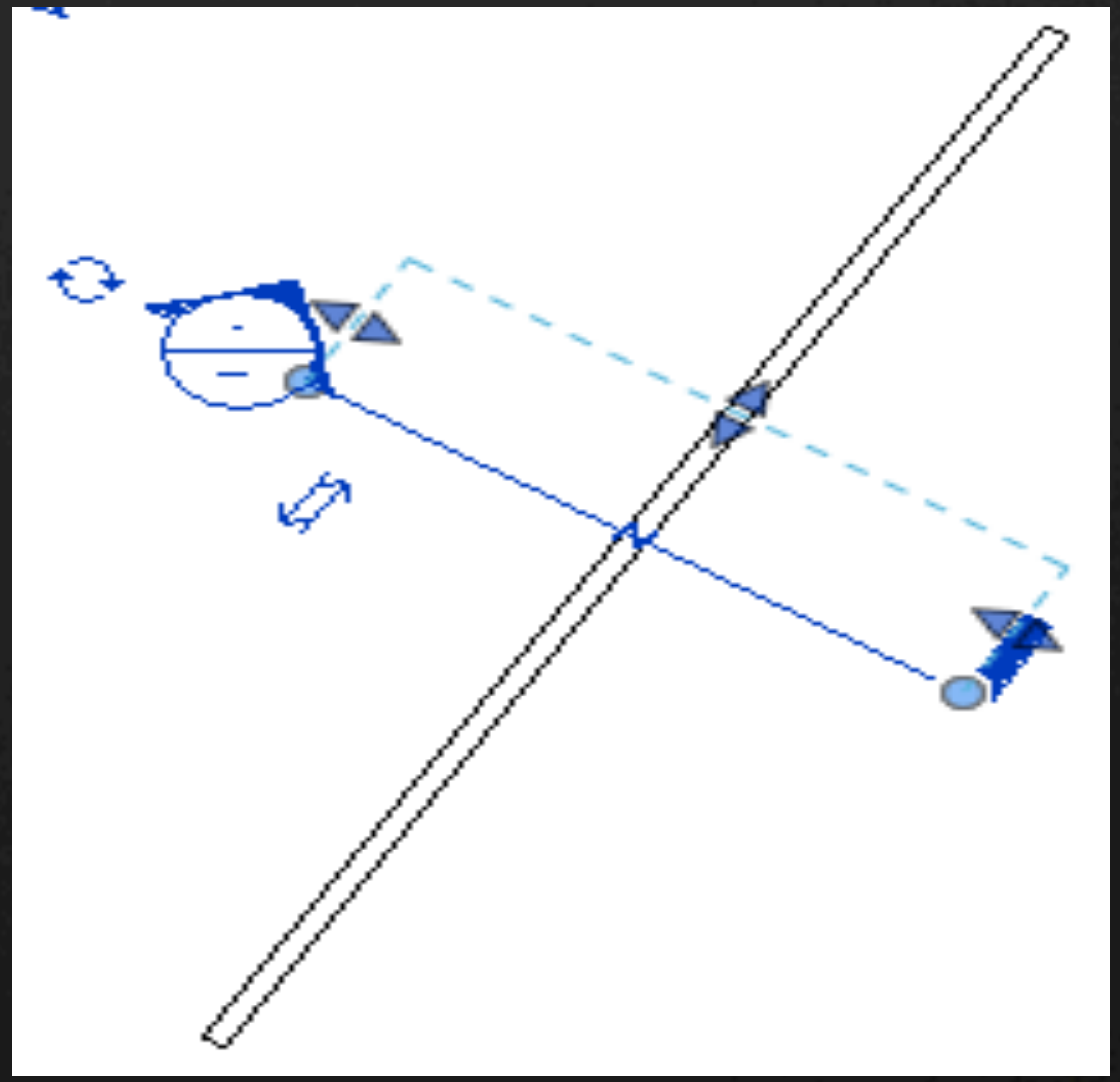
Computing the Section Box (Handout)

