Wind Turbine design based on QuietRevolution.
Model created by Spine 3D

Introduction to Revit Programming

Revit UI API

Developer Technical Services

AUTODESK.

Agenda

UI Topics

- Ribbon
- User Selection
- Task dialog
- Events
- Dynamic model update

Ribbon API

How to add your own Ribbon buttons

Ribbon API
Overview

- The Ribbon API is the only GUI customization API
  - Menus and toolbars need to be migrated to ribbon
- Easy to use
- No WPF knowledge needed
- Guidelines provided
  - Ribbon design guidelines.pdf
  - Autodesk Icon Guidelines.pdf

Ribbon API Overview

Custom ribbon panels are by default added to the Add-Ins tab

Custom ribbon panels can also be placed on the Analyze tab

Custom ribbon tabs can be created (since Revit 2012, max. 20)

External commands are placed under Add-Ins > External Tools

External applications can use custom ribbon panel or tab

- Push button
- Pull-down button
- Single or stacked layout with two or three rows
- Split button
- Radio Group
- Combo box
- Text box
- Slide-Out panel

Ribbon API Classes

RibbonPanel
- A panel containing ribbon items or buttons

RibbonItem
- A button, push or pull-down, ComboBox, TextBox, RadioButton, etc.

PushButton, PushButtonData
- Manage push button information

PulldownButton, PulldownButtonData
- Manage pull-down button information

SplitButton, SplitButtonData
- Manage split button information

ComboBox, ComboBoxData
- Manage combo box information

…

Ribbon API Since Revit 2011~

Namespace

Autodesk.Revit.UI

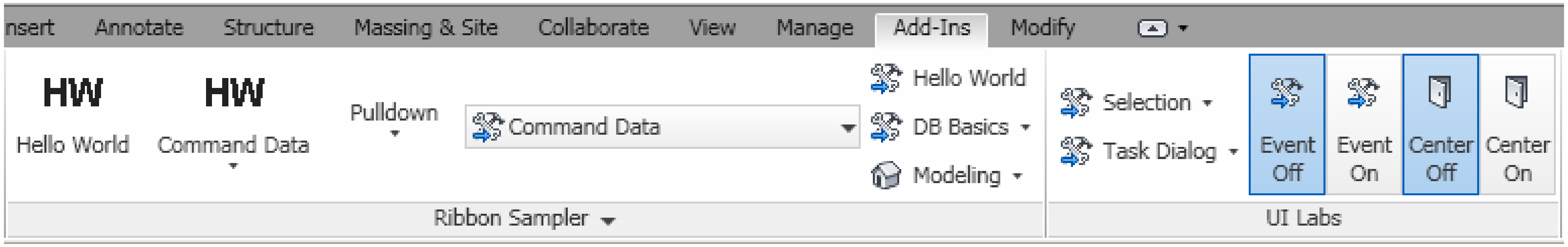
Widgets (SplitButton, ComboBox, TextBox, etc)

