Wind Turbine design based on QuietRevolution.

Model created by Spine 3D

Introduction to Revit Programming

Database Fundamentals

Developer Technical Services

Agenda

Introduction

- Products, SDK, documentation and samples

Getting Started and Hello World

- Development environment, Revit add-ins, external command and application, add-in manifest, RvtSamples and RevitLookup

Database Fundamentals

- Understanding the representation of Revit elements
- Element iteration, filtering and queries
- Element modification
- Model creation

Overview

Products, SDK and assembly dlls

Revit Products

Revit Architecture,Revit MEP and Revit Structure are no longer separate products

Product build and distribution

- DVD version posted to ADN member web site (members only)
  - Software & Support > Revit > Downloads
  - Posted once only at initial product release time
- Web version and Web Update version on Autodesk home page (public)
  - *Products > Revit > Buy or Store > USA & Canada ($ USD)> All Products > Revit Architecture/Structure/MEP*
  - Latest download version from the public product site
  - Revit uses service pack technology, so no need for full installation on update

Revit API Assemblies

Revit API assembly DLLs are present in Revit installation

- RevitAPI.dll
- RevitAPIUI.dll

Separate DB and UI modules for database and user interface

Revit SDK

The SDK is provided with the product

- From installer under “Install Tools and Utilities”
- Web and download version

  <extraction folder>\Utilities\SDK\RevitSDK.exe

Latest SDK update is posted to Revit Developer Center

- http://www.autodesk.com/developrevit

SDK is only Documentation

Read once

- Read Me First.doc
- Getting Started with the Revit API.docx
- Revit Platform API Changes and Additions.docx

Familiarize yourself with

- Revit API Developer's Guide (http://www.autodesk.com/revitapi-help)
- RevitAPI.chm
  - What's New section is similar to Changes and Additions doc

Read if needed

- RevitAddInUtility.chm – installer
- Autodesk Icon Guidelines.pdf – user interface
- Macro Samples – Revit Macros
- Revit Server SDK – file access on server
- Revit Structure – section definitions and material properties
- REX SDK – Revit extensions framework
- Structural Analysis SDK – Analysis and code checking

Important utilities

- Add-In Manager

Samples

SDK Samples

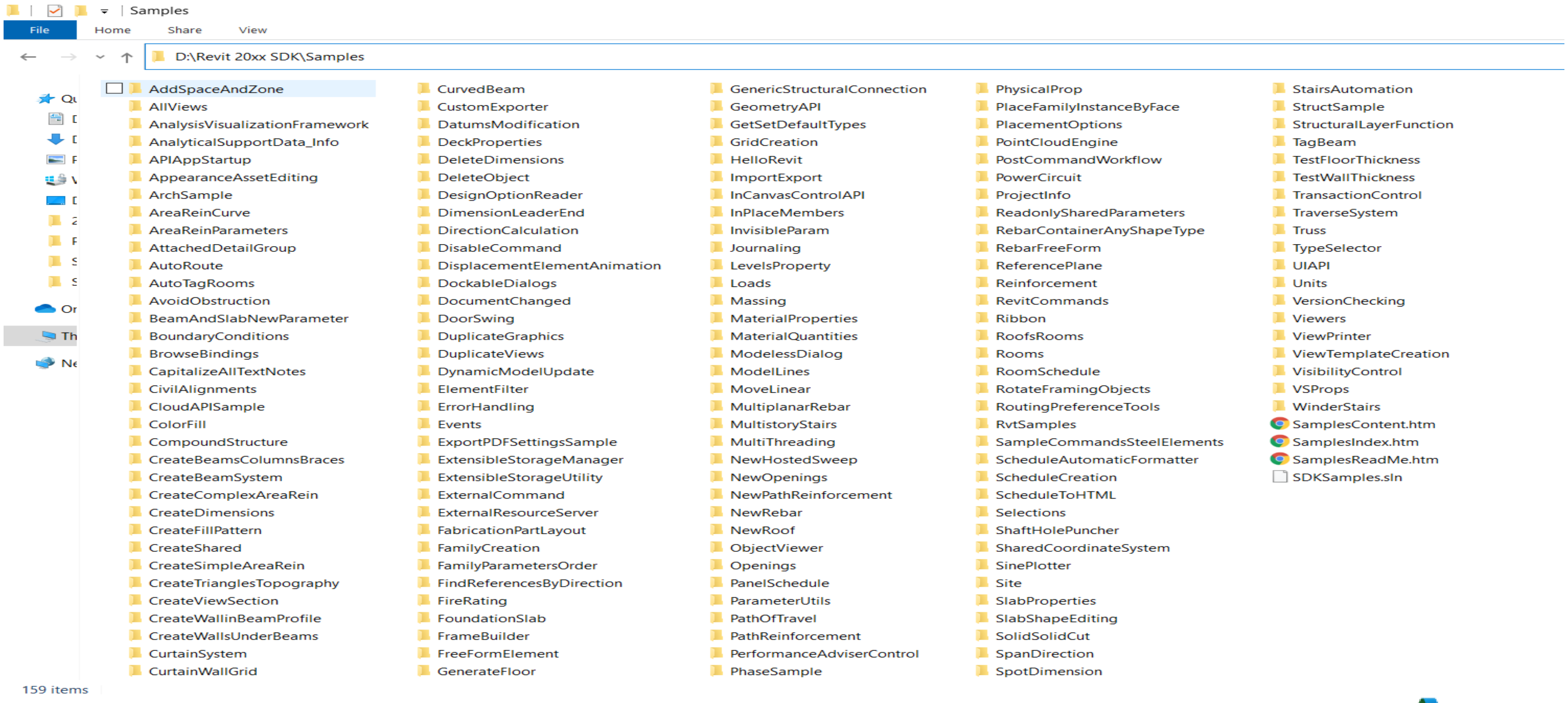
Documentation

- ReadMeFirst.doc

Main samples solution

- SDKSamples.sln

And the samples themselves!



D:\Revit 20xx SDK\Samples

AddSpaceAndZone
AllViews
AnalysisVisualizationFramework
AnalyticalSupportData_Info
APIAppStartup
AppearanceAssetEditing
ArchSample
AreaReinCurve
AreaReinParameters
AttachedDetailGroup
AutoRoute
AutoTagRooms
AvoidObstruction
BeamAndSlabNewParameter
BoundaryConditions
BrowseBindings
CapitalizeAllTextNotes
CivilAlignments
CloudAPISample
ColorFill
CompoundStructure
CreateBeamsColumnsBraces
CreateBeamSystem
CreateComplexAreaRein
CreateDimensions
CreateFillPattern
CreateShared
CreateSimpleAreaRein
CreateTrianglesTopography
CreateViewSection
CreateWallinBeamProfile
CreateWallsUnderBeams
CurtainSystem
CurtainWallGrid
CurvedBeam
CustomExporter
DatumsModification
DeckProperties
DeleteDimensions
DeleteObject
DesignOptionReader
DimensionLeaderEnd
DirectionCalculation
DisableCommand
DisplacementElementAnimation
DockableDialogs
DocumentChanged
DoorSwing
DuplicateGraphics
DuplicateViews
DynamicModelUpdate
ElementFilter
ErrorHandling
Events
ExportPDFSettingsSample
ExtensibleStorageManager
ExtensibleStorageUtility
ExternalCommand
ExternalResourceServer
FabricationPartLayout
FamilyCreation
FamilyParametersOrder
FindReferencesByDirection
FireRating
FoundationSlab
FrameBuilder
FreeFormElement
GenerateFloor
GenericStructuralConnection
GeometryAPI
GetSetDefaultTypes
GridCreation
HelloRevit
ImportExport
InCanvasControlAPI
InPlaceMembers
InvisibleParam
Journaling
LevelsProperty
Loads
Massing
MaterialProperties
MaterialQuantities
ModelessDialog
ModelLines
MoveLinear
MultiplanarRebar
MultistoryStairs
MultiThreading
NewHostedSweep
NewOpenings
NewPathReinforcement
NewRebar
NewRoof
ObjectViewer
Openings
PanelSchedule
ParameterUtils
PathOfTravel
PathReinforcement
PerformanceAdviserControl
PhaseSample
PhysicalProp
PlaceFamilyInstanceByFace
PlacementOptions
PointCloudEngine
PostCommandWorkflow
PowerCircuit
ProjectInfo
ReadonlySharedParameters
RebarContainerAnyShapeType
RebarFreeForm
ReferencePlane
Reinforcement
RevitCommands
Ribbon
RoofsRooms
Rooms
RoomSchedule
RotateFramingObjects
RoutingPreferenceTools
RvtSamples
SampleCommandsSteelElements
ScheduleAutomaticFormatter
ScheduleCreation
ScheduleToHTML
Selections
ShaftHolePuncher
SharedCoordinateSystem
SinePlotter
Site
SlabProperties
SlabShapeEditing
SolidSolidCut
SpanDirection
SpotDimension
StairsAutomation
StructSample
StructuralLayerFunction
TagBeam
TestFloorThickness
TestWallThickness
TransactionControl
TraverseSystem
Truss
TypeSelector
UIAPI
Units
VersionChecking
Viewers
ViewPrinter
ViewTemplateCreation
VisibilityControl
VSProps
WinderStairs
SamplesContent.htm
SamplesIndex.htm
SamplesReadMe.htm
SDKSamples.sln

159 items

AUTODESK.

