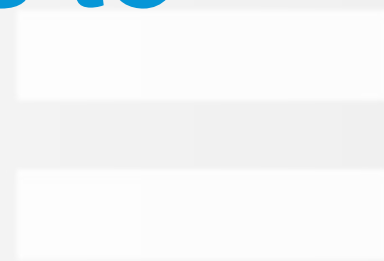


Using Revit API Events to Manage Miracles

Don Rudder

Associate Director of Desktop Applications

@aybabtm *case*



bldgs

data



Quick Introduction

- I'm from Austin, TX
- Started AEC Career in 1996 (Civil & MEP)
- Actively developing software for 18+ years
- Morphed into BIM/CAD Management in 2004
 - 4 years MEP, 3 in Architecture
- Started working for CASE in 2011

bldgs

data

Class Materials

- This presentation
- 12 page handout
- Visual Studio 2013 Solution



Full Class Materials Download

The Four Learning Objectives

- Understand what Revit API events are and some notable limitations
- Know how to subscribe and unsubscribe to events in Revit
- See several examples on creative Revit API event uses
- Bypass some of the Revit API event rule limitations with IUpdater





Understand what Revit API events are and a few notable limitations

bldgs

data

Image credit www.case-inc.com

Revit API Events Defined (Basic)

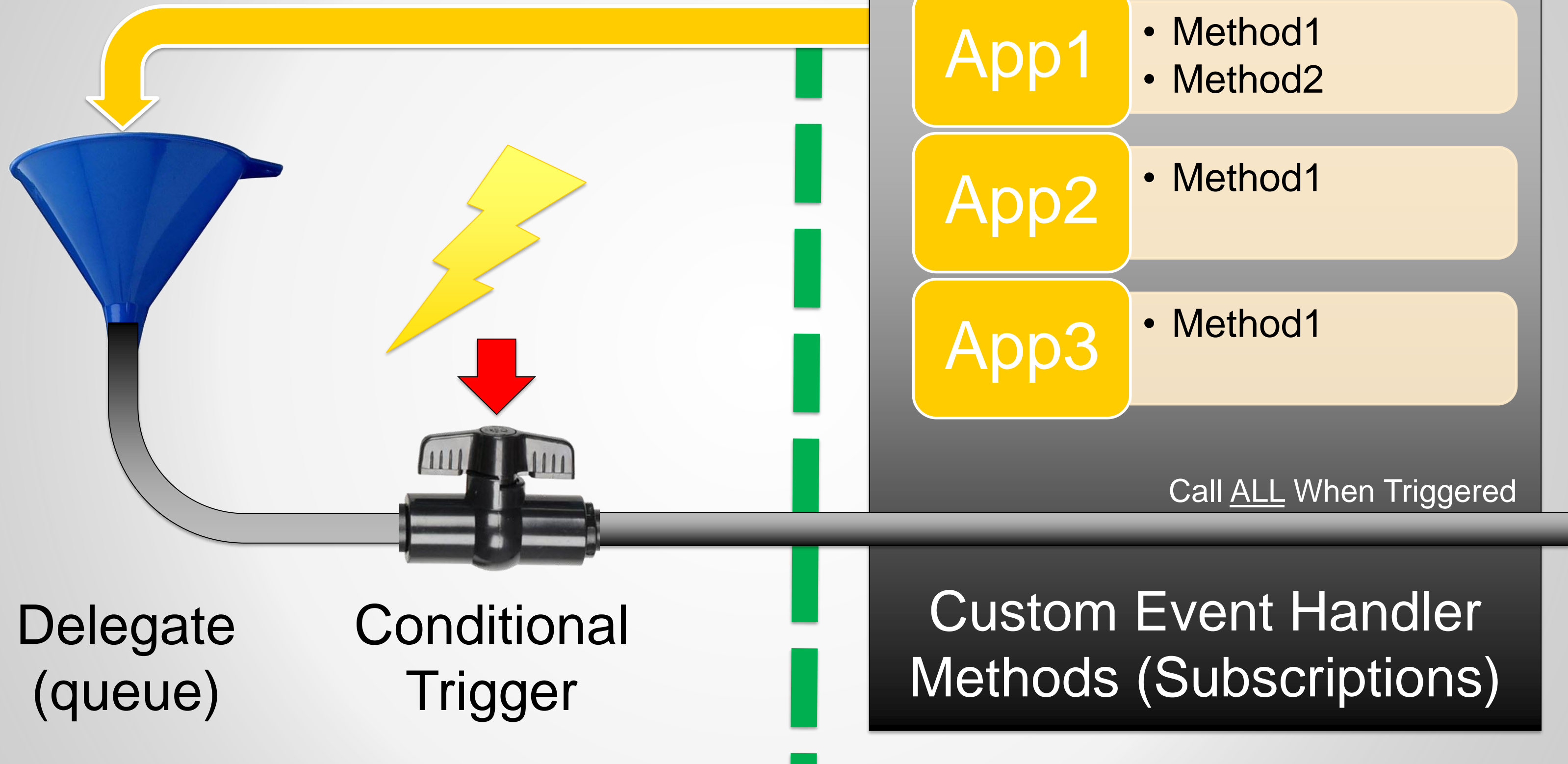
- A means for a custom Revit add-in to receive notifications when something specific changes
- Provide the ability for a custom add-in to perform action(s) without user intervention



Slightly Deeper Definition

- Revit events are a kind of delegate
 - Delegates are smart objects that deal with references to methods, as opposed to objects or properties
 - Delegates can reference any number of methods so long as their signature matches the requirement
- How they work
 - Event fires off, or is “raised”
 - Internal Revit API event delegate calls each registered event handler method sending its predefined internal values as input parameters

Basic Delegate Diagram



Supported Events

The [Document](#) type exposes the following members.

Events

	Name
⚡	DocumentClosing
⚡	DocumentPrinted
⚡	DocumentPrinting
⚡	DocumentSaved
⚡	DocumentSavedAs
⚡	DocumentSaving
⚡	DocumentSavingAs
⚡	ViewPrinted
⚡	ViewPrinting

The [UIApplication](#) type exposes the following members.