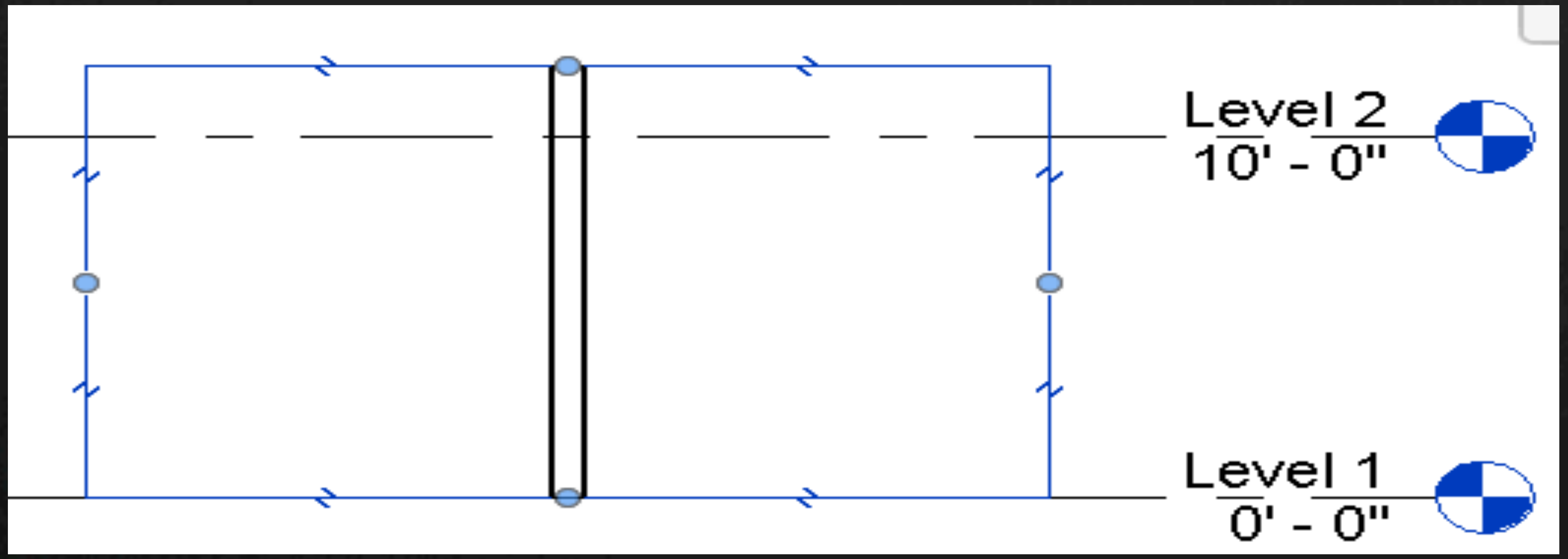
Need a Transform based on an
Origin (wall center, in this demo)
normal direction (direction out from wall)
& up direction to orient the section line

Need a BoundingBoxXYZ to specify the far clip offset and the length of the section line

The BoundingBoxXYZ contains clip offsets as well as a transform.

Test in ViewLab_Sections.rvt





Section Box to place Section Mid-Wall

- API Demo – create a few section views
- Tip - SelectionUtility

Creating Detail Views (Handout)

Inputs: document, ViewFamilyType, BoundingBoxXYZ

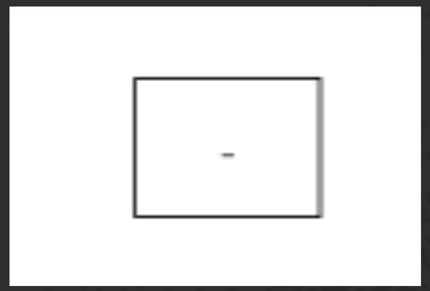
ViewSection detail = ViewSection.**CreateDetail**(document, ElementId viewFamilyType , sectionBox);

- Very similar to section views.
- Be sure to crop your view, or you will wonder why nothing happened.
- API Demo – creating detail views in ViewLab_Towers.rvt
- Tip - ExceptionPattern