Extending Revit

1. External command

- Implement IExternalCommand; install an add-in manifest
- Commands are added to the External Tools pulldown in the ribbon Add-Ins tab
- Tools > External Tools

2. External application

- Implement IExternalApplication; install an add-in manifest
- Applications can create new panels in the ribbon Add-Ins tab
- External applications can invoke external commands

3. SharpDevelop macro *) not today's focus

Revit Add-In Compilation and API References

- .NET API
- .NET Framework 4.8
- Microsoft Visual Studio 2017
- C# or VB.NET, managed C++, any .NET compliant language
- Class library
- References
  - <revit install folder>\Program\RevitAPI.dll
  - <revit install folder>\Program\RevitAPIUI.dll
  - Remember to set 'Copy Local' to False

Getting Started

First Steps to Hello World: External command and add-ins manifest

Steps to Hello World External Command

- New .NET class library
- References (minimum):
  - RevitAPI.dll
  - RevitAPIUI.dll
- Most commonly used namespaces
  - Autodesk.Revit.DB
  - Autodesk.Revit.UI
  - Autodesk.Revit.ApplicationServices
  - Autodesk.Revit.Attributes
  - If you use VB.NET, set namespaces in project properties
- Implement IExternalCommand and Execute() method
- Create and install the add-in manifest file

Implementing External Command

Minimum Code in VB.NET

```
<VB.NET>
''  Hello World #1 - A minimum Revit external command.
<Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)> _
Public Class HelloWorld
    Implements IExternalCommand

    Public Function Execute( _
            ByVal commandData As Autodesk.Revit.UI.ExternalCommandData, _
            ByRef message As String, _
            ByVal elements As Autodesk.Revit.DB.ElementSet) _

            As Autodesk.Revit.UI.Result _
            Implements Autodesk.Revit.UI.IExternalCommand.Execute

        Autodesk.Revit.UI.TaskDialog.Show("My Dialog Title", "Hello World!")

        Return Result.Succeeded

    End Function

End Class
</VB.NET>
```

Implementing External Command

Minimum Code in C#

```
<C#>
// Hello World #1 - A minimum Revit external command.
[Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)]
public class HelloWorld : IExternalCommand
{
    public Autodesk.Revit.UI.Result Execute(
        Autodesk.Revit.UI.ExternalCommandData commandData,
        ref string message,
        Autodesk.Revit.DB.ElementSet elements)
    {
        Autodesk.Revit.UI.TaskDialog.Show("My Dialog Title", "Hello World!");

        return Result.Succeeded;
    }
}
</C#>
```

Implementing External Command

IExternalCommand Class

```
<VB.NET>
''  Hello World #1 - A minimum Revit external command.
<Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)> _
Public Class HelloWorld
    Implements IExternalCommand


Public Function Execute( _
            ByVal commandData As Autodesk.Revit.UI.ExternalCommandData, _
            ByRef message As String, _
            ByVal elements As Autodesk.Revit.DB.ElementSet) _

            As Autodesk.Revit.UI.Result _
            Implements Autodesk.Revit.UI.IExternalCommand.Execute

        Autodesk.Revit.UI.TaskDialog.Show("My Dialog Title", "Hello World!")

        Return Result.Succeeded

    End Function

End Class
</VB.NET>
```

1.Derive a class from IExternalCommand

Implementing External Command

Execute() Method

```
<VB.NET>
''  Hello World #1 - A minimum Revit external command.
<Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)> _
Public Class HelloWorld
    Implements IExternalCommand

    Public Function Execute( _
            ByVal commandData As Autodesk.Revit.UI.ExternalCommandData, _
            ByRef message As String, _
            ByVal elements As Autodesk.Revit.DB.ElementSet) _

            As Autodesk.Revit.UI.Result _
            Implements Autodesk.Revit.UI.IExternalCommand.Execute

        Autodesk.Revit.UI.TaskDialog.Show("My Dialog Title", "Hello World!")

        Return Result.Succeeded

    End Function

End Class
</VB.NET>
```

2.Implement Execute() method

Arguments:
1st Access to the Revit object model
2nd Message to the user when a command fails
3rd A set of elements to be highlighted when a command fails

Return value:
- Succeeded
- Failed
- Cancelled

AUTODESK.

Implementing External Command

Attributes

```
<VB.NET>
''  Hello World #1 - A minimum Revit external command.
<Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)> _
Public Class HelloWorld
    Implements IExternalCommand

    Public Function Execute( _
            ByVal commandData As Autodesk.Revit.UI.ExternalCommandData, _
            ByRef message As String, _
            ByVal elements As Autodesk.Revit.DB.ElementSet) _

            As Autodesk.Revit.UI.Result _
            Implements Autodesk.Revit.UI.IExternalCommand.Execute

        Autodesk.Revit.UI.TaskDialog.Show("My Dialog Title", "Hello World!")

        Return Result.Succeeded

    End Function

End Class
</VB.NET>
```

3. Set attributes

A Transaction mode: controls the transaction behavior

- Manual
- ReadOnly

© 2015 Autodesk Developer Network

AUTODESK.

Implementing External Command

Show Hello World

```
<VB.NET>
''  Hello World #1 - A minimum Revit external command.
<Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)> _
Public Class HelloWorld
    Implements IExternalCommand

    Public Function Execute( _
            ByVal commandData As Autodesk.Revit.UI.ExternalCommandData, _
            ByRef message As String, _
            ByVal elements As Autodesk.Revit.DB.ElementSet) _

            As Autodesk.Revit.UI.Result _
            Implements Autodesk.Revit.UI.IExternalCommand.Execute

        Autodesk.Revit.UI.TaskDialog.Show("My Dialog Title", "Hello World!")

        Return Result.Succeeded

    End Function

End Class
</VB.NET>
```

4. Show a dialog with a message

Task Dialog:
Revit-style message box
to say “Hello World”

Add-in Manifest

Registration Mechanism

Automatically read by Revit at startup

Two locations: All Users, and <user> specific location

Windows 7,8,10

C:\ProgramData\Autodesk\Revit\Addins\202x

C:\Users\<user>\AppData\Roaming\Autodesk\Revit\Addins\202x

Add-in Manifest

.addin File

```
<?xml version="1.0" encoding="utf-8">
<RevitAddIns>
  <AddIn Type="Command">
    <Name>Hello World<Name>
    <Assembly>C:\...\HelloWorld.dll</Assembly>
    <FullClassName>IntroVb.HelloWorld</FullClassName>
    <Text>Hello World</Text>
    <AddInId>0B997216-52F3-412a-8A97-58558DC62D1E</AddInId>
    <VendorId>ADNP</VendorId>
    <VendorDescription>Autodesk, Inc. www.autodesk.com</VendorDescription>
</AddIn>
</RevitAddIns>
```

Information about:

- Type of the add-in: command or application
- Text that appears in Revit under [Add-Ins] tab >> [External Tools] panel
- Full class name including namespace

  Class name is Case Sensitive
- Full path to the dll
- GUID or a unique identifier of the command

More options

See Developer Guide section 3.4.1 (pp40)

Registered Developer Symbol for Vendor Id

The Vendor Id should be unique

A safe way to obtain a unique symbol:

- Use an Autodesk registered developer symbol (RDS)
- Google for "autodesk register developer symbol"

Symbols Registration on the Autodesk Developer Center

- Exactly four alphanumeric characters
- Cannot contain: %, ., @, *, [,], {, }, ^, $, /, \ or other special characters such as umlaut and accent

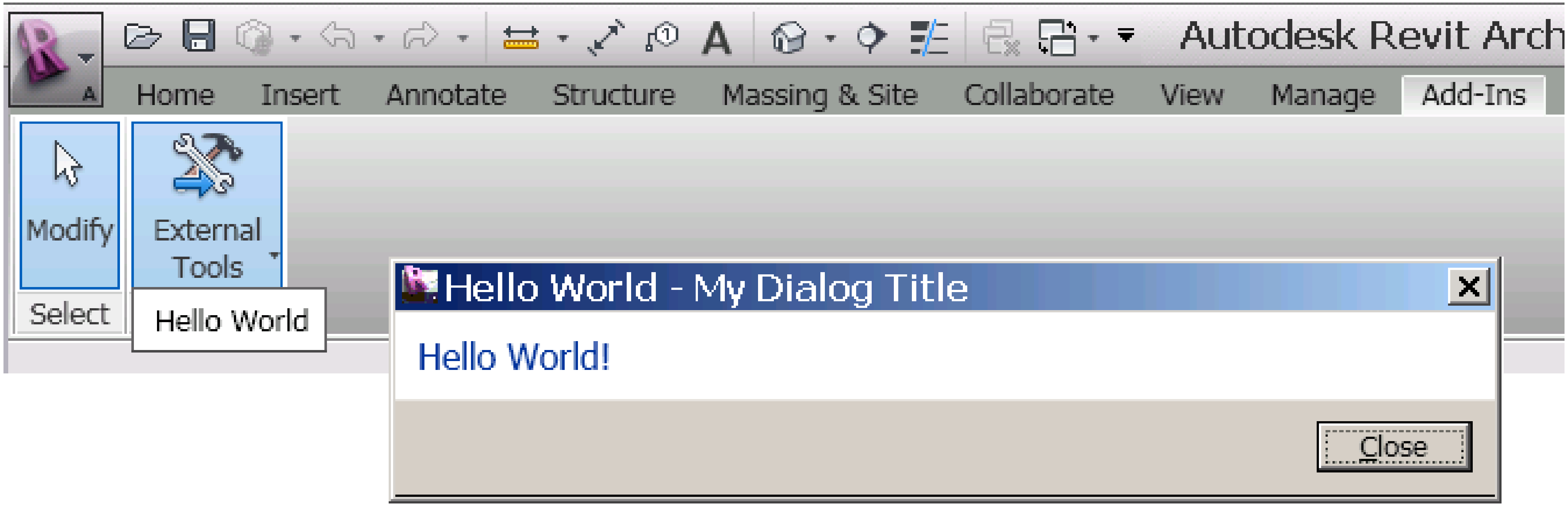
All ADN plug-ins use "ADNP" for "ADN Plugin"

External Tools Panel

Run Your Add-in

Once .addin manifest is in place, you will see [Add-Ins] tab and [External Tools] panel. (not visible with no add-ins)

Run your command from the pull down menu



Hello World - My Dialog Title

Hello World!