Events

	Name	Description
⚡	ApplicationClosing	Subscribe to the ApplicationClosing event to be notified when the Revit application is just about to be closed.
⚡	DialogBoxShowing	Subscribe to the DialogBoxShowing event to be notified when Revit is just about to show a dialog box or a message box.
⚡	DisplayingOptionsDialog	Subscribe to the options dialog displaying event to be notified when Revit options dialog is displaying.
⚡	DockableFrameFocusChanged	Subscribe to this event to be notified when a Revit GenericDockableFrame has gained focus or lost focus in the Revit user interface. This event is called only for API-created GenericDockableFrames.
⚡	DockableFrameVisibilityChanged	Subscribe to this event to be notified when a Revit GenericDockableFrame has been shown or hidden in the Revit user interface. This event is called only for API-created GenericDockableFrames.
⚡	Idling	Subscribe to the Idling event to be notified when Revit is not in an active tool or transaction.
⚡	ViewActivated	Subscribe to the ViewActivated event to be notified immediately after Revit has finished activating a view of a document.
⚡	ViewActivating	Subscribe to the ViewActivating event to be notified when Revit is just about to activate a view of a document.

Partial List of Revit 2015 API Supported Events

- Some new and a few of the more common include:
 - DocumentWorksharingEnabled
 - ElementTypeDuplicating & ElementTypeDuplicated
 - ViewActivating & ViewActivated
 - FamilyLoadingIntoDocument & FamilyLoadedIntoDocument
 - DocumentSynchronizingWithCentral & DocumentSynchronizedWithCentral
- Refer to the Revit API SDK help file (chm) for a complete list

Notable Rules & Limitations

- Custom event handler methods must match the signature for the event that you are subscribing to
- Most Revit API events do not support model changes
 - The Idling event is an exception to this rule
- Event subscriptions will only persist as long as the source that subscribed them



System Resources Caution

- Events alone do not consume a lot of resources, its what you do with them that will
- Focus on specific tasks
- Avoid long running sequences





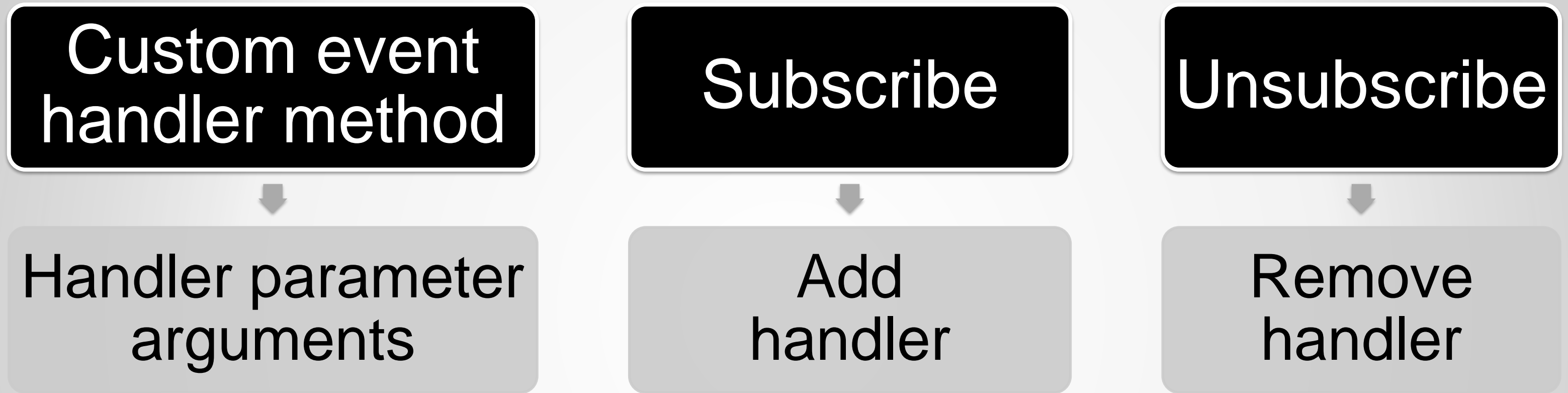
Know how to subscribe and unsubscribe to events in Revit