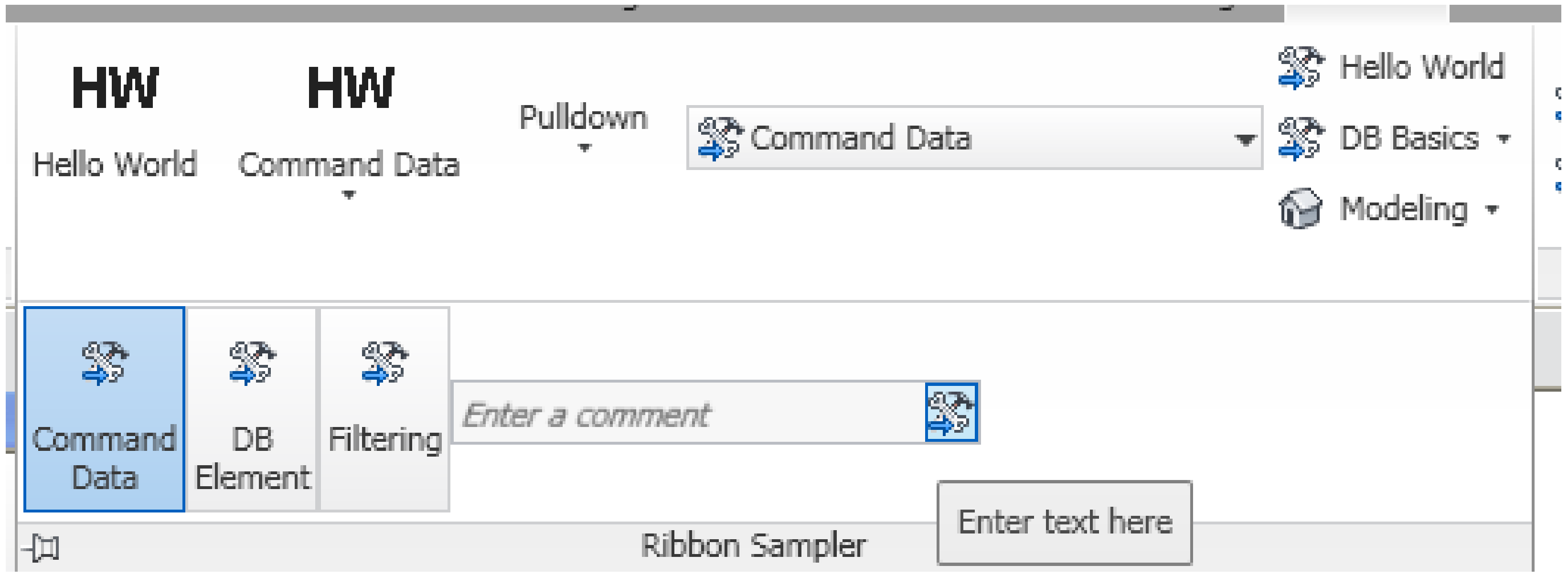
- Events for ComboBox and TextBox

Properties

- RibbonItem.Visible
- RibbonItem.LongDescription
- RibbonItem.ToolTipImage
- PushButton.AvailabilityClassName

Lab - Ribbon API





User Selection

Point and object(s) selection using the API

User Selection

Overview

Ability to select Object(s), Point, Edge and Face

Add new selection to active collection using:

- PickObject()
- PickObjects()

```
UIDocument uidoc = new UIDocument(document);
Selection choices = uidoc.Selection;

// Choose objects from Revit.

IList<Element> hasPickSome =
    choices.PickElementsByRectangle("Select by rectangle");

if (hasPickSome.Count > 0)
{
    int newSelectionCount = choices.Elements.Size;
    string prompt = string.Format("{0} elements added to Selection.",
        newSelectionCount - selectionCount);
    TaskDialog.Show("Revit", prompt);
}
```

User Selection

Overview

Ability to specify type of object

- Element, PointOnElement, Edge, Face

Ability to add custom status messages

- StatusbarTip

```
public void PickPoint(UIDocument uidoc)
{
     ObjectSnapTypes snapTypes =
         ObjectSnapTypes.Endpoints | ObjectSnapTypes.Intersections;

     XYZ point = uidoc.Selection.PickPoint(
         snapTypes, "Select an end point or intersection");

     string strCoords = "Selected point is " + point.ToString();
     TaskDialog.Show("Revit", strCoords);
}
```

Ability to set the active workplane

- View.SketchPlane

User Selection

Selection Filter

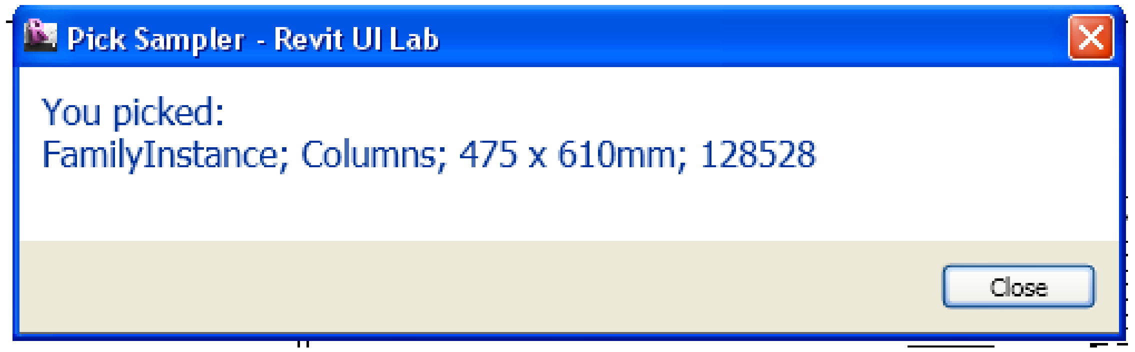
ISelection Interface to help filter objects during selection

- AllowElement()
- AllowReference()

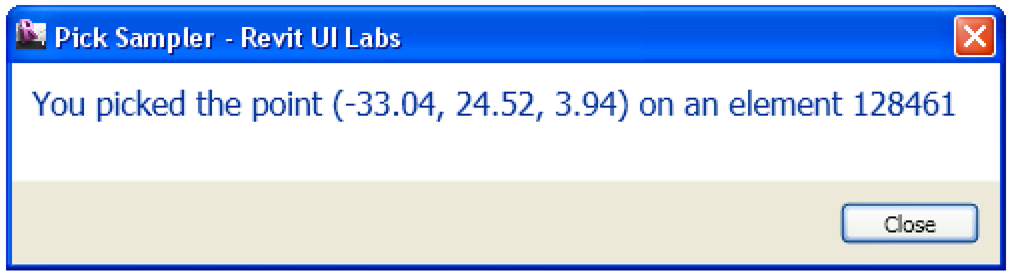
```
public void SelectPlanarFaces(Autodesk.Revit.DB.Document document)
{
        UIDocument uidoc = new UIDocument(document);
        ISelectionFilter selFilter = new PlanarFacesSelectionFilter();
        IList<Reference> faces = uidoc.Selection.PickObjects(
            ObjectType.Face, selFilter, "Select multiple planar faces");
}

public class PlanarFacesSelectionFilter : ISelectionFilter
{
        public bool AllowElement(Element element)
        {
           return true;
        }

        public bool AllowReference(Reference refer, XYZ point)
        {
           if (refer.GeometryObject is PlanarFace)  { return true; }
           return false;
        }
}
```

