



FORGE

- Crane Tower Accelerator
- Lightning Talk: Migrate Revit plugin to Design Automation
 - Zhong Wu
 - Forge Developer Advocate
 - May 2020

Goal

Understand the main tasks to migrate your existing Revit desktop plugin to run on Design Automation engine.



AUTODESK[®]
FORGE

Success Stories Solutions ▾ Getting Started Documentation Community ▾ Support ▾ Pricing

Design Automation API v3 ▾

> Developer's Guide

▼ Step-by-Step Tutorials

> Design Automation API for 3ds Max
> Design Automation API for AutoCAD
> Design Automation API for Inventor
> Design Automation API for Revit
 About this Tutorial
 Task 1 – Convert Revit Add-in
 Task 2 – Obtain an Access Token
 Task 3 – Create a Nickname
 Task 4 – Upload an AppBundle
 Task 5 – Publish an Activity
 Task 6 – Prepare Cloud Storage
 Task 7 – Submit a WorkItem

> Common Tutorials (Advanced)

> Code Samples & Blog Posts

> API Reference

> Change History

Task 1 – Convert Revit Add-in to Design Automation Add-in

This task converts an add-in that runs on Revit to an add-in that runs on Design Automation.

Prerequisites for this task are:

- Visual Studio 2017 or Visual Studio 2019
- Revit 2018, Revit 2019, Revit 2020 or Revit 2021: This is required to compile the changes into the add-in.
- Basic knowledge of C#

Expected task outcome

By the end of this task you will know how to convert a regular Revit add-in to one that runs on Design Automation.

Step 1 - Clone Git repository

Clone the [Git Repository for the DeleteWalls Sample](#), go to the folder named `Desktop_Version`, and open `DeleteWalls.sln` in Visual Studio.

The DeleteWalls Sample Git repository contains the source code for an add-in named DeleteWalls. DeleteWalls reads a `.rvt` file and produces another `.rvt` file with all the walls deleted.

The repository contains two folders for two different versions of DeleteWalls. The folder named `Desktop_Version` contains a C# project that produces a typical Revit add-in that runs only on Revit. It does not run on Design Automation. The folder named `Design-Automation_Version` contains the same project, modified to run on Design Automation. The objective of this task is to start with the C# project in `Desktop_Version` and modify it to run on Design Automation. You will do this by cloning the repository, opening the solution in Visual Studio, and modifying the project files to target Design Automation.

Step 2 - Repair references

The C# project you cloned may expect to find `RevitAPI.dll` in a location that is different to where it resides on your computer. To eliminate the risk of a broken reference:

1. Find `RevitAPI.dll` in your Revit install location and note its location.
2. In Visual Studio, remove the reference to `RevitAPI.dll`.
3. Add a reference to `RevitAPI.dll`, pointing to the location you noted down earlier.

Step 3 - Add a package reference to the DesignAutomationBridge DLL

Autodesk provides a library that contains the functionality an add-in needs to interface with Design Automation. This library is known as the Design Automation Bridge and is distributed as a NuGet package at <https://www.nuget.org/packages/Autodesk.Forge.DesignAutomation.Revit>.

1. In Visual Studio, remove the reference to `RevitAPIUI.dll`.
2. Insert a package reference to the Design Automation Bridge corresponding to the Revit version you want to run.

Please refer [Microsoft Documentation](#) for instructions.

Tip: When inserting the package reference using Visual Studio, search for `Autodesk.Forge.DesignAutomation.Revit` with the `Include prerelease` option selected.

Step 4 - Remove references to user interface elements

Since there is no UI interaction in Design Automation, you must remove all references to UI elements.

In the .cs file that implements your add-in (`DeleteWalls.cs` in this case):

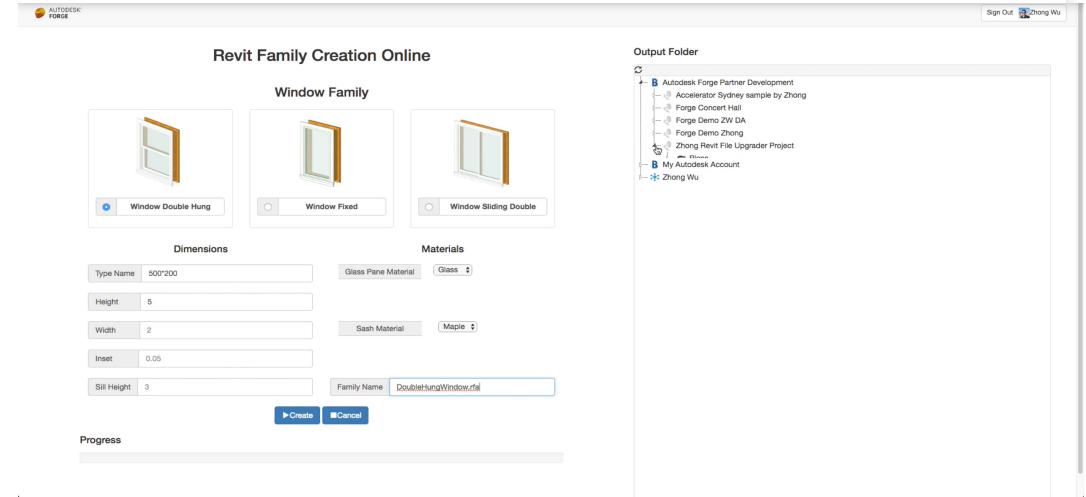