bldgs

data

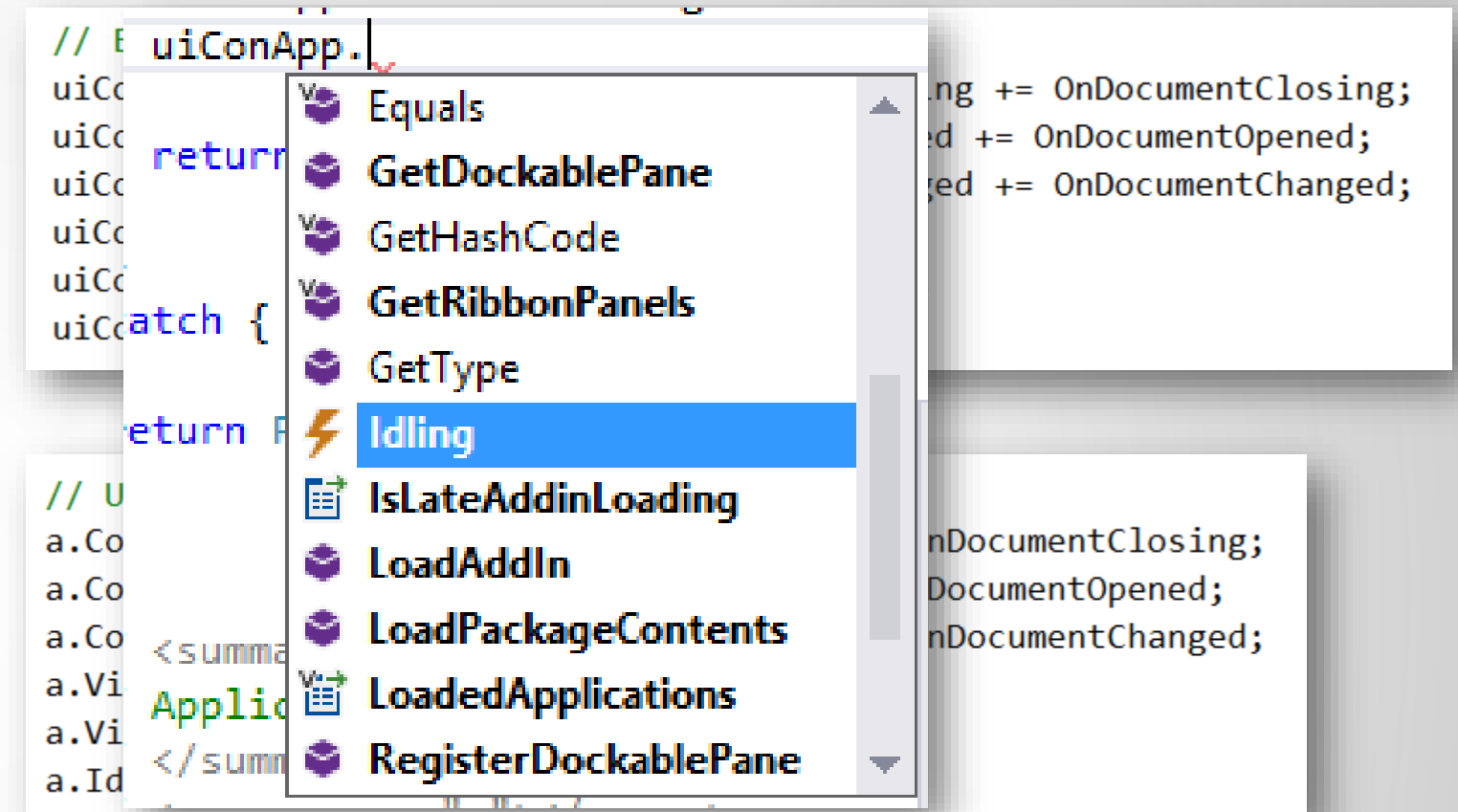
Image credit www.case-inc.com

Anatomy of a Revit API Event Implementation



Subscribe and Unsubscribe from Events

- Event subscriptions remain active until either
 - Unsubscribed
 - Original subscribing command or application terminates
- Always unsubscribe your events either at application shutdown or when you are done using them





**See several examples on creative
Revit API event uses**

bldgs

data

Image credit www.case-inc.com

Sample #1 – Log Time Spent in Views

```
/// <summary>
/// Fires off after a view has been activated.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
internal void OnViewActivated(object sender, ViewActivatedEventArgs e)
{
    try
    {
        ActiveDoc = e.Document;
        if (ActiveDoc == null) return;

        // New View Timer
        ActiveViewTimer = new clsViewTime(e.CurrentActiveView);

        PageEventData.LabelActiveViewName.Content = e.CurrentActiveView == null
            ? "No Active View"
            : string.Format("{0} {1}",
                e.CurrentActiveView.ViewType,
                e.CurrentActiveView.Name);
    }
}
```

```
/// <summary>
/// Fires off each time a view is switched. This is
/// also a good way to know when a new model is open and
/// active in the editor.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
internal void OnViewActivating(object sender, ViewActivatingEventArgs e)
{
    if (e.Document == null) return;

    try
    {
        // End Active View
        if (ActiveViewTimer != null)
        {
            ActiveViewTimer.SetClosed();
            Data.Views.Add(ActiveViewTimer);
            Data.Views.SortBackwards(a => a.Opened);
        }
    }
}
```



Sample #2 – Export View Images

- ViewActivating
- ViewActivated

Export an image using the current date and time along with the name of the model and view.