Carrying On …

External application and external command data

External Application

Minimum Code in VB.NET

```
<VB.NET>
''  Hello World App - minimum external application
Public Class HelloWorldApp
    Implements IExternalApplication
    ''  OnShutdown() - called when Revit ends.
    Public Function OnShutdown(ByVal application As UIControlledApplication) _
    As Result _
    Implements IExternalApplication.OnShutdown
        Return Result.Succeeded
    End Function
    ''  OnStartup() - called when Revit starts.
    Public Function OnStartup(ByVal application As UIControlledApplication) _
    As Result _
    Implements IExternalApplication.OnStartup
        TaskDialog.Show("My Dialog Title", "Hello World from App!")
        Return Result.Succeeded
    End Function
End Class
</VB.NET>
```

Implement IExternalApplication

OnShutdown is called when Revit ends

OnStartup is called when Revit starts

External Application

Minimum Code in VB.NET

<C#>

```
// Hello World #3 - minimum external application
public class HelloWorldApp : IExternalApplication
{
 // OnStartup() - called when Revit starts.
    public Result OnStartup(UIControlledApplication application)
    {
        TaskDialog.Show("My Dialog Title", "Hello World from App!");
        return Result.Succeeded;
    }

    // OnShutdown() - called when Revit ends.
    public Result OnShutdown(UIControlledApplication application)
    {
        return Result.Succeeded;
    }
}
```

</C#>

External Application

.addin Manifest

Type = “Application” instead of “Command”
<Name> instead of <Text>

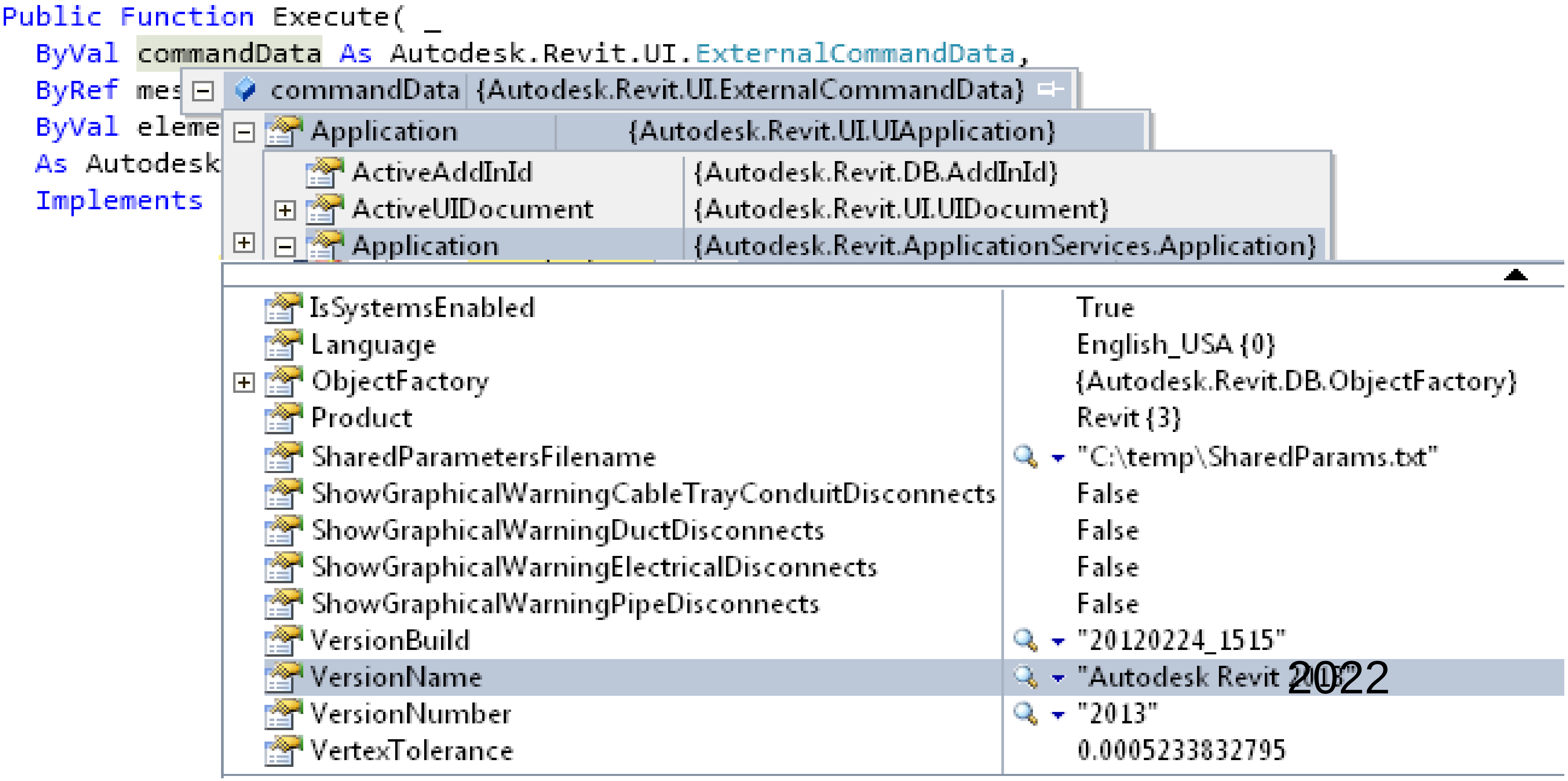
```
 <?xml version="1.0" encoding="utf-8" standalone="no"?>
<RevitAddIns>
<AddIn Type=“Application">
    <Name>Hello World App</Name>
    <FullClassName>IntroVb.HelloWorldApp</FullClassName>
    <Text>Hello World App</Text>
    <Description>Hello World</Description>
    <VisibilityMode>AlwaysVisible</VisibilityMode>
    <Assembly> C:\....\IntroVB.dll</Assembly>
    <AddInId>021BD853-36E4-461f-9171-C5ACEDA4E723</AddInId>
    <VendorId>ADSK</VendorId>
    <VendorDescription>Autodesk, Inc, www.autodesk.com</VendorDescription>
  </AddIn>
</RevitAddIns>
```

Command Data

Access to the Revit Object Model

ExternalCommandData = 1st argument of Execute() method

Top most object that allows us to access a Revit model



```
Public Function Execute( _
  ByVal commandData As Autodesk.Revit.UI.ExternalCommandData,
  ByRef mes
  ByVal eleme
  As Autodesk
  Implements
```

```
⊟ commandData   {Autodesk.Revit.UI.ExternalCommandData}
  ⊟ Application   {Autodesk.Revit.UI.UIApplication}
      ActiveAddInId        {Autodesk.Revit.DB.AddInId}
    ⊞ ActiveUIDocument     {Autodesk.Revit.UI.UIDocument}
  ⊞ ⊟ Application          {Autodesk.Revit.ApplicationServices.Application}

    IsSystemsEnabled                                True
    Language                                        English_USA {0}
  ⊞ ObjectFactory                                   {Autodesk.Revit.DB.ObjectFactory}
    Product                                         Revit {3}
    SharedParametersFilename                        "C:\temp\SharedParams.txt"
    ShowGraphicalWarningCableTrayConduitDisconnects False
    ShowGraphicalWarningDuctDisconnects             False
    ShowGraphicalWarningElectricalDisconnects       False
    ShowGraphicalWarningPipeDisconnects             False
    VersionBuild                                    "20120224_1515"
    VersionName                                     "Autodesk Revit 2013"
    VersionNumber                                   "2013"
    VertexTolerance                                 0.0005233832795
```

2022

© 2015 Autodesk Developer Network

Command Data

Access to the Revit Object Model

Examples:

```
''  access to the version of Revit and the title of the document currently in use
Dim versionName As String = _
    commandData.Application.Application.VersionName
Dim documentTitle As String = _
    commandData.Application.ActiveUIDocument.Document.Title
```

```
''  print out wall types available in the current rvt project
Dim collector As New FilteredElementCollector(rvtDoc)
Collector.OfClass(GetType(WallType))
Dim s As String = ""
    For Each wType As WallType In wallTypes
        s = s + wType.Name + vbCr
    Next
```

CommandData

Access to the Revit Object Model

Access to Application and Document in DB and UI portions

```
<VB.NET>
Public Class DBElement
    Implements IExternalCommand

    ''  member variables
    Dim m_rvtApp As Application
    Dim m_rvtDoc As Document

    Public Function Execute(ByVal commandData As ExternalCommandData, _
                            ...
        ''  Get the access to the top most objects.
        Dim rvtUIApp As UIApplication = commandData.Application
        Dim rvtUIDoc As UIDocument = rvtUIApp.ActiveUIDocument
        m_rvtApp = rvtUIApp.Application
        m_rvtDoc = rvtUIDoc.Document

        ''  ...
</VB.NET>
```

Tools

Revit Lookup, Add-In Manager, SDKSamples202x.sln, and RvtSamples

Tools

Must Know

RevitLookup – allows you to “snoop” into the Revit database structure. “must have” for any Revit API programmers. Available on ADN DevTech on Github

Add-In Manager – allows you to load your dll while running Revit without registering an addin and to rebuild dll without restarting Revit

SDKSamples20xx.sln – allows you to build all the sample projects at once. **RevitAPIDllsPathUpdater.exe** is provided to update the location of references in each MSVS projects in case your installation of Revit is different from the default location or if you are using different verticals.