Lab - User Selection

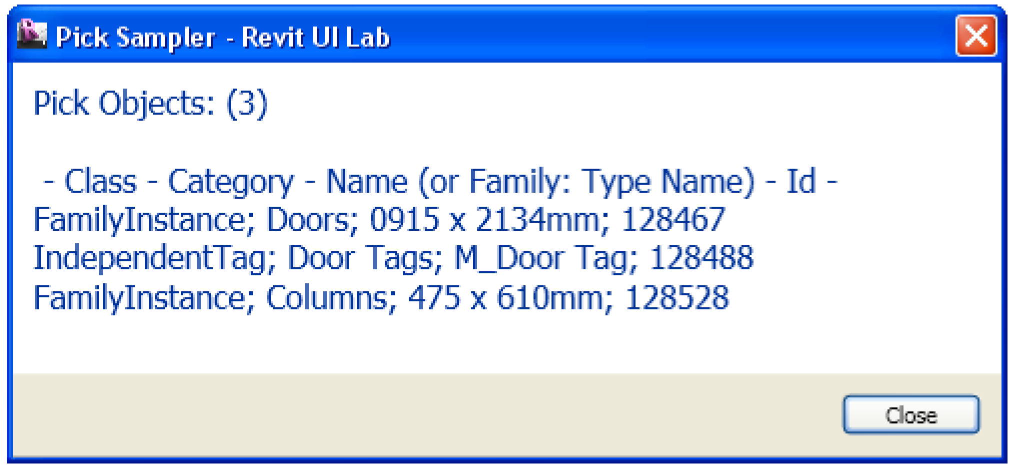
Pick Sampler



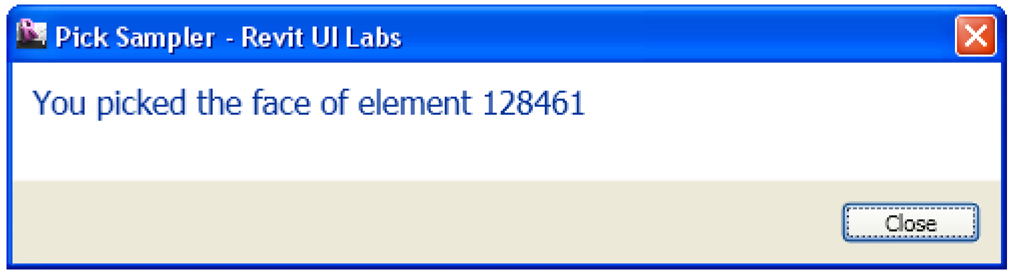
```
You picked:
FamilyInstance; Columns; 475 x 610mm; 128528
```



```
You picked the point (-33.04, 24.52, 3.94) on an element 128461
```



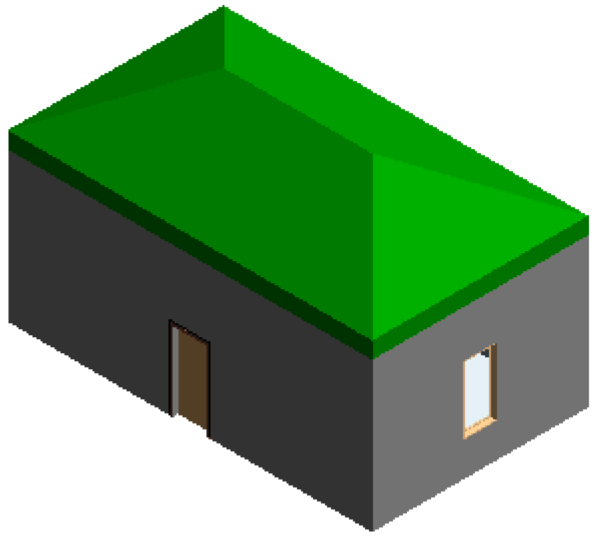
```
Pick Objects: (3)

 - Class - Category - Name (or Family: Type Name) - Id -
FamilyInstance; Doors; 0915 x 2134mm; 128467
IndependentTag; Door Tags; M_Door Tag; 128488
FamilyInstance; Columns; 475 x 610mm; 128528
```



```
You picked the face of element 128461
```