Supports image export on view open, closed, or both

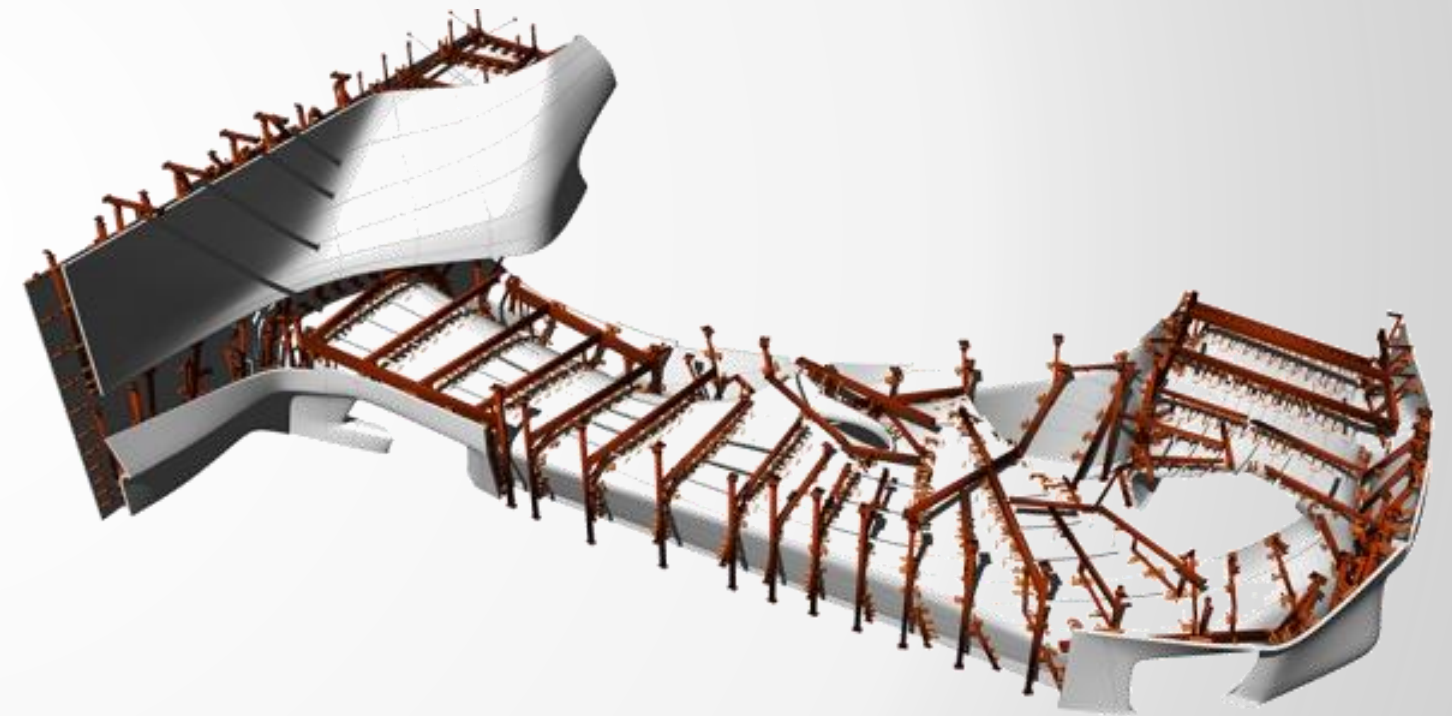


Image credit www.case-inc.com

Sample #2 – Export View Images

```
/// <summary>
/// Export image from active view
/// </summary>
/// <param name="isOpened"></param>
internal static void ExportViewImage(bool isOpened)
{
    try
    {
        // Export view image
        string m_event = isOpened ? "Opened" : "Closed";
        ImageExportOptions m_options = new ImageExportOptions
        {
            ExportRange = ExportRange.CurrentView
        };

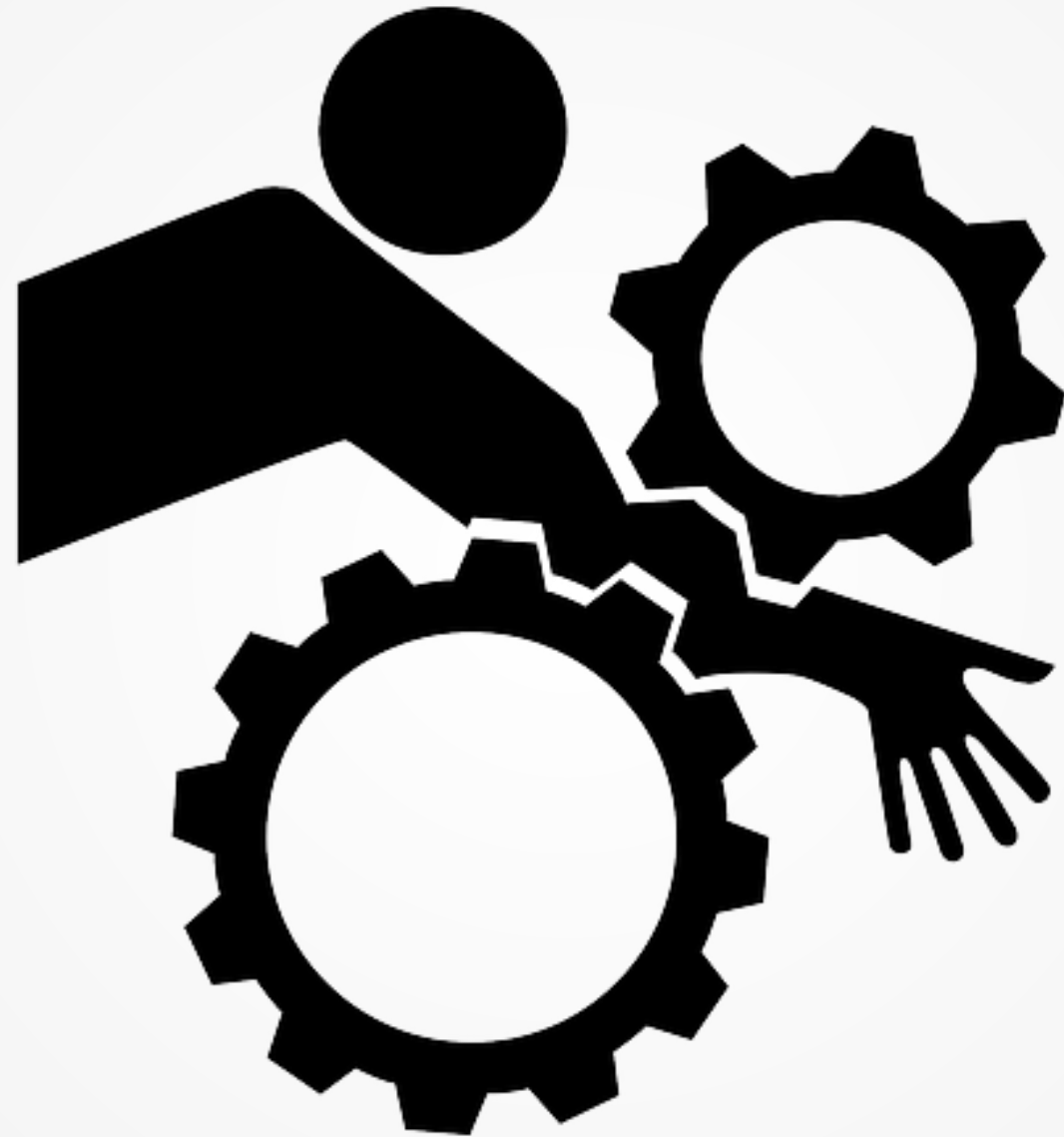
        // File Name
        string m_fileName = String.Format("{0}_{1}_{2}_{3}",
            DateTime.Now.ToString("yy-MM-dd-hh-mm-ss"),
            Path.GetFileNameWithoutExtension(AppMain.ActiveDoc.Title),
            AppMain.ActiveDoc.ActiveView.Name,
            m_event);

        // Save the image in same directory as executing dll
        m_options.FilePath = Path.Combine(GetAssemblyPath(), m_fileName);

        m_options.ImageResolution = ImageResolution.DPI_300;
        m_options.ShadowViewsFileType = ImageFileType.PNG;
        AppMain.ActiveDoc.ExportImage(m_options);
    }
    catch { }
}
```

- Can be used to visually track progress for projects similar to how construction companies use video time-lapse
- Be careful to monitor disk space at the location where you send the images