RvtSamples – application that creates a ribbon panel for all the samples for easy testing

DB Element

Understanding the Representation of Revit Element

DB Element

Element Basics

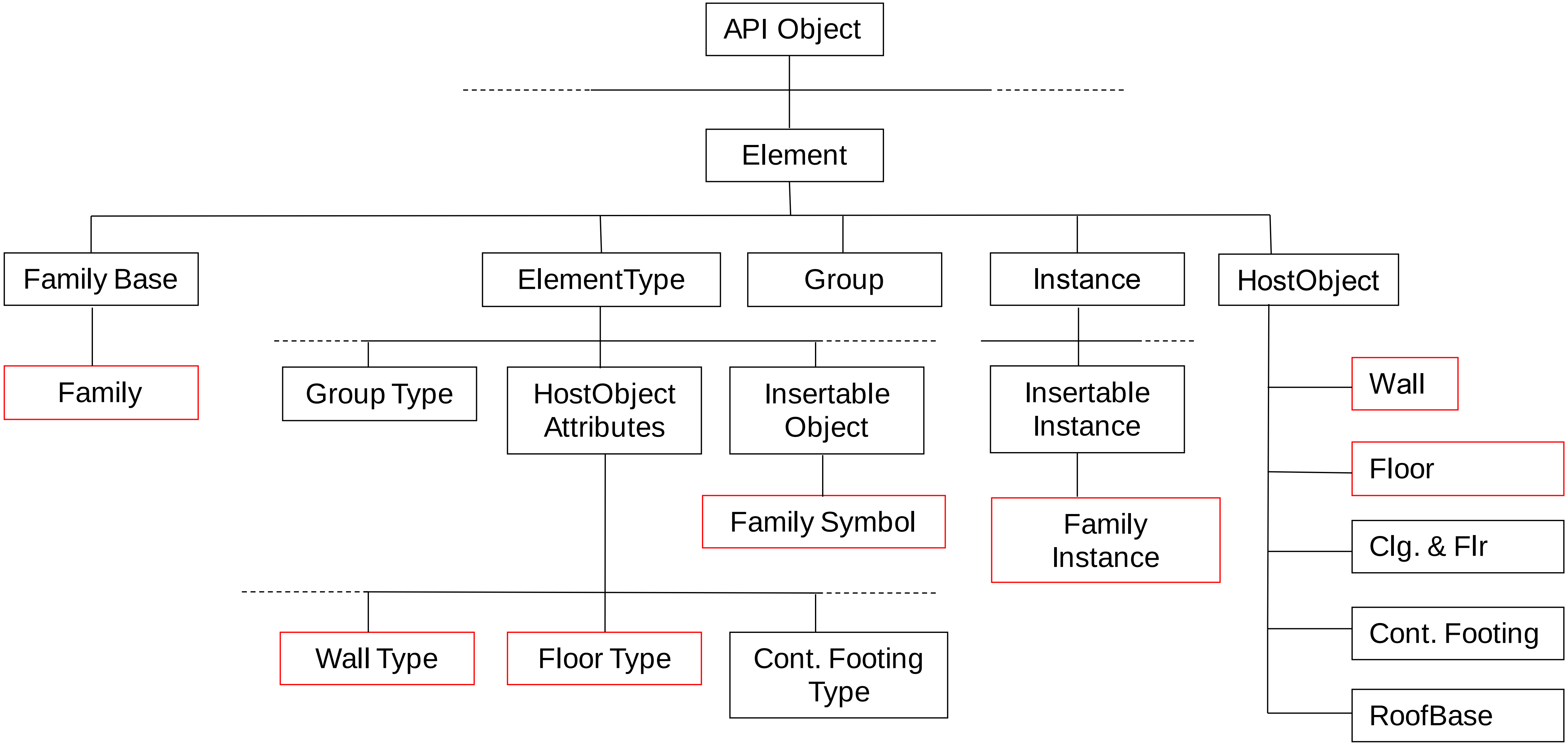
In typical programming, we identify the given object by checking its class name. Does the same apply to Revit API?

Answer is “not exactly”

Let’s take a look to understand why …

DB Element

Class Derivations



Families and types, aka symbols

Host and component objects, standard and system

DB Element

Element versus Symbol

Element			Symbol	
Kind of Element in UI	Derived from Element/TypeOf	Category	Derived from Symbol/TypeOf	Category
Wall	HostObject/Wall	Walls	HostObjAttributes/WallType	Walls
Door	Instance/InsertableInstance/FamilyInstance	Doors	InsertableObject /FamilySymbol	Doors
Door Tag	IndependentTag	Door Tags	InsertableObject /FamilySymbol	Door Tags
Window	Instance/InsertableInstance/FamilyInstance	Windows	InsertableObject /FamilySymbol	Windows
WindowTag	IndependentTag	Window Tags	InsertableObject /FamilySymbol	Window Tags
Opening	Opening	Rectangular Straight Wall Opening	< null >	---
Floor	HostObject/Floor	Floors	HostObjAttributes/FloorType	Floors
Ceiling	< Element >	Ceilings	HostObjAttributes	Ceilings
Roof	HostObject/RoofBase/FootPrintRoof,ExtrusionRoof	Roofs	HostObjAttributes/RoofType	Roofs
Column	Instance/InsertableInstance/FamilyInstance	Columns	InsertableObject /FamilySymbol	Columns
Component (Desk)	Instance/InsertableInstance/FamilyInstance	Furniture	InsertableObject /FamilySymbol	Furniture
Component (Tree)	Instance/InsertableInstance/FamilyInstance	Planting	InsertableObject /FamilySymbol	Planting
Stairs	< Element >	Stairs	< Symbol >	Staies
Railing	< Element >	Railings	< Symbol >	Railings
Room	Room	Rooms	< null >	---
Room Tag	RoomTag	Room Tags	< Symbol >	Room Tags
Grid	Grid	Grids	LineAndTextAttrSymbol/GridType	< null >
Lines	ModelCurve/ModelLine	Lines	< null >	---
Ref Plane	ReferencePlane	Reference Planes	< null >	---
Dimension	Dimension	Dimensions	DimensionType	< null >
Section	< Element >	Views	< Symbol >	< null >
Text	TextElement/TextNote	Text Notes	LineAndTextAttrSymbol/TextElementType/TextNoteType	< null >
Level	Level	Levels	LevelType	Levels
Model Group	Group	Model Group	GroupType	Model Groups
Create.../Walls	Instance/InsertableInstance/FamilyInstance	Walls	InsertableObject /FamilySymbol	Walls

DB Element

Identifying Element

A system family is a built-in object in Revit. There is a designated class for it. You can use it to identify the element.

A component family has a generic form as FamilyInstance/FamilySymbol. Category is the way to further identify the kind of object it is representing in Revit.

Depending on an element you are interested in, you will need to check the following:

- Class name
- Category property
- If an element is Element Type (Symbol) or not

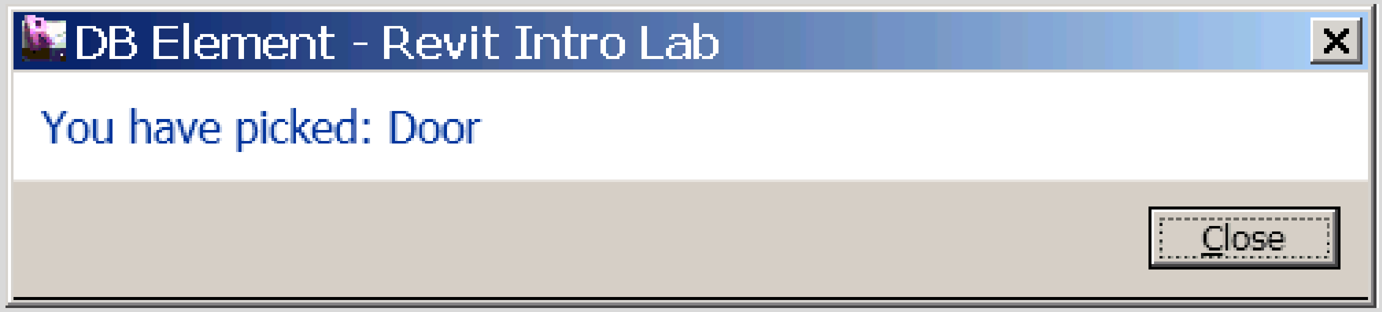
	System Family	**Component Family**
Family Type	WallType FloorType	FamilySymbol & Category - Doors, Windows
Instance	Wall Floor	FamilyInstance & Category - Doors, Windows

DB Element

Identifying Element

```
<VB.NET>
    ''  identify the type of the element known to the UI.
    Public Sub IdentifyElement(ByVal elem As Element)

        Dim s As String = ""
        If TypeOf elem Is Wall Then
            s = "Wall"
        ElseIf TypeOf elem Is Floor Then
            s = "Floor"
        ElseIf TypeOf elem Is RoofBase Then
            s = "Roof"
        ElseIf TypeOf elem Is FamilyInstance Then
            ''  An instance of a component family is all FamilyInstance.
            ''  We'll need to further check its category.
            If elem.Category.Id.IntegerValue = _
                BuiltInCategory.OST_Doors Then
                s = "Door"
            ElseIf elem.Category.Id.IntegerValue = _
                BuiltInCategory.OST_Windows Then
                s = "Window"
            ElseIf elem.Category.Id.IntegerValue = _
                BuiltInCategory.OST_Furniture Then
                s = "Furniture"
            Else
                s = "Component family instance"  '' e.g. Plant
            End If
     End sub
      ...
</VB.NET>
```


DB Element - Revit Intro Lab

You have picked: Door

DB Element

Element.Parameters

Parameters property of an Element class largely corresponds to an element or family “properties” in the UI.