Lab - User Selection

Create House Pick



```
<CS>
    XYZ pt1 = rvtUIDoc.Selection.PickPoint("Pick the first corner of walls");
    XYZ pt2 = rvtUIDoc.Selection.PickPoint("Pick the second corner");

    // simply create four walls with orthogonal rectangular profile
    // from the two points picked.
    List<Wall> walls = RevitIntroVB.ModelCreation.CreateWalls(
        rvtUIDoc.Document, pt1, pt2);

    // pick a wall to add a front door
    SelectionFilterWall selFilterWall = new SelectionFilterWall();
    Reference @ref = rvtUIDoc.Selection.PickObject(
        ObjectType.Element, selFilterWall, "Select a wall to place a front door");
    Wall wallFront = @ref.Element as Wall;

    // add a door to the selected wall
    RevitIntroVB.ModelCreation.AddDoor(rvtUIDoc.Document, wallFront);
</CS>
```

Task Dialogs

Revit styled message boxes

Task Dialogs

Overview

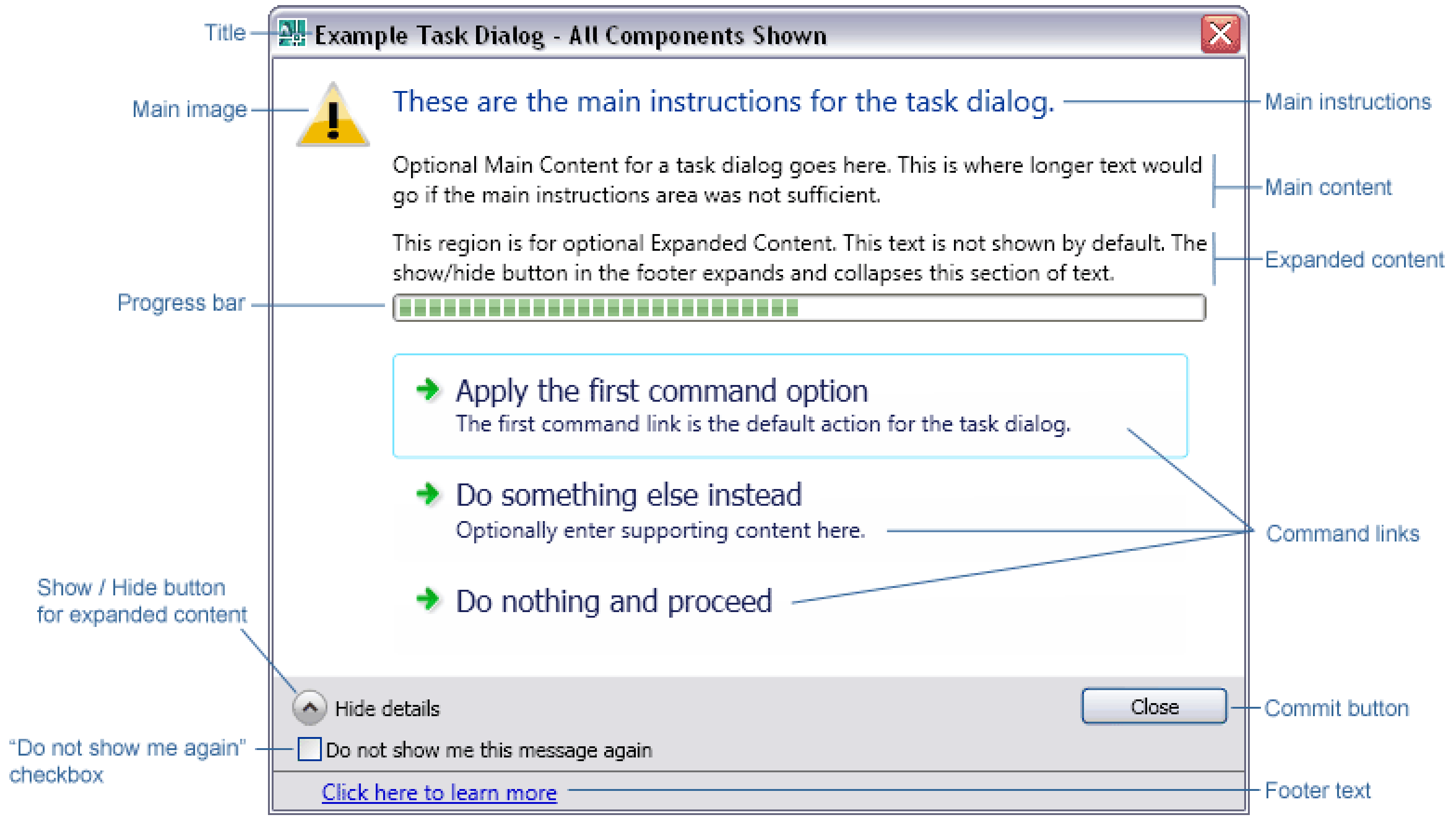
A modal dialog with set of controls

Revit style alternative to simple Windows message box.