Demo



Sample #3 – Log Key Element Deletions

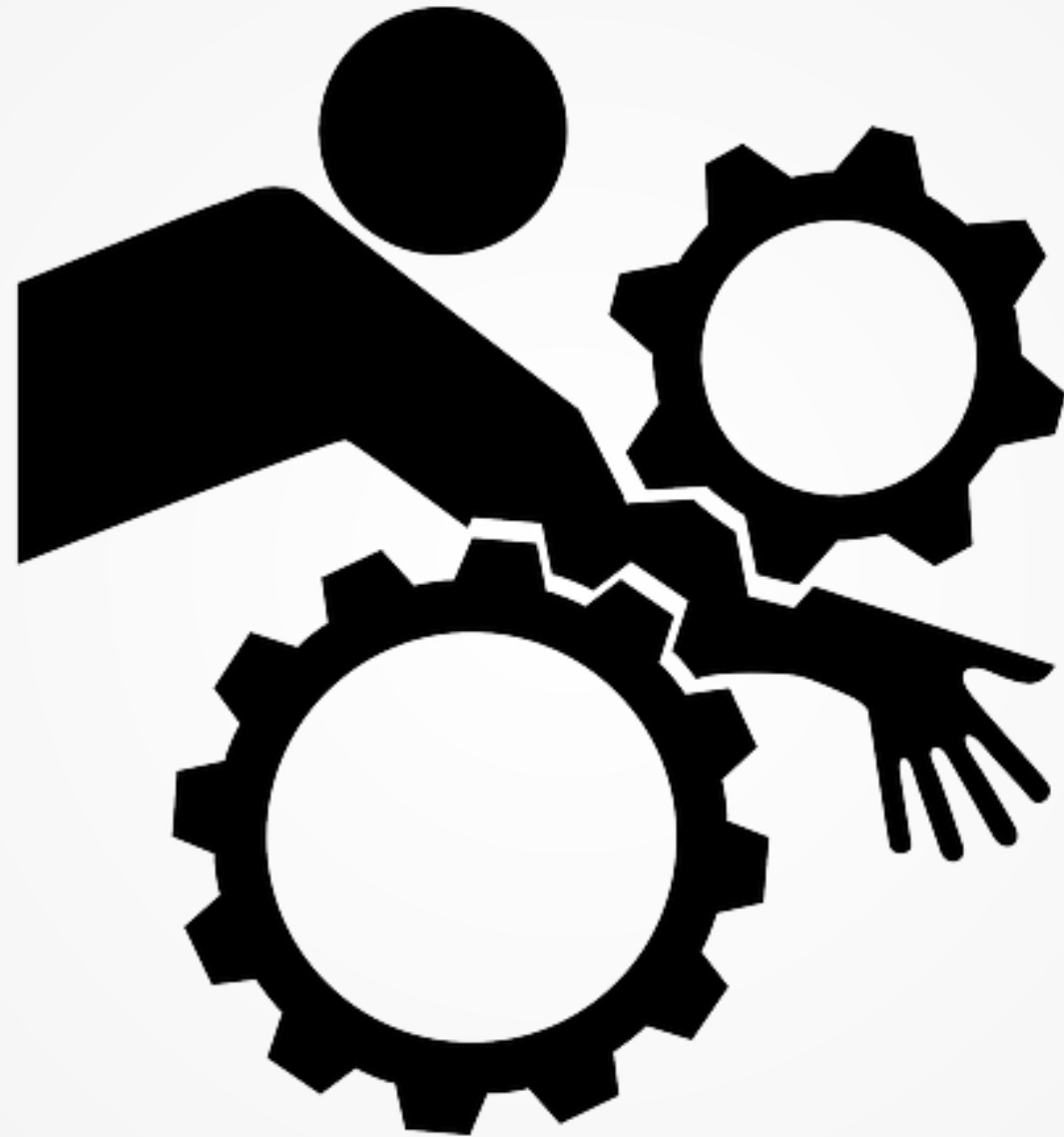
- DocumentChanged

Know when elements of specific categories get deleted and who deleted them.

Sample code focuses on walls, doors, and levels



Demo

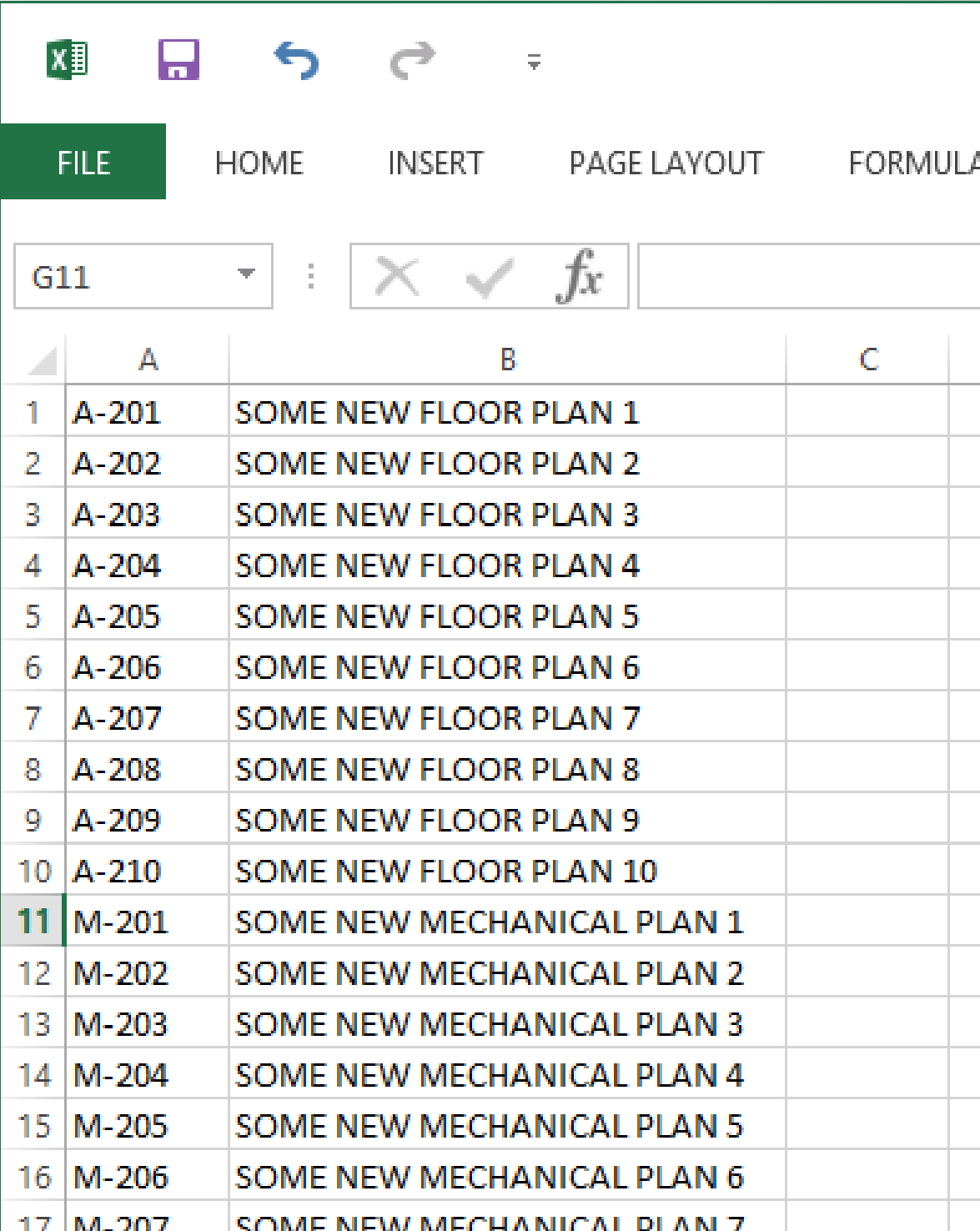


Sample #4 – Generate Sheets from External File

- Idling

Adds sheets if they don't already exist in the model

Easily coordinate sheet listings between external consultants using nothing more than a simple text file



The screenshot shows an Excel spreadsheet with the following data:

	A	B	C
1	A-201	SOME NEW FLOOR PLAN 1	
2	A-202	SOME NEW FLOOR PLAN 2	
3	A-203	SOME NEW FLOOR PLAN 3	
4	A-204	SOME NEW FLOOR PLAN 4	
5	A-205	SOME NEW FLOOR PLAN 5	
6	A-206	SOME NEW FLOOR PLAN 6	
7	A-207	SOME NEW FLOOR PLAN 7	
8	A-208	SOME NEW FLOOR PLAN 8	
9	A-209	SOME NEW FLOOR PLAN 9	
10	A-210	SOME NEW FLOOR PLAN 10	
11	M-201	SOME NEW MECHANICAL PLAN 1	
12	M-202	SOME NEW MECHANICAL PLAN 2	
13	M-203	SOME NEW MECHANICAL PLAN 3	
14	M-204	SOME NEW MECHANICAL PLAN 4	
15	M-205	SOME NEW MECHANICAL PLAN 5	
16	M-206	SOME NEW MECHANICAL PLAN 6	
17	M-207	SOME NEW MECHANICAL PLAN 7	