Elevation

Elevation Marker + Plan View = Elevation View

1. Create the ElevationMarker

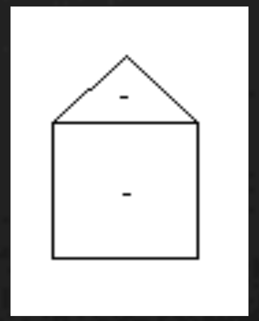


Marker with no view

- Document, ViewFamilyType, Origin – store this origin point for later
- Scale
  (ratio of true model size to paper size, e.g. use 96 to get 1/8” = 1’)

```
ElevationMarker marker =
  ElevationMarker.CreateElevationMarker(document, viewFamilyType, xyzPoint , 12*8);
```

1. Create the Elevation view
   - Document
   - Id of ViewPlan in which the ElevationMarker is visible
   - Index on the ElevationMarker where the new elevation will be placed



Marker with 1 view

```
ViewSection elevationView =
  marker.CreateElevation(document, planView.Id, 1);
```

Rotating the Elevation Marker

- There is no location point you can get from an elevation marker.
- Save the location point you used when you placed the marker to begin with.

```
Line axis = Line.get_Unbound(markerLocation, XYZ.BasisZ);
ElementTransformUtils.RotateElement(Doc(), marker.Id, axis , rotation);
```

Setting the Far Clip Offset

- Nearly all views have some sort of clipping range you can define.
- For elevations views, it is controlled by BuiltinParameter.VIEWER_BOUND_OFFSET_FAR.
- You may want to set this to a high value initially and then scale back after deciding what background elements are appropriate.

- API Demo – creating elevation views
- Tip - SuffixBuilder

Schedule Views

Schedules

Schedules can be created, modified, and added to drawing sheets.

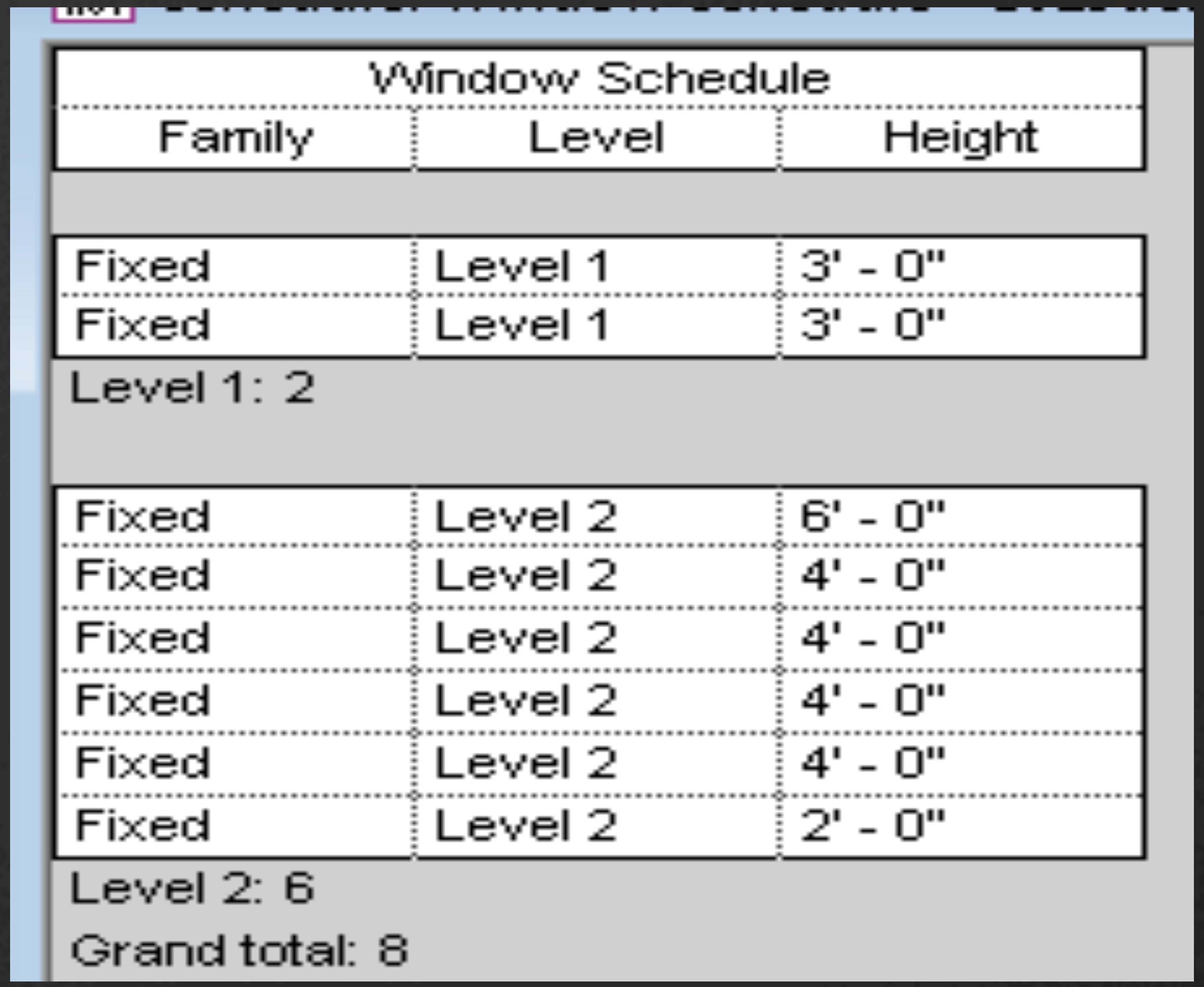
- ViewSchedule represents the schedule view.
  - Contains several create methods to make new schedules. (View list, part list,
- ScheduleableField – a field that could be added to a given schedule
- ScheduleField – the individual fields in a schedule
- ScheduleSheetInstance represents schedules placed on sheets.
  - The create method creates an instance of a schedule on a sheet.
- ScheduleDefinition defines the contents of a schedule view, including:
  - Properties that determine the kind of schedule, such as the schedule's category.
  - Fields that become the columns of the schedule.
  - Filters that restrict the set of elements visible in the schedule. (ScheduleFilter)
  - Sorting and grouping criteria. (ScheduleSortGroupField)
  - Run in ViewLab_Schedules.rvt