Used when system needs to

- Provide information
- Ask a question
- Allow users to select options to perform task

*) progress bar is not available



Example Task Dialog - All Components Shown

These are the main instructions for the task dialog.

Optional Main Content for a task dialog goes here. This is where longer text would go if the main instructions area was not sufficient.

This region is for optional Expanded Content. This text is not shown by default. The show/hide button in the footer expands and collapses this section of text.

Apply the first command option
The first command link is the default action for the task dialog.

Do something else instead
Optionally enter supporting content here.

Do nothing and proceed

Hide details

Close

Do not show me this message again

Click here to learn more

Task Dialog

Overview

Two ways to create task dialogs:

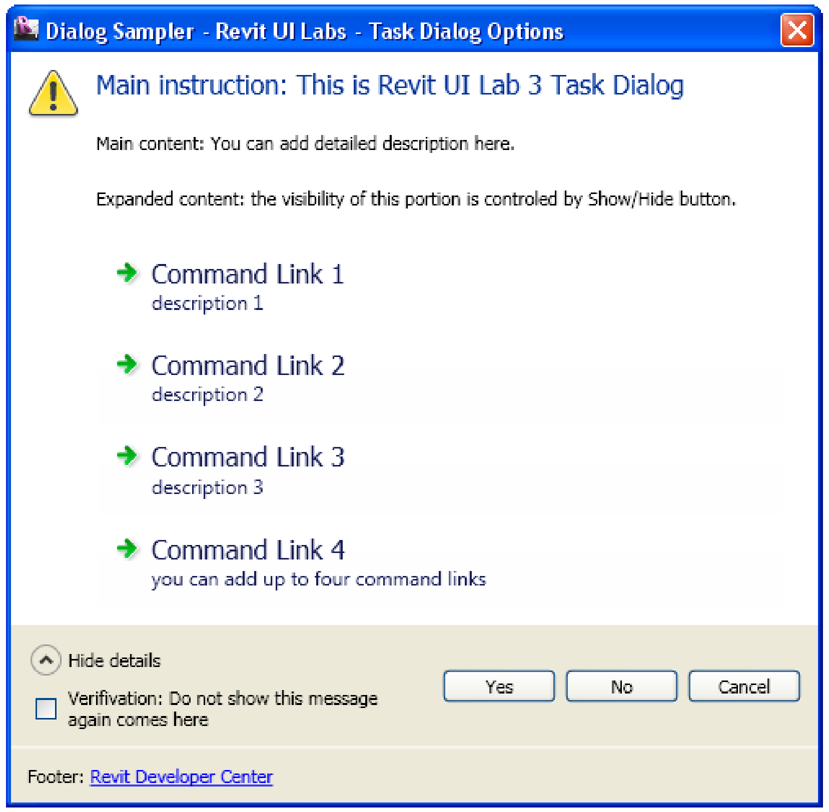
- Construct TaskDialog, set properties and use instance method Show()
  - Instance of Autodesk.Revit.UI.**TaskDialog**
- Use one of the static Show() methods to show in one step

And use it to set

- instructions
- detailed text
- icons
- buttons
- command links
- verification text, etc

Lab - Task Dialog

Dialog Sampler



Dialog Sampler - Revit UI Labs - Task Dialog Options

Main instruction: This is Revit UI Lab 3 Task Dialog

Main content: You can add detailed description here.

Expanded content: the visibility of this portion is controled by Show/Hide button.