Sample #4 – Generate Sheets from External File

```
/// <summary>
/// Document Idle!
/// Handlers of this event are permitted to make modifications
/// to any document (including the active document), except for
/// documents that are currently in read-only mode.
/// </summary>
/// <param name="sender"></param>
/// <param name="idlingEventArgs"></param>
private void OnIdling(object sender, IdlingEventArgs idlingEventArgs)
{
    UIApplication m_uiApp = sender as UIApplication;
    if (m_uiApp == null) return;
    if (m_uiApp.ActiveUIDocument == null) return;
    Document m_doc = m_uiApp.ActiveUIDocument.Document;

    const string c_fileName = @"c:\temp\sheets_new.txt";
    if (!File.Exists(c_fileName)) return;
    Dictionary<string, string> m_newSheets = new Dictionary<string, string>();
    using (StreamReader sr = new StreamReader(c_fileName))
    {
        while (!sr.EndOfStream)
        {
            string m_line = sr.ReadLine();
            if (string.IsNullOrEmpty(m_line)) continue;
            string[] m_txt = m_line.Split('\t');
            if (!string.IsNullOrEmpty(m_txt[0]) && !string.IsNullOrEmpty(m_txt[1]))
            {
                m_newSheets.Add(m_txt[0], m_txt[1]);
            }
        }
    }
    if (m_newSheets.Count < 1) return;

    // Check for a titleblock
    FamilySymbol m_titleBlock = (from e in new FilteredElementCollector(m_doc)
                                  .WhereElementIsElementType()
                                  .OfCategory(BuiltInCategory.OST_TitleBlocks)
                                  .Cast<FamilySymbol>()
                                  select e).First();
```

```
// Sheets
IEnumerable<ViewSheet> m_query = from e in new FilteredElementCollector(m_doc)
    .WhereElementIsNotElementType()
    .OfClass(typeof(ViewSheet))
    let sh = e as ViewSheet
    select sh;

Dictionary<string, ViewSheet> m_sheets =
    m_query.ToList().ToDictionary(x => x.SheetNumber);

// Create Missing Sheets
foreach (var x in m_newSheets)
{
    if (m_sheets.ContainsKey(x.Key)) continue;
    using (Transaction t = new Transaction(m_doc))
    {
        if (t.Start("New Sheet") != TransactionStatus.Started) continue;
        try
        {
            ViewSheet m_sheet = m_titleBlock != null
                ? ViewSheet.Create(m_doc, m_titleBlock.Id)
                : ViewSheet.CreatePlaceholder(m_doc);

            m_sheet.SheetNumber = x.Key;
            m_sheet.Name = x.Value;
            t.Commit();
        }
        catch { }
    }
}

// Rename the text file
File.Move(c_fileName, @"c:\temp\sheets.txt");
}
```


Demo





Bypass some of the Revit API event rule limitations with IUpdater

bldgs

data

Image credit www.case-inc.com

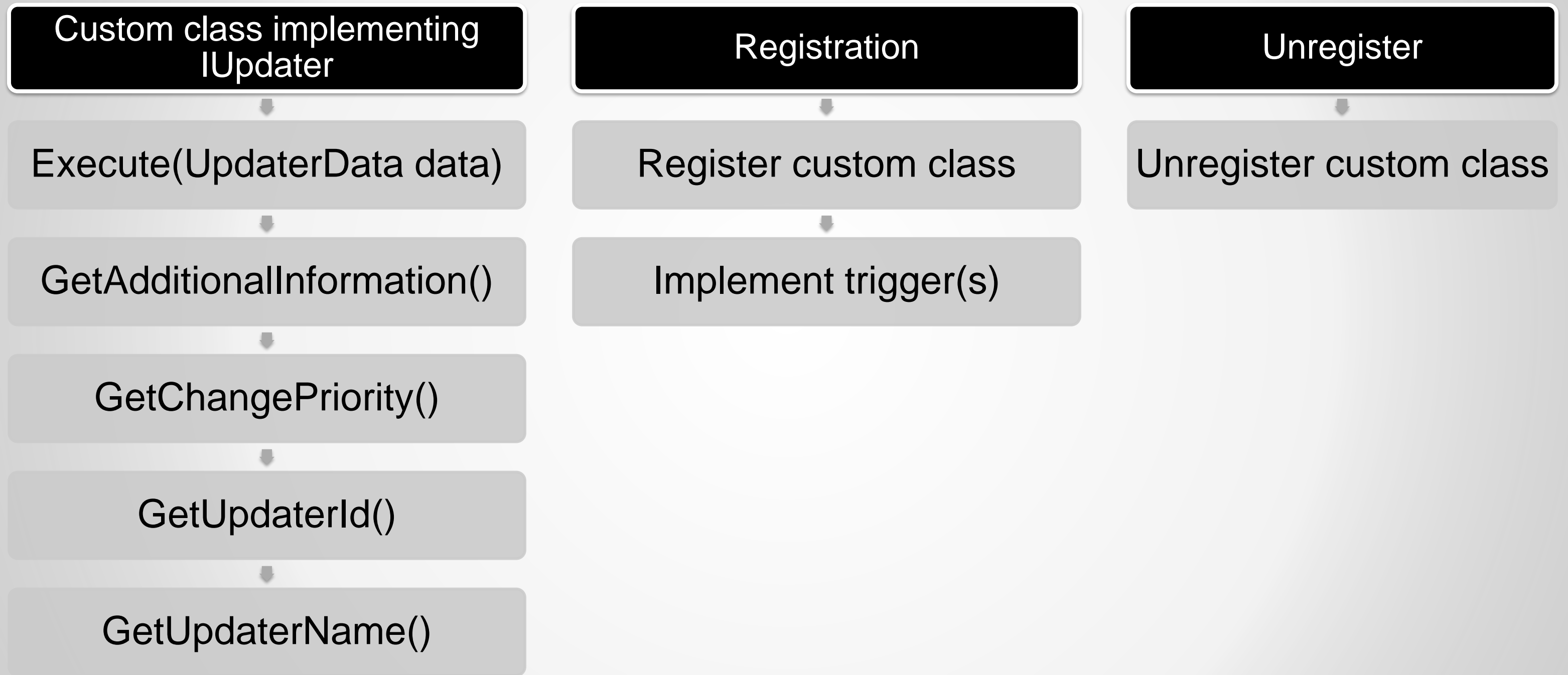
IUpdater Features

- Customize what you want to “listen” for
- Customize how to react
- Can be turned on or off at any time
- Several can be used concurrently
- Supports model changes
 - Changes made to the model are handled under the original transaction where the model change occurred (for undo)