Extracting Schedule Data

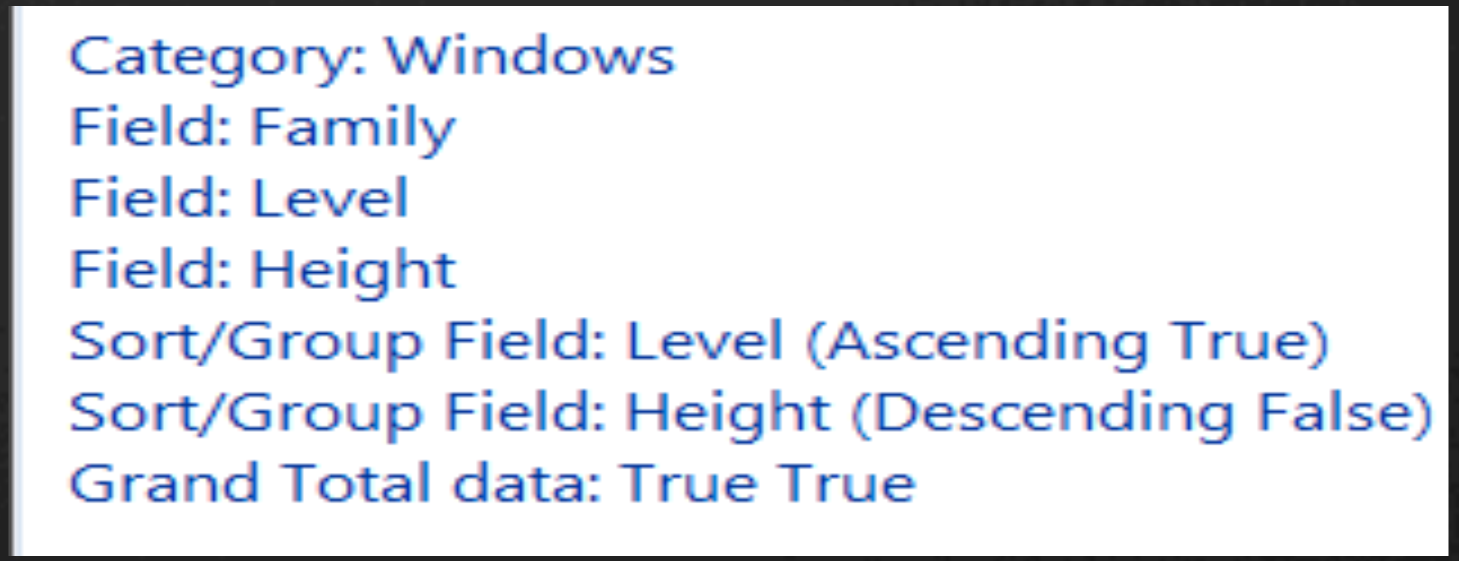
Items that can be extracted from Schedules:

- Field names and Ids
- Category Ids
- Filtering and grouping data

- API Demo – query a schedule



Window Schedule		
Family	Level	Height
Fixed	Level 1	3' - 0"
Fixed	Level 1	3' - 0"
Level 1: 2		
Fixed	Level 2	6' - 0"
Fixed	Level 2	4' - 0"
Fixed	Level 2	4' - 0"
Fixed	Level 2	4' - 0"
Fixed	Level 2	4' - 0"
Fixed	Level 2	2' - 0"
Level 2: 6		
Grand total: 8		



```
Category: Windows
Field: Family
Field: Level
Field: Height
Sort/Group Field: Level (Ascending True)
Sort/Group Field: Height (Descending False)
Grand Total data: True True
```

Schedule Creation – Add Fields

To add a field, you need:

1) A parameter
2) A ScheduleableField

```
ScheduleField areaField =  definition.AddField(new SchedulableField(ScheduleFieldType.ViewBased,
new ElementId(BuiltInParameter.ROOM_AREA)));

ScheduleField perimeterField = definition.AddField(new SchedulableField(ScheduleFieldType.ViewBased,
new ElementId(BuiltInParameter.ROOM_PERIMETER)));
```

Note the ScheduleFieldType…

Schedule Creation – Sort & Filter (Handout)

We can sort schedules as well as filter them

```
//Sorting
definition.AddSortGroupField( new ScheduleSortGroupField(numberField.FieldId,  ScheduleSortOrder.Descending));

//Filtering
definition.AddFilter(new ScheduleFilter(areaField.FieldId,
    ScheduleFilterType.GreaterThan, 10.0));
```

- API Demo – creating schedule view with fields, sorting, and filtering
- Tip - EnumParse

Schedule API – not implemented in Revit 2013

- Calculated fields
- Conditional Formatting
- Appearance
- Split schedules on a sheet
- Grouping headers
- Unit format options for schedule fields
- Access to the grid of cells for data access

Sheet Views

Sheet Creation

- Find a title block and create
- Quick LINQ+ ElementFilter trick to get a title block

```
IEnumerable< FamilySymbol> familyList = from elem in new FilteredElementCollector (document)
    OfClass( typeof (FamilySymbol))
    .OfCategory( BuiltInCategory.OST_TitleBlocks)
    let type = elem as FamilySymbol
    where type.Name.Contains("E1")
    select type;

ViewSheet sheet = ViewSheet.Create (document, familyList.First().Id);
```

Placing Views on Sheets

"Regular" views use the Viewport class
(plan, elevation, drafting, 3d, etc.)

```
if (Viewport.CanAddViewToSheet(document, sheet.Id, viewPlan.Id))
        // false if view is already on the sheet
 Viewport viewport = Viewport.Create(document, sheet.Id, viewPlan.Id, new XYZ(0,0,0));
```

Origin units are feet from lower-left-sheet corner to site origin on view.