Command Link 1
description 1

Command Link 2
description 2

Command Link 3
description 3

Command Link 4
you can add up to four command links

Hide details

Verifivation: Do not show this message again comes here

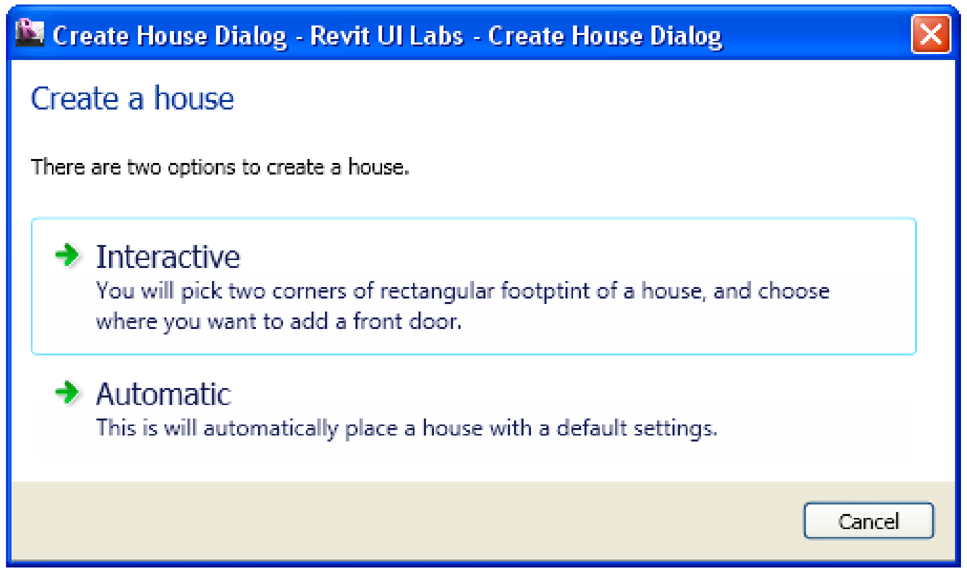
Yes No Cancel

Footer: Revit Developer Center

```
<CS>
    // (0) create an instance of task dialog to set more options.
    TaskDialog myDialog = new TaskDialog("Revit UI Labs - Task Dialog Options");

    // (1) set the main area. these appear at the upper portion of the dialog.

    myDialog.MainIcon = TaskDialogIcon.TaskDialogIconWarning;
    // or TaskDialogIcon.TaskDialogIconNone.
    myDialog.MainInstruction =
        "Main instruction: This is Revit UI Lab 3 Task Dialog";
    myDialog.MainContent = "Main content: You can add detailed description here.";

    if (stepByStep) myDialog.Show();
</CS>
```

Lab - Task Dialog

Create House Dialog



Create House Dialog - Revit UI Labs - Create House Dialog

Create a house

There are two options to create a house.

Interactive
You will pick two corners of rectangular footptint of a house, and choose where you want to add a front door.

Automatic
This is will automatically place a house with a default settings.

```
<CS>
    TaskDialog houseDialog = new TaskDialog("Revit UI Labs - Create House Dialog");
    houseDialog.MainInstruction = "Create a house";
    houseDialog.MainContent = "There are two options to create a house.";
    houseDialog.AddCommandLink(TaskDialogCommandLinkId.CommandLink1, "Interactive",
"You will pick two corners of rectangular footprint of a house, and choose where you
want to add a front door.");
    houseDialog.AddCommandLink(TaskDialogCommandLinkId.CommandLink2, "Automatic",
"This is will automatically place a house with a default settings.");
    houseDialog.CommonButtons = TaskDialogCommonButtons.Cancel;
    houseDialog.DefaultButton = TaskDialogResult.CommandLink1;

    // show the dialog to the user.
    TaskDialogResult res = houseDialog.Show();
</CS>
```

Events and Dynamic Model Update

Application, Document and Element events

Events

Overview

Notifications triggered on specific actions

Compliant to .NET event standard

- Pre and Post events
- Single event (DocumentChanged and FailureProcessing)

Types :

- Application level
- Document level
- Element level

Events

Overview

Also Classified as DB and UI events

- DB events available from Application and Document classes
- UI events available from UIApplication class

Edit model during events using

- Document.IsModifiable
- Document.IsReadOnly

Many of the new pre-events are cancellable

- RevitEventArgs.Cancellable
- RevitAPIPreEventArgs.Cancel

Events

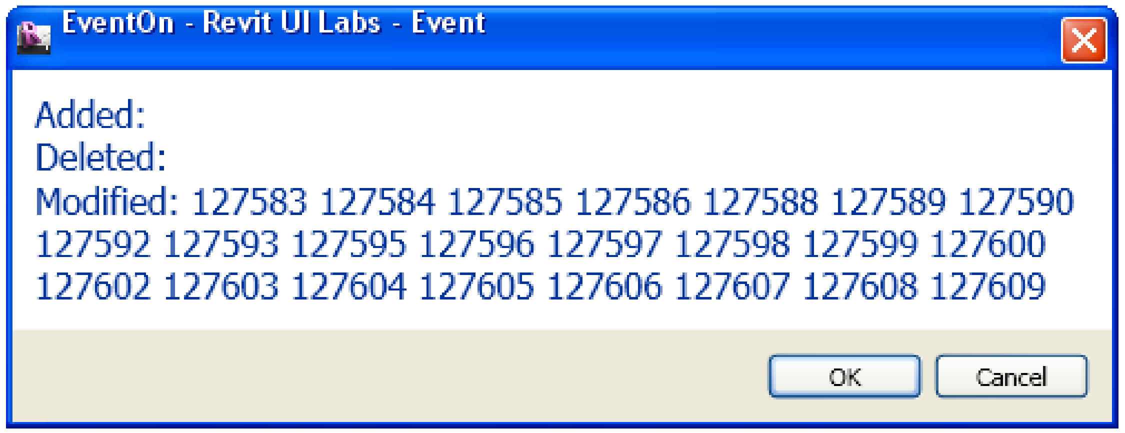
Event Handler, Registering and Unregistering events

```
public void UILabs_DocumentChanged(object sender, DocumentChangedEventArgs args)
{
         // Do something here
}
```