In API, there are three ways to access those properties or parameters:

- Element.Parameters – returns a set of parameters applicable to the given element.
- Element.LookupParameter – takes an argument that can identify the kind of parameter and returns the value of single parameter.
- Element.get_Parameter

DB Element

Element.Parameters

```
<VB.NET>
    ''  show all the parameter values of the element
    Public Sub ShowParameters(ByVal elem As Element, _
                              Optional ByVal header As String = "")

        Dim s As String = header + vbCr + vbCr
        Dim params As ParameterSet = elem.Parameters

        For Each param As Parameter In params
            Dim name As String = param.Definition.Name
            ''  see the helper function below
            Dim val As String = ParameterToString(param)
            s = s + name + " = " + val + vbCr
        Next

        TaskDialog.Show("Revit Intro Lab", s)

    End Sub
</VB.NET>
```

```
<VB.NET>
    ''  Helper function: return a string from of a given parameter.
    Public Shared Function ParameterToString(ByVal param As Parameter) As String
        Dim val As String = "none"
        If param Is Nothing Then
            Return val
        End If
        ''  to get to the parameter value, we need to pause it depending on
        ''  its strage type
        Select Case param.StorageType
            Case StorageType.Double
                Dim dVal As Double = param.AsDouble
                val = dVal.ToString
            Case StorageType.Integer
                Dim iVal As Integer = param.AsInteger
                val = iVal.ToString()
            Case StorageType.String
                Dim sVal As String = param.AsString
                val = sVal
            Case StorageType.ElementId
                Dim idVal As ElementId = param.AsElementId
                val = idVal.IntegerValue.ToString
            Case StorageType.None
            Case Else
        End Select
        Return val
    End Function
</VB.NET>
```

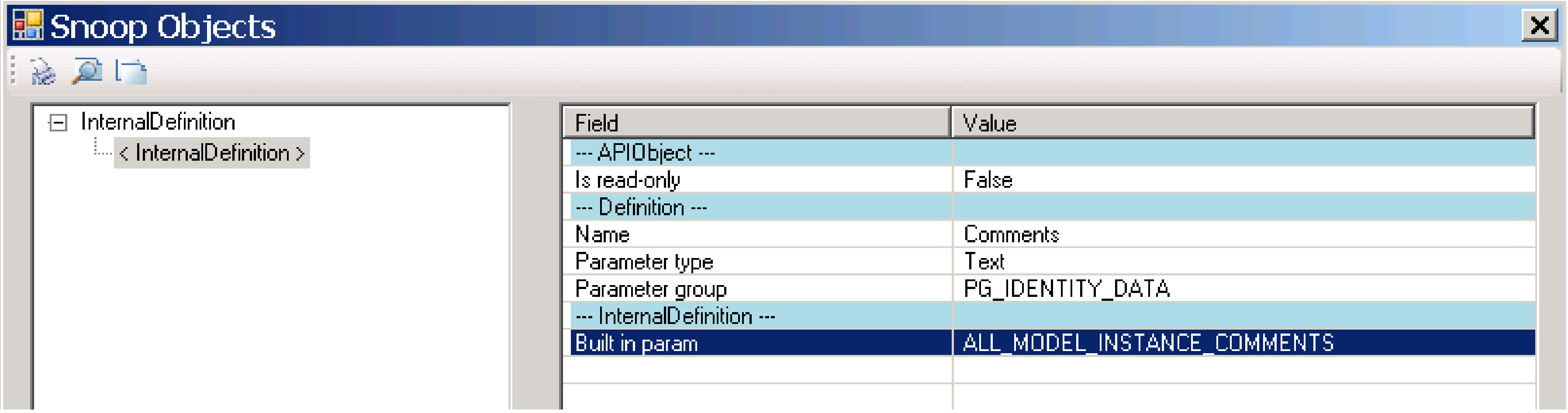
DB Element

Element.Parameter and Built-In Parameters

There are four ways to access individual parameters:

- Parameter(**BuiltInParameter**) – retrieve a parameter using the parameter Id.
- Parameter(String) – retrieve using the name.
- Parameter(Definition) – retrieve from its definition.
- Parameter(GUID) – retrieve shared parameter using GUID.

RevitLookup tool comes handy to explore and find out which BuiltInParameter corresponds to which parameter name



Field	Value
--- APIObject ---	
Is read-only	False
--- Definition ---	
Name	Comments
Parameter type	Text
Parameter group	PG_IDENTITY_DATA
--- InternalDefinition ---	
Built in param	ALL_MODEL_INSTANCE_COMMENTS

DB Element

Element.Parameter and Built-In Parameters

```
<VB.NET>
    ''  examples of retrieving a specific parameter individlly.
    Public Sub RetrieveParameter(ByVal elem As Element, _
                                 Optional ByVal header As String = "")

        Dim s As String = header + vbCr + vbCr
        ''  comments - most of instance has this parameter
        ''  (1) by name.  (Mark - most of instance has this parameter.)
        param = elem.Parameter("Mark")
        ...
        ''  (2) by BuiltInParameter.
        Dim param As Parameter = _
            elem.Parameter(BuiltInParameter.ALL_MODEL_INSTANCE_COMMENTS)
        ...
        ''  using the BuiltInParameter, you can sometimes access one
        ''  that is not in the parameters set.
        param = elem.Parameter(BuiltInParameter.SYMBOL_FAMILY_AND_TYPE_NAMES_PARAM)
        ...
        param = elem.Parameter(BuiltInParameter.SYMBOL_FAMILY_NAME_PARAM)
        ...
</VB.NET>
```

DB Element

Location

Location property

Location is further derived in two forms:

- LocationPoint - point-based location (e.g., furniture)
- LocationCurve – line-based location (e.g., wall)

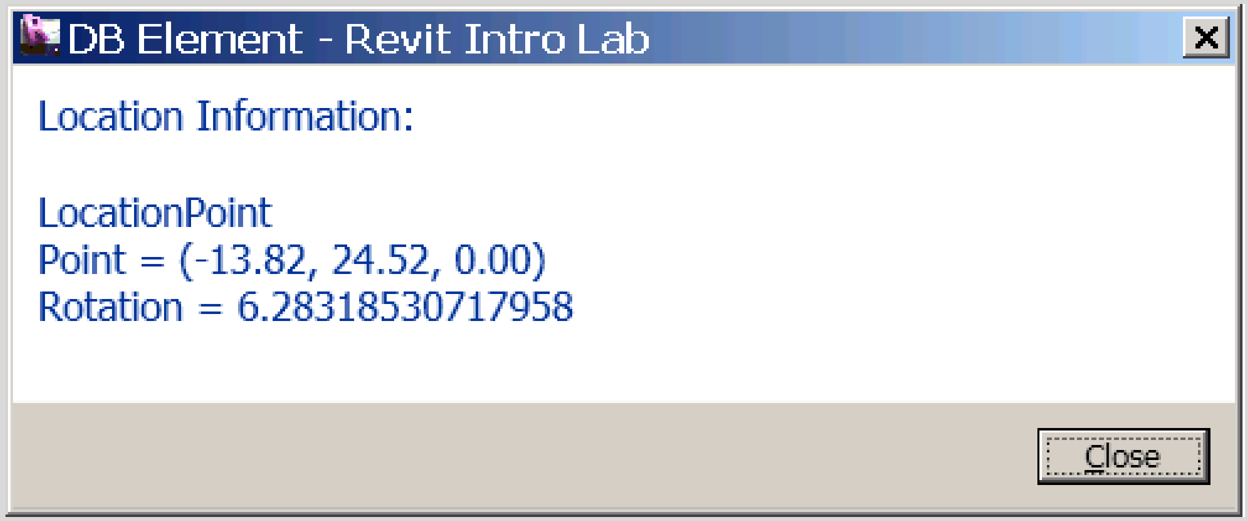
You will need to cast to LocationPoint or LocationCurve in order to access more properties.