Schedules are different:

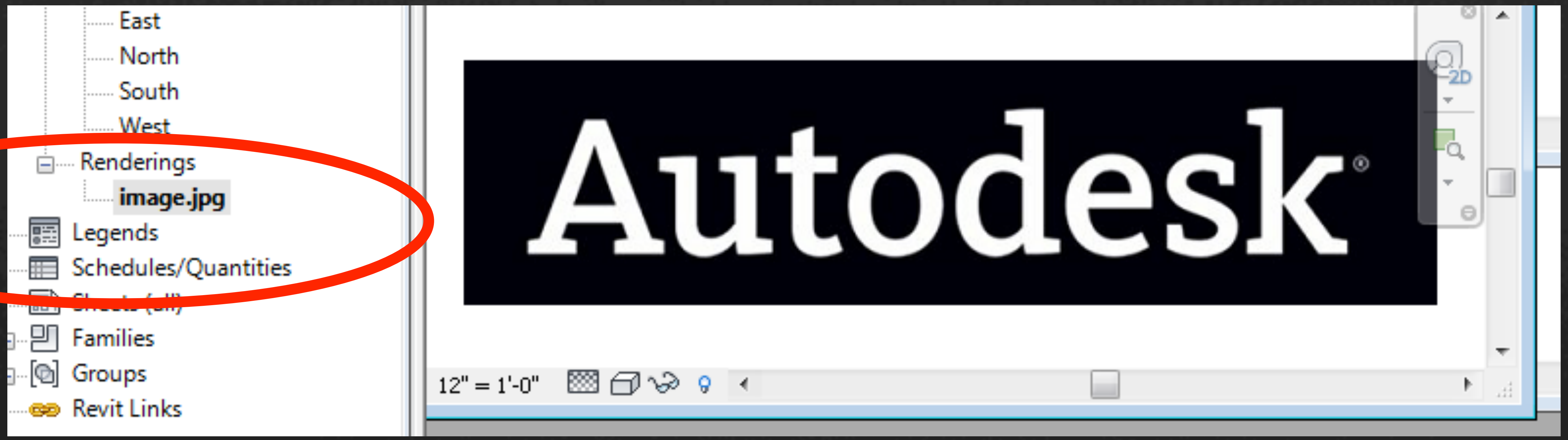
```
ScheduleSheetInstance.Create(document, sheet.Id, mySchedule.Id, new XYZ(0,0,0));
```

- API Demo – add view to sheet

Other View Operations

ImageView Class for Rendering views

ImageView imageView = Autodesk.Revit.DB.ImageView.**Create**(document, “C:\\work\\image.jpg");



Zoom Views to Selected Element (Handout)