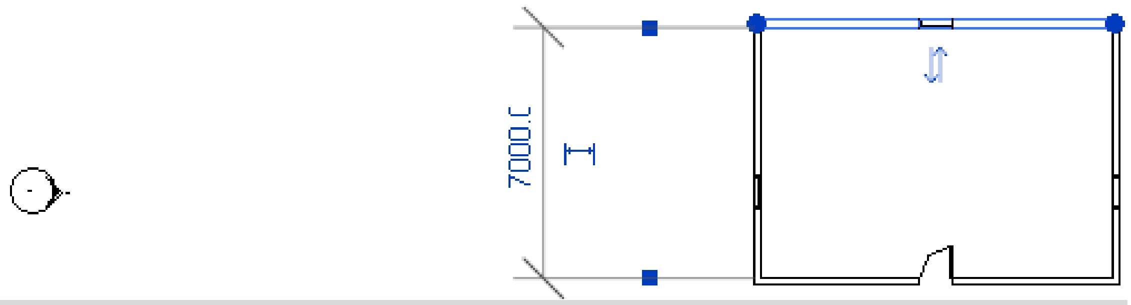
```
public Result OnStartup(UIControlledApplication application)
 {
    application.ControlledApplication.DocumentChanged += UILabs_DocumentChanged;
    return Result.Succeeded;
 }
```

```
public Result OnShutdown(UIControlledApplication application)
{
     application.ControlledApplication.DocumentChanged -= UILabs_DocumentChanged;
     return Result.Succeeded;
}
```

Lab - Events


EventOn - Revit UI Labs - Event

Added:
Deleted:
Modified: 127583 127584 127585 127586 127588 127589 127590 127592 127593 127595 127596 127597 127598 127599 127600 127602 127603 127604 127605 127606 127607 127608 127609



```
// register the document changed event
application.ControlledApplication.DocumentChanged += UILabs_DocumentChanged;
```

```
// you can get the list of ids of element added/changed/modified.
Document rvtdDoc = args.GetDocument();

ICollection<ElementId> idsAdded = args.GetAddedElementIds();
ICollection<ElementId> idsDeleted = args.GetDeletedElementIds();
ICollection<ElementId> idsModified = args.GetModifiedElementIds();

// put it in a string to show to the user.
string msg = "Added: ";
foreach (ElementId id in idsAdded)
{
    msg += id.IntegerValue.ToString() + " ";
}
```

Dynamic Model Update Overview

“Ability for a Revit API application to modify the Revit model as a reaction to changes happening in the model”.

Helps track element addition, modification and deletion

Dynamic Model Update

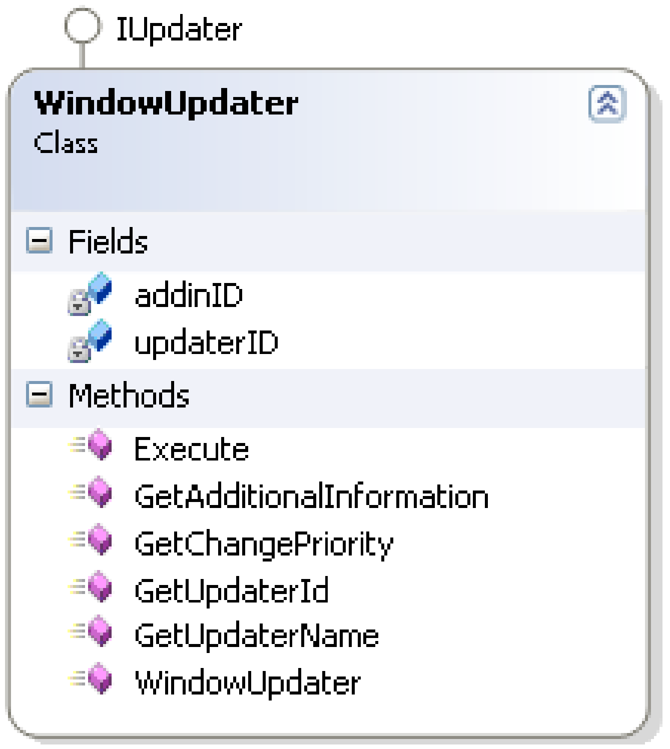
Updaters

Updaters :

Ability to implement a method that is informed of the scope of changes

Implements the *IUpdater* interface.