<VB.NET>

```
    ''  show the location information of the given element.
    Public Sub ShowLocation(ByVal elem As Element)
        Dim s As String = "Location Information: " + vbCr + vbCr
        Dim loc As Location = elem.Location

        If TypeOf loc Is LocationPoint Then
            '' (1) we have a location point
            Dim locPoint As LocationPoint = loc
            Dim pt As XYZ = locPoint.Point
            Dim r As Double = locPoint.Rotation
            ...
        ElseIf TypeOf loc Is LocationCurve Then
            '' (2) we have a location curve
            Dim locCurve As LocationCurve = loc
            Dim crv As Curve = locCurve.Curve
            ...
            s = s + "EndPoint(0)/Start Point = " + PointToString(crv.EndPoint(0))
            s = s + "EndPoint(1)/End point = " + PointToString(crv.EndPoint(1))
            s = s + "Length = " + crv.Length.ToString + vbCr
        End If
        …
    End Sub
```

</VB.NET>



```
Location Information:

LocationPoint
Point = (-13.82, 24.52, 0.00)
Rotation = 6.28318530717958
```

DB Element

Geometry

Geometry Options – specify the detail level

Kinds of geometry objects

- Solid
- Geometry Instance (a instance of a symbol element, e.g. door or window)
- Curve
- Mesh

Further drill down into Solids/Faces/Edges - use RevitLookup

RevitCommands SDK sample has a simple example

SDK samples show geometry access with a viewer

- ElementViewer
- RoomViewer
- AnalyticalViewer

Further viewing options

- SVG Simple Vector Graphics, VRML Virtual Reality Markup Language, OpenGL, DirectX, many public domain viewers

Element Filtering

Element Iterations, Filtering and Queries

Element Filtering

Retrieving an Element

Elements in Revit are bundled in one single sack

To retrieve an element of interest, you filter for it

Typically, we would like to:

1. Retrieve a list of family types (e.g., wall types, door types)
2. Retrieve instances of a specific object class
   (e.g., all the walls, all the doors)
3. Find a specific family type with a given name
   (e.g., “Basic Wall: Generic – 200mm”, “M_Single-Flush: 0915 x 2134mm”)
4. Find specific instances (e.g., “Level 1” “View Plan 1”)

Similar to identifying element, you will need to consider a different approach depending on whether an element is a component-based or system-based.

FilteredElementCollector - documentation

Used to search, filter and iterate through a set of elements

Assign a variety of conditions to filter the elements which are returned.

Requires that at least one condition be set before making the attempt to access the elements, otherwise exception thrown.

Supports the IEnumerable interface

- Tip: because the ElementFilters and the shortcut methods offered by this class process elements in native code before their managed wrappers are generated, better performance will be obtained by using as many native filters as possible on the collector before attempting to process the results using LINQ queries

Filter types

Logical Filters – help to combine filter logic

- And
- Or

Quick filters - use an internal element record to determine passing state. This allows Revit to find elements which have not been expanded into internal memory yet.

- Examples:
  - ElementClassFilter
  - ElementCategoryFilter

Slow filters – not all information can be obtained by the element record, so these filters must expand to determine passing state.

- Examples:
  - FamilyInstanceFilter
  - AreaFilter

Efficiency guidelines

Filter quick aspects first

Filter slow second

After using built-in filtering techniques, consider LINQ to narrow down further.

Tip: Use the shortcut methods on FilteredElementCollector

- Because there are currently no shortcuts for Slow Filters, you can be sure when using a shortcut you are getting a Quick Filter.
- Examples:
    - OfClass
    - OfCategory

Logical Filters

Filter Name	Passing Criteria	Shortcut Methods
LogicalAndFilter	Where elements must pass 2 or more filters	WherePasses()- adds one additional filter IntersectWith() - joins two sets of independent filters
LogicalOrFilter	Where elements must pass at least one of 2 or more filters	UnionWith() - joins two sets of independent filters

Quick Filters

Name	Passing Criteria	Shortcut Methods
ElementCategoryFilter	Elements matching the input category id	OfCategoryId
ElementClassFilter	Elements matching the input runtime class	OfClass
ElementIsElementTypeFilter	Elements which are "Element types" (symbols)	WhereElementIsElementType WhereElementIsNotElementType
ElementOwnerViewFilter	Elements which are view-specific	OwnedByView WhereElementIsViewIndependent
ElementDesignOptionFilter	Elements in a particular design option	ContainedInDesignOption
ElementIsCurveDrivenFilter	Elements which are curve driven	WhereElementIsCurveDriven
ElementStructuralTypeFilter	Elements matching the given structural type	none
FamilySymbolFilter	Symbols of a particular family	none
ExclusionFilter	All elements except the element ids input to the filter	Excluding
BoundingBoxIntersectsFilter	Elements which have a bounding box which intersects a given outline.	none
BoundingBoxIsInsideFilter	Elements which have a bounding box inside a given outline	none
BoundingBoxContainsPointFilter	Elements which have a bounding box that contain a given point	none

Slow Filters

Name	Passing Criteria	Shortcut Methods
FamilyInstanceFilter	Instances of a particular family symbol	none
ElementLevelFilter	Elements associated to a given level id	none
ElementParameterFilter	Parameter existence, value matching, range matching, and/or string matching	none
PrimaryDesignOptionMemberFilter	Elements owned by any primary design option.	none
StructuralInstanceUsageFilter	Structural usage parameter for FamilyInstances	none
StructuralWallUsageFilter	Structural usage parameter for Walls	none
StructuralMaterialTypeFilter	Material type applied to FamilyInstances	none
RoomFilter	Finds rooms	none
SpaceFilter	Finds spaces	none
AreaFilter	Finds areas	none
RoomTagFilter	Finds room tags	none
SpaceTagFilter	Finds space tags	none
AreaTagFilter	Finds area tags	none
CurveElementFilter	Finds specific types of curve elements (model curves, symbolic curves, detail curves, etc)	none

Element Filtering

1.1 A List of Family Types - System Family

Collect all the wall types (2nd and 3rd using shortcuts)

```
<VB.NET>
        Dim wallTypeCollector1 = new FilteredElementCollector(m_rvtDoc)
        wallTypeCollector1.WherePasses(New ElementClassFilter(GetType(WallType)))
        Dim wallTypes1 As IList(Of Element) = wallTypeCollector1.ToElements
</VB.NET>
```

```
<VB.NET>
        Dim wallTypeCollector2 = new FilteredElementCollector(m_rvtDoc)
        wallTypeCollector2.OfClass(GetType(WallType))
</VB.NET>
```

```
<VB.NET>
       Dim wallTypeCollector3 = _
          new FilteredElementCollector(m_rvtDoc).OfClass(GetType(WallType))
</VB.NET>
```

Element Filtering

1.2 A List of Family Types - Component Family

Collect all the door types

```
<VB.NET>
        Dim doorTypeCollector = new FilteredElementCollector(m_rvtDoc)
        doorTypeCollector.OfClass(GetType(FamilySymbol))
        doorTypeCollector.OfCategory(BuiltInCategory.OST_Doors)
        Dim doorTypes As IList(Of Element) = doorTypeCollector.ToElements
</VB.NET>
```

```
<VB.NET>
 Dim doorTypes As IList(Of Element) _
   = new FilteredElementCollector(m_rvtDoc) _
     .OfClass(GetType(FamilySymbol)) _
     .OfCategory(BuiltInCategory.OST_Doors) _
     .ToElements
</VB.NET>
```

Element Filtering

2.1 List of Instances of a Specific Object Class - System Family

Collect all the instances of wall

```
<VB.NET>
    Dim wallCollector = New FilteredElementCollector(m_rvtDoc).OfClass(GetType(Wall))
    Dim wallList As IList(Of Element) = wallCollector.ToElements
</VB.NET>
```

Element Filtering

2.2 List of Instances of a Specific Object Class - Component Family

Collect all the instances of door

```
<VB.NET>
    Dim doorCollector = New FilteredElementCollector(m_rvtDoc). _
        OfClass(GetType(FamilyInstance))
    doorCollector.OfCategory(BuiltInCategory.OST_Doors)
    Dim doorList As IList(Of Element) = doorCollector.ToElements
</VB.NET>
```

Element Filtering

3.1 Find a Specific Family Type – System Family Type

Find a wall type e.g., “Basic Wall: Generic – 200mm”

```
<VB.NET>
    Function FindFamilyType_Wall_v1(ByVal wallFamilyName As String, _
         ByVal wallTypeName As String) As Element
        ''  narrow down a collector with class.
        Dim wallTypeCollector1 = New FilteredElementCollector(m_rvtDoc)
        wallTypeCollector1.OfClass(GetType(WallType))
        ''  LINQ query
        Dim wallTypeElems1 = _
            From element In wallTypeCollector1 _
            Where element.Name.Equals(wallTypeName) _
            Select element
        ''  get the result.
        Dim wallType1 As Element = Nothing '' result will go here.
        If wallTypeElems1.Count > 0 Then
            wallType1 = wallTypeElems1.First
        End If
        Return wallType1
    End Function
</VB.NET>
```