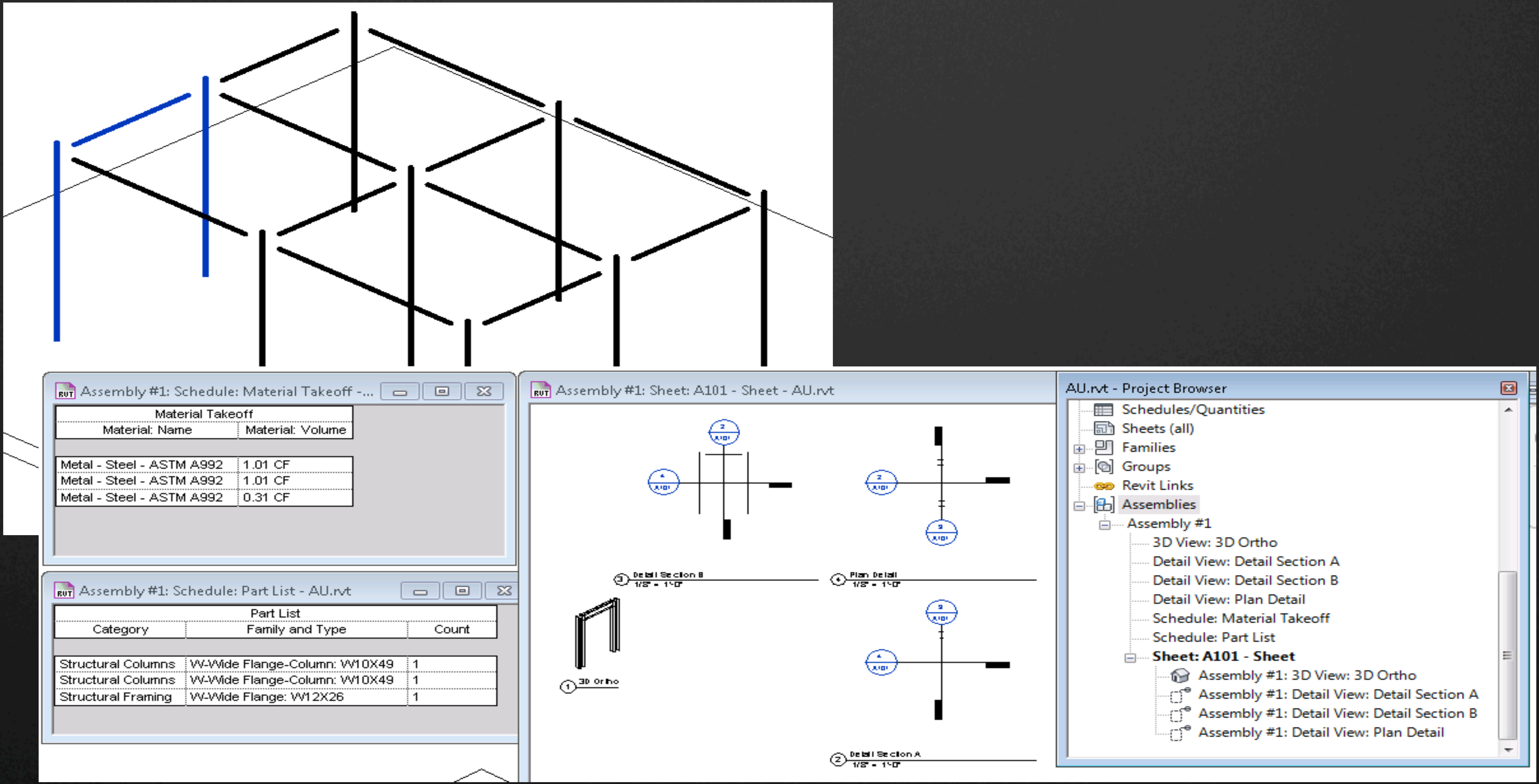
```
private void ZoomElementToViews(Element element)
{
    if (element == null)
        throw new ArgumentNullException("element");

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

- API Demo – Zoom to selected

Assembly Views

- Collections of elements
  - Can be used to create shop drawings for prefabricated building components
  - Can be scheduled, visually isolated, and tagged.
- Key methods
  - AssemblyInstance.Create()
  - AssemblyInstance.GetMemberIds()
  - AssemblyInstance.SetMemberIds()
  - AssemblyInstance.Disassemble()
  - AssemblyViewUtils.CreateDetailSection()
  - AssemblyViewUtils.CreateMaterialTakeOff()
  - AssemblyViewUtils.CreatePartList();
- Assembly Views display only the elements in the assembly



Assembly #1: Schedule: Material Takeoff -...

Material Takeoff	
Material: Name	Material: Volume
Metal - Steel - ASTM A992	1.01 CF
Metal - Steel - ASTM A992	1.01 CF
Metal - Steel - ASTM A992	0.31 CF

Assembly #1: Schedule: Part List - AU.rvt

Part List		
Category	Family and Type	Count
Structural Columns	W-Wide Flange-Column: W10X49	1
Structural Columns	W-Wide Flange-Column: W10X49	1
Structural Framing	W-Wide Flange: W12X26	1

Assembly #1: Sheet: A101 - Sheet - AU.rvt

AU.rvt - Project Browser

- Schedules/Quantities
- Sheets (all)
- Families
- Groups
- Revit Links
- Assemblies
  - Assembly #1
    - 3D View: 3D Ortho
    - Detail View: Detail Section A
    - Detail View: Detail Section B
    - Detail View: Plan Detail
    - Schedule: Material Takeoff
    - Schedule: Part List
    - **Sheet: A101 - Sheet**
      - Assembly #1: 3D View: 3D Ortho
      - Assembly #1: Detail View: Detail Section A
      - Assembly #1: Detail View: Detail Section B
      - Assembly #1: Detail View: Plan Detail

- API Demo – create assembly views