Image courtesy of Wikipedia

Anatomy of IUpdater



Implementing the IUpdater Interface

```
public class UpdaterXYZ : IUpdater
{
    static UpdaterId _updaterId;

    /// <summary>
    /// Constructor
    /// </summary>
    /// <param name="aId"></param>
    /// <param name="updGuid"></param>
    public UpdaterXYZ(AddInId aId, string updGuid)
    {
        _updaterId = new UpdaterId(aId, new Guid(updGuid));
    }

    /// <summary>
    /// Reaction for Notified Elements
    /// </summary>
    /// <param name="data"></param>
    public void Execute(UpdaterData data)
    {
        // Update new elements
        UpdateXYZData(data.GetDocument(), data.GetAddedElementIds());

        // Update modified elements
        UpdateXYZData(data.GetDocument(), data.GetModifiedElementIds());
    }
}
```

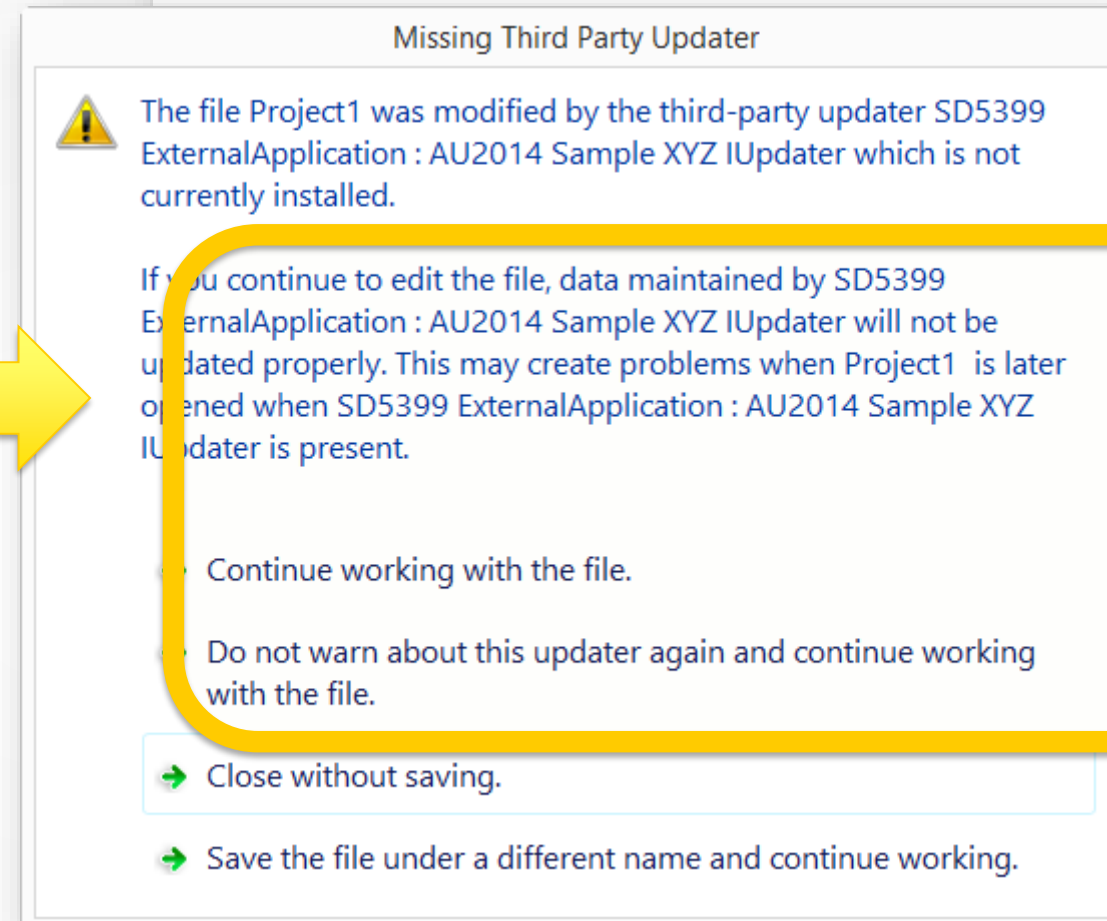
- Create new custom class
- Implement IUpdater
- Implement your custom change code within the provided Execute method

IUpdater Registration

- UpdaterRegistry
 - Custom class
 - Flag to set updater as optional (for model)
- Set Trigger(s)
 - ChangeType (Reqd.)
 - Document (Optional)
 - Element Filter or Collection of ElementId's