Element Filtering

3.2 Find a Specific Family Type – Component Family

- Find a door type, e.g., “M_Single-Flush: 0915 x 2134”

```
Function FindFamilyType_Door_v1(ByVal doorFamilyName As String, ByVal doorTypeName As
String) As Element
        ''  narrow down the collection with class and category.
        Dim doorFamilyCollector1 = New FilteredElementCollector(m_rvtDoc)
        doorFamilyCollector1.OfClass(GetType(FamilySymbol))
        doorFamilyCollector1.OfCategory(BuiltInCategory.OST_Doors)
        ''  parse the collection for the given name using LINQ query here.
        Dim doorTypeElems = _
            From element In doorFamilyCollector1 _
            Where element.Name.Equals(doorTypeName) And _
            element.Parameter(BuiltInParameter.SYMBOL_FAMILY_NAME_PARAM). _
            AsString.Equals(doorFamilyName) _
            Select element
        ''  get the result.
        Dim doorType1 As Element = Nothing
        Dim doorTypeList As IList(Of Element) = doorTypeElems.ToList()
        If doorTypeList.Count > 0 Then '' we should have only one.
            doorType1 = doorTypeList(0) ' found it.
        End If
        Return doorType1
End Function
```

Element Filtering

4.1 Find Instances of a Given Family Type

- Find doors with a given type e.g.,“M_Single-Flush: 0915 x 2134”

<VB.NET>

```
''  Find a list of element with the given Class, family type and Category (optional).
Function FindInstancesOfType(ByVal targetType As Type, _
     ByVal idFamilyType As ElementId, _
     Optional ByVal targetCategory As BuiltInCategory = Nothing) As IList(Of Element)
   ''  narrow down to the elements of the given type and category
   Dim collector = New FilteredElementCollector(m_rvtDoc).OfClass(targetType)
   If Not (targetCategory = Nothing) Then
        collector.OfCategory(targetCategory)
   End If
   ''  parse the collection for the given family type id. using LINQ query here.
   Dim elems = _
        From element In collector _
        Where element.Parameter(BuiltInParameter.SYMBOL_ID_PARAM). _
              AsElementId.Equals(idType) _
        Select element
   ''  put the result as a list of element for accessibility.
   Return elems.ToList()
End Function
```

</VB.NET>

Element Filtering

4.2 Find Elements with a Given Class and Name

- Find elements with a given name and type

```
<VB.NET>
''  Find a list of elements with given class, name, category (optional).
Public Shared Function FindElements(ByVal rvtDoc As Document, _
    ByVal targetType As Type, ByVal targetName As String, _
    Optional ByVal targetCategory As BuiltInCategory = Nothing) As IList(Of Element)
    ''  narrow down to the elements of the given type and category
    Dim collector = _
            New FilteredElementCollector(rvtDoc).OfClass(targetType)
    If Not (targetCategory = Nothing) Then
            collector.OfCategory(targetCategory)
    End If
    ''  parse the collection for the given names. using LINQ query here.
    Dim elems = _
        From element In collector _
        Where element.Name.Equals(targetName) _
        Select element
    ''  put the result as a list of element for accessibility.
    Return elems.ToList()
End Function
</VB.NET>
```

Element Filtering

More Options

We have learnt how to use the following classes:

- FilteredElementCollector
- ElementClassFilter
- ElemetCategoryFilter

There are more different kinds of filters, such as:

- BoundingBoxContainsPointFilter
- ElementDesignOptionFilter
- ElementIsCurveDrivenFilter
- ElementIsElementTypeFilter
- ElementParameterFilter
- …

cf. Online Developer Guide.

Element Modification

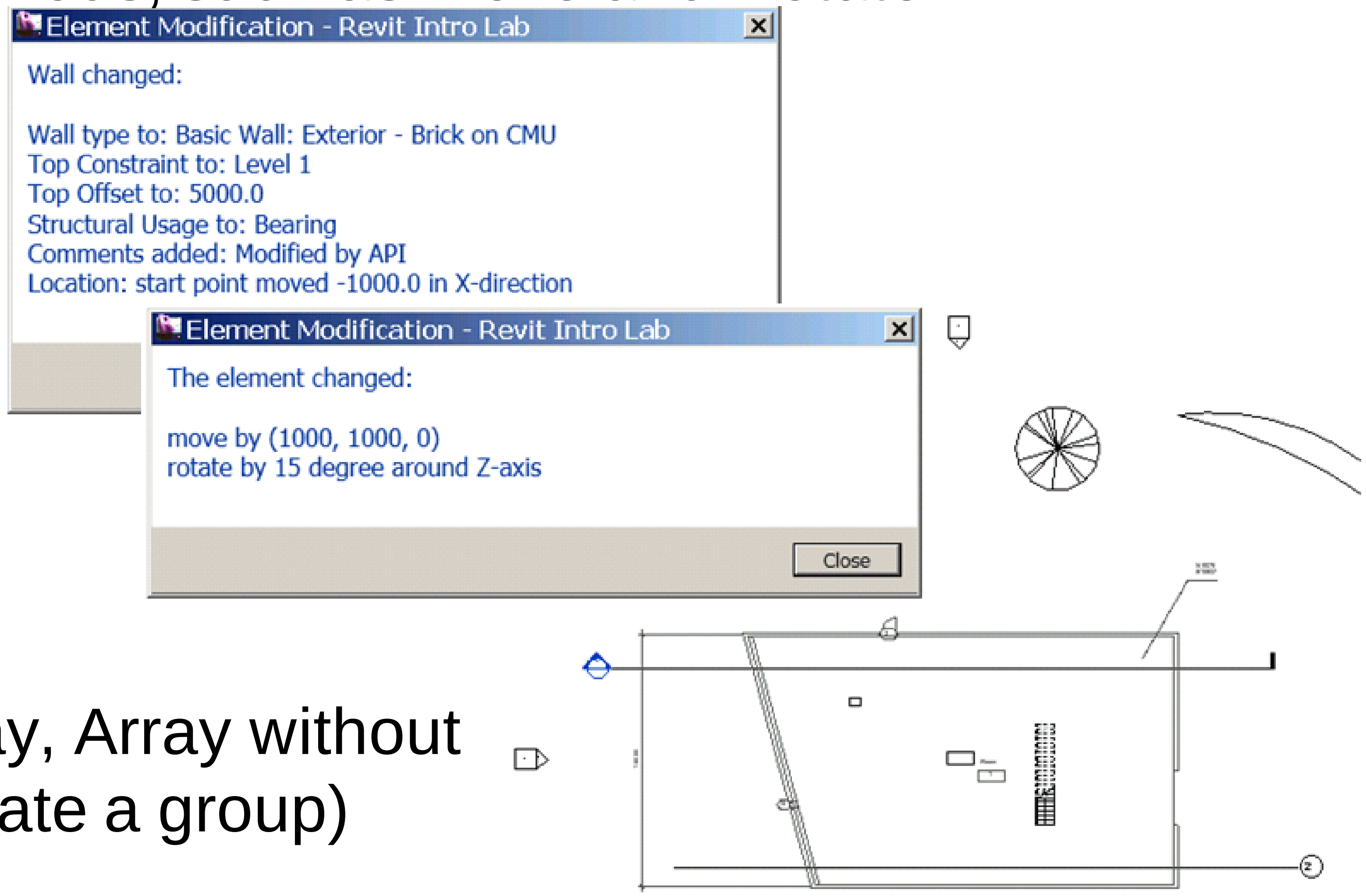
How to modify an element

Element Modification

Element Level vs. Document Level Modification

Two approaches to modify an element:

- by changing its properties, parameters and location at each element level
- using Document level methods, such as Move and Rotate



Element Modification - Revit Intro Lab

Wall changed:

Wall type to: Basic Wall: Exterior - Brick on CMU
Top Constraint to: Level 1
Top Offset to: 5000.0
Structural Usage to: Bearing
Comments added: Modified by API
Location: start point moved -1000.0 in X-direction

Element Modification - Revit Intro Lab

The element changed:

move by (1000, 1000, 0)
rotate by 15 degree around Z-axis

At each element level, you can change:

- Family type
- Parameters
- Location

By Document methods:

- Move, Rotate, Mirror, Array, Array without associate (this will not create a group)

Element Modification

Element Level – Family Type

Change the family type of an instance (e.g., a wall and a door)

```
<VB.NET>
    ''  e.g., an element we are given is a wall.
    Dim aWall As Wall = elem
    ''  find a wall family type with the given name.
    Dim newWallType As Element = ElementFiltering.FindFamilyType( m_rvtDoc, _
         GetType(WallType), "Basic Wall", "Exterior - Brick on CMU")
    ''  assign a new family type.
    aWall.WallType = newWallType
</VB.NET>
```

```
<VB.NET>
    ''  e.g., an element we are given is a door.
    Dim aDoor As FamilyInstance = elem
    ''  find a door family type with the given name.
    Dim newDoorType As Element = ElementFiltering.FindFamilyType( _
         GetType(FamilySymbol), "M_Single-Flush", "0762 x 2032mm", _
         BuiltInCategory.OST_Doors)
    ''  assign a new family type.
    aDoor.Symbol = newDoorType
</VB.NET>
```

```
<VB.NET>
    ''  or use a general way: ChangeTypeId
     aDoor.ChangeTypeId(newDoorType.Id)
</VB.NET>
```

Element Modification

Element Level – Parameter

Change a parameter of an element (e.g., a wall and a door)

```
<VB.NET>
aWall.Parameter(BuiltInParameter.WALL_TOP_OFFSET).Set(14.0)
aWall.Parameter(BuiltInParameter.ALL_MODEL_INSTANCE_COMMENTS).Set( _
     "Modified by API")
</VB.NET>
```

Element Modification

Element Level – Location Curve

Change a value of location information (e.g., a wall)

```
<VB.NET>
    Dim wallLocation As LocationCurve = aWall.Location
    ''  create a new line bound.
    Dim newPt1 = New XYZ(0.0, 0.0, 0.0)
    Dim newPt2 = New XYZ(20.0, 0.0, 0.0)
    Dim newWallLine As Line = Line.CreateBound(newPt1, newPt2)
    ''  change the curve.
    wallLocation.Curve = newWallLine
</VB.NET>
```

Element Modification

Document Level – Move and Rotate

Move and rotate an element (e.g., a wall)

```
<VB.NET>
        ''  move by displacement
        Dim v As XYZ = New XYZ(10.0, 10.0, 0.0)

        ElementTransformUtils.MoveElement(doc, elem.Id, v)
</VB.NET>
```

```
<VB.NET>
        ''  rotate by 15 degree around z-axis.
        Dim pt1 = XYZ.Zero
        Dim pt2 = XYZ.BasisZ
        Dim axis As Line = Line.CreateBound(pt1, pt2)
ElementTransformUtils.RotateElement(doc, elem.Id, axis, Math.PI / 12.0)
</VB.NET>
```

Element Modification

Regeneration of Graphics

When you modify an element that results in changes in a model geometry and you need to access to the updated geometry, the graphics need to be regenerated.