- GetUpdaterId ()
- GetUpdaterName()
- GetAdditionalInformation ()
- GetChangePriority()
- Execute()



Dynamic Model Update

Registration and Triggers

Register the Updater

- OnStartUp for application level scope

```
WindowUpdater updater = new WindowUpdater(application.ActiveAddInId );
// Register the updater in the singleton UpdateRegistry class
UpdaterRegistry.RegisterUpdater( updater );
```

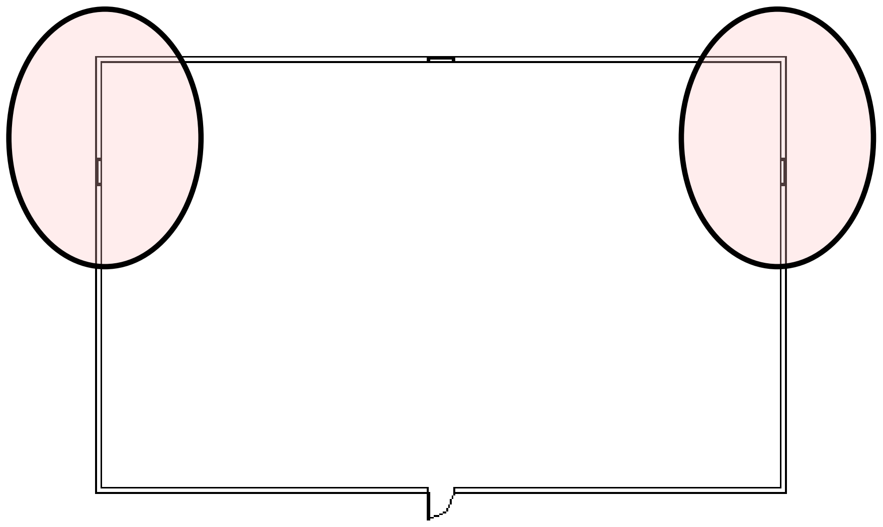
Add Trigger

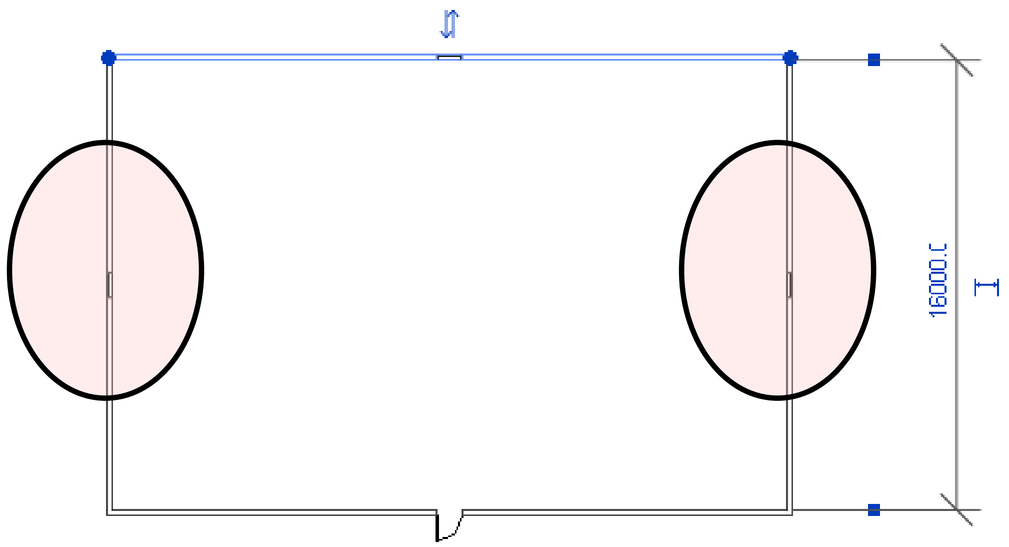
- Change of Scope - list of ElementIds or list of elements via ElementFilter.

```
// Set the filter
ElementClassFilter filter = new ElementClassFilter( typeof( Wall ) );
// Add trigger
UpdaterRegistry.AddTrigger(updater.GetUpdaterId(),filter,
Element.GetChangeTypeGeometry());
```

Lab - Dynamic Model Update

```
    // construct our updater.
    WindowDoorUpdater winDoorUpdater =
        new WindowDoorUpdater(application.ActiveAddInId);

    // ActiveAddInId is from addin menifest. register it
    UpdaterRegistry.RegisterUpdater(winDoorUpdater);

    // tell which elements we are interested in notified.
    // we want to know when wall changes it's length.

    ElementClassFilter wallFilter = new ElementClassFilter(typeof(Wall));
    UpdaterRegistry.AddTrigger(
        winDoorUpdater.GetUpdaterId(), wallFilter, Element.GetChangeTypeGeometry());
```





Conclusion

Where do we go next …

We have covered…

UI Topics

- Ribbon
- User Selection
- Task dialog
- Events
- Dynamic model update

Learning More

Online Help, Developer's Guide and SDK Samples

Developer Resources for Revit API

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

Discussion Groups

- http://discussion.autodesk.com > Revit Architecture > Revit API

API Training Classes

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

The Building Coder, Jeremy Tammik's Revit API Blog

- http://thebuildingcoder.typepad.com

ADN AEC Developer Blog

- http://adndevblog.typepad.com/aec/

Developer Wiki

- http://www.autodesk.com/revit-help/?guid=GUID-F0A122E0-E556-4D0D-9D0F-7E72A9315A42

Autodesk Developer Network

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

DevHelp Online for ADN members

- http://adn.autodesk.com

Thank you!

AUTODESK®