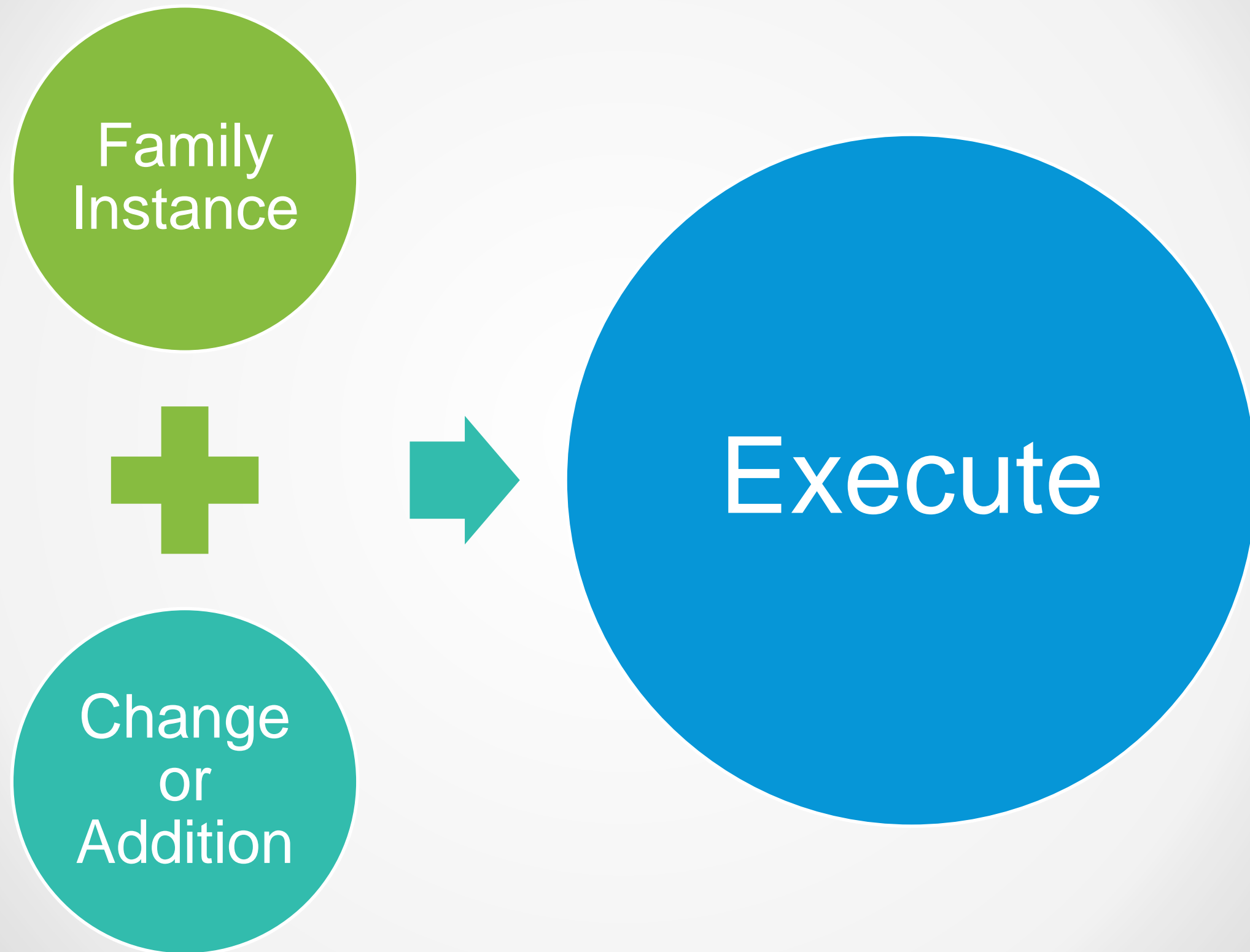


```
/// <summary>
/// Register an IUpdater
/// </summary>
private void RegisterUpdater(UIApplication uiApp)
{
    try
    {
```

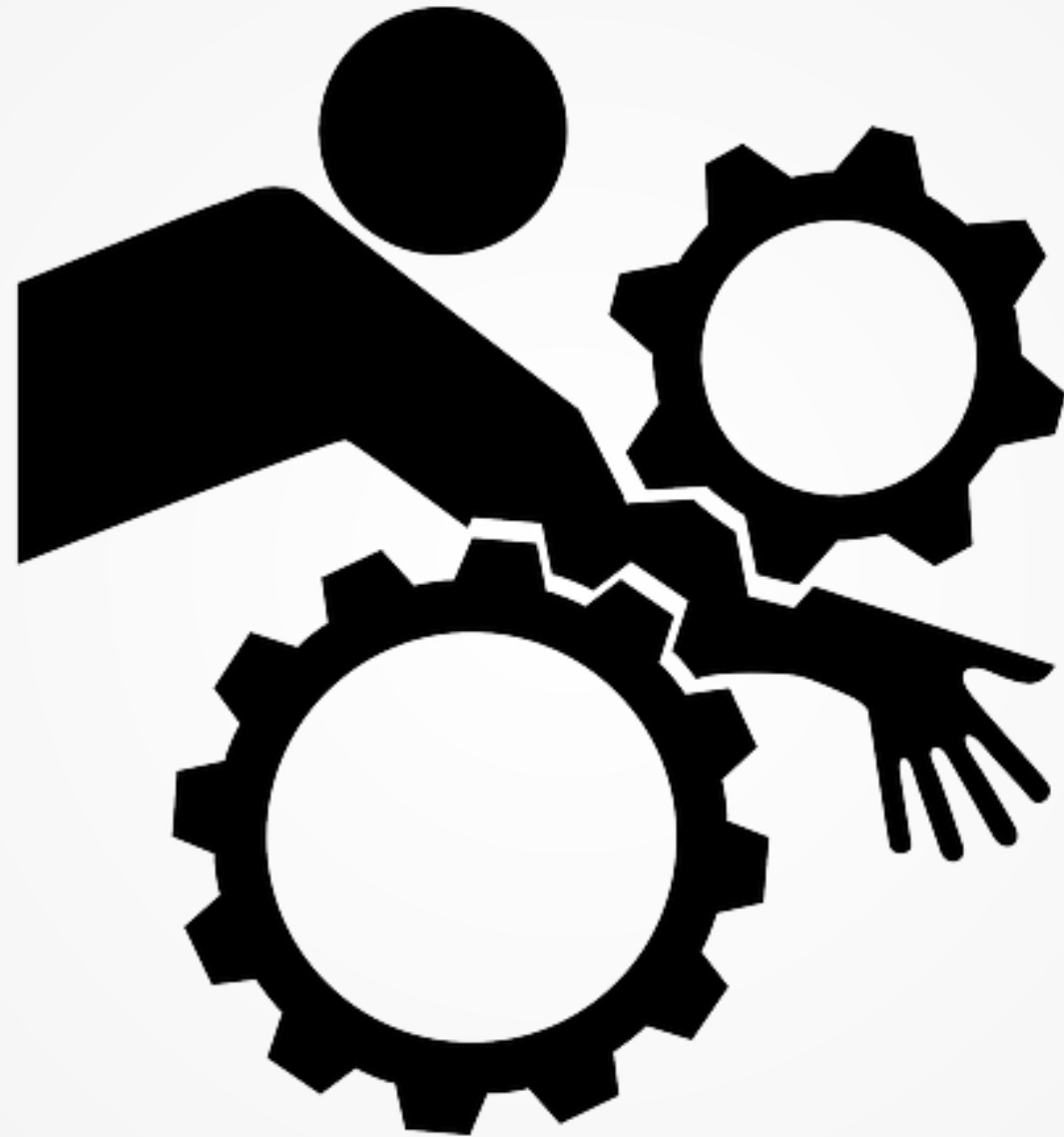


```
{
    td.TitleAutoPrefix = false;
    td.MainInstruction = "Failed to Register Updater";
    td.MainContent = string.Format(
        "Something failed while trying to register the IUpdater: \n\n{0}",
        ex);
    td.Show();
}
}
```


IUpdater Sample – XYZ to Comments Parameter



Demo





Please Remember to Fill Out a Class Survey for a Chance to Win

Don Rudder

Associate Director of Desktop Applications

@aybabtm **case**

bldgs

data