You can control this by calling Document.Regenerate()

```
m_rvtDoc.Regenerate()
```

Model Creation

How to create instances of Revit elements

Model Creation

Create Instances of Revit Elements

Create a new geometry element:

Application.Create.NewXxx() e.g., NewBoundingBoxXYZ()

Create a new model element:

Document.Create.NewXxx() e.g., NewFamilyInstance()

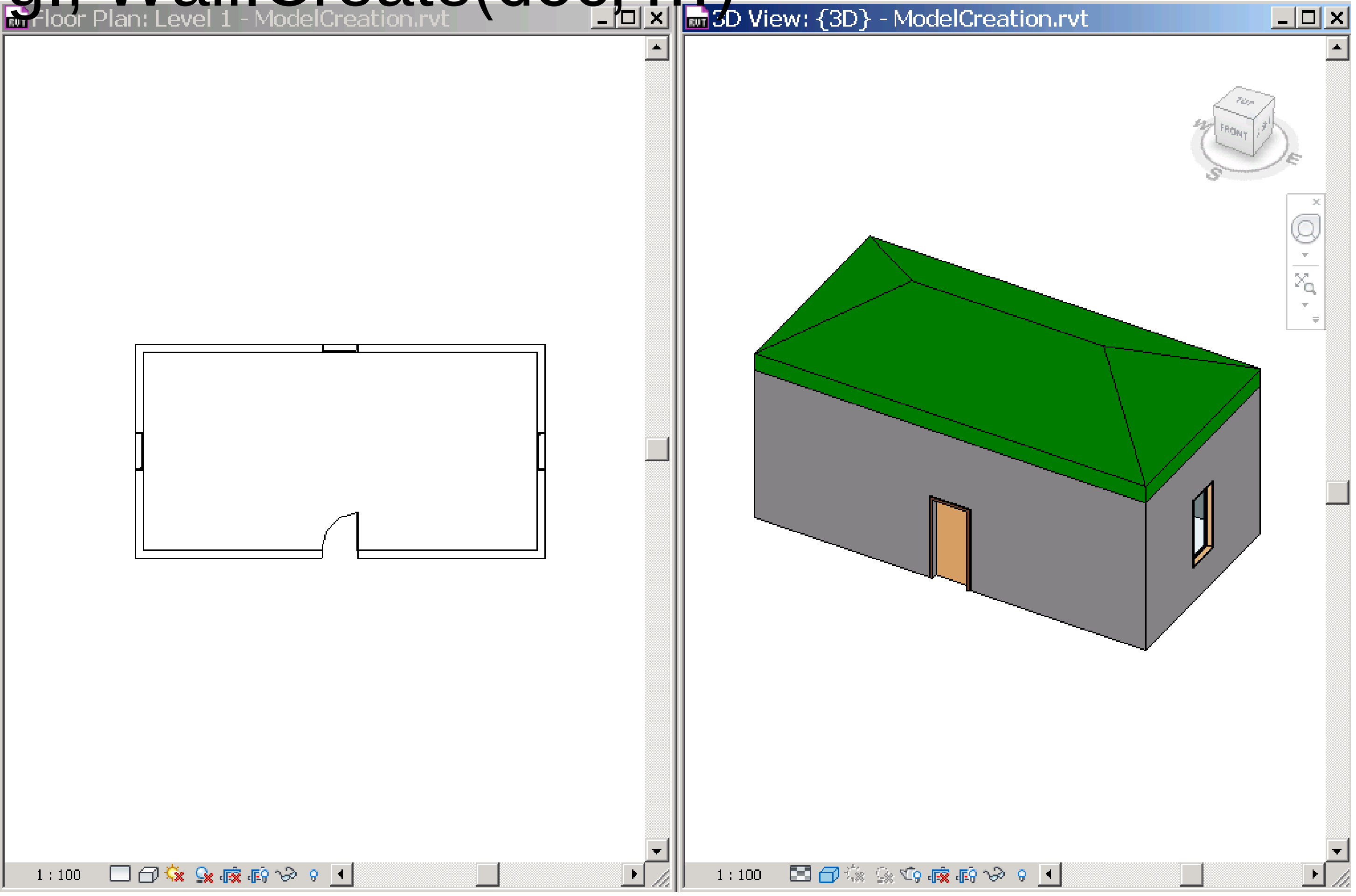
Use static Create methods e.g., Wall.Create(doc, ...)

Multiple overloaded methods, each for a specific condition and/or apply only certain types of elements.

e.g., 5 Wall.Create(),

9 NewFamilyInstance()

cf. Dev Guide


Floor Plan: Level 1 - ModelCreation.rvt
1 : 100
3D View: {3D} - ModelCreation.rvt
1 : 100

Model Creation

New Walls

```
<VB.NET>
'' create walls
Sub CreateWalls()
    '' get the levels we want to work on.
    Dim level1 As Level = ElementFiltering.FindElement(m_rvtDoc, GetType(Level), "Level 1")
    Dim level2 As Level = ElementFiltering.FindElement(m_rvtDoc, GetType(Level), "Level 2")

    '' set four corner of walls.
    Dim pts As New List(Of XYZ)(5)
    ...

    Dim isStructural As Boolean = False '' flag for structural wall or not.

    '' loop through list of points and define four walls.
    For i As Integer = 0 To 3
        '' define a base curve from two points.
        Dim baseCurve As Line = Line.CreateBound(pts(i), pts(i + 1))
        '' create a wall using the one of overloaded methods.
        Dim aWall As Wall = Wall.Create(m_rvtDoc, baseCurve, level1, isStructural)
        '' set the Top Constraint to Level 2
        aWall.Parameter(BuiltInParameter.WALL_HEIGHT_TYPE).Set(level2.Id)
    Next
    '' This is important. we need these lines to have shrinkwrap working.
    m_rvtDoc.Regenerate()
    m_rvtDoc.AutoJoinElements()
End Sub
</VB.NET>
```

Model Creation

A New Door

```
<VB.NET>
  ''  add a door to the center of the given wall.
  Sub AddDoor(ByVal hostWall As Wall)

    ''  get the door type to use.
    Dim doorType As FamilySymbol = _
     ElementFiltering.FindFamilyType(m_rvtDoc, GetType(FamilySymbol), _
     "M_Single-Flush", "0915 x 2134mm", BuiltInCategory.OST_Doors)

    ''  get the start and end points of the wall.
    Dim locCurve As LocationCurve = hostWall.Location
    Dim pt1 As XYZ = locCurve.Curve.GetEndPoint(0)
    Dim pt2 As XYZ = locCurve.Curve.GetEndPoint(1)
    ''  calculate the mid point.
    Dim pt As XYZ = (pt1 + pt2) / 2.0

    ''  we want to set the reference as a bottom of the wall or level1.
    Dim idLevel1 As ElementId = _
    hostWall.Parameter(BuiltInParameter.WALL_BASE_CONSTRAINT).AsElementId
    Dim level1 As Level = m_rvtDoc.Element(idLevel1)

    ''  finally, create a door.
    Dim aDoor As FamilyInstance = m_rvtDoc.Create.NewFamilyInstance( _
    pt, doorType, hostWall, level1, StructuralType.NonStructural)
  End Sub
</VB.NET>
```

Revit API Intro Labs

Revit API fundamentals

- Revit Add-ins: external command/application, attributes, add-in manifest and object model
- Representation of Revit elements
- Element iteration, filtering and queries
- Element modification
- Model creation

Exercises

- Lab1 – “Hello World”
- Lab2 – DB element
- Lab3 – element filtering
- Lab4 – element modification
- Lab5 – model creation

Addendum: Additional Lab Exercises

If interested, work on additional labs:

- **Extensible Storage Lab** – Learn to add custom data to Revit element
- **Shared Parameter Lab** – Learn to create shared parameters

Thank you!

AUTODESK®

Autodesk is a registered trademark of Autodesk, Inc., and/or its subsidiaries and/or affiliates in the USA and/or other countries. All other brand names, product names, or trademarks belong to their respective holders. Autodesk reserves the right to alter product and services offerings, and specifications and pricing at any time without notice, and is not responsible for typographical or graphical errors that may appear in this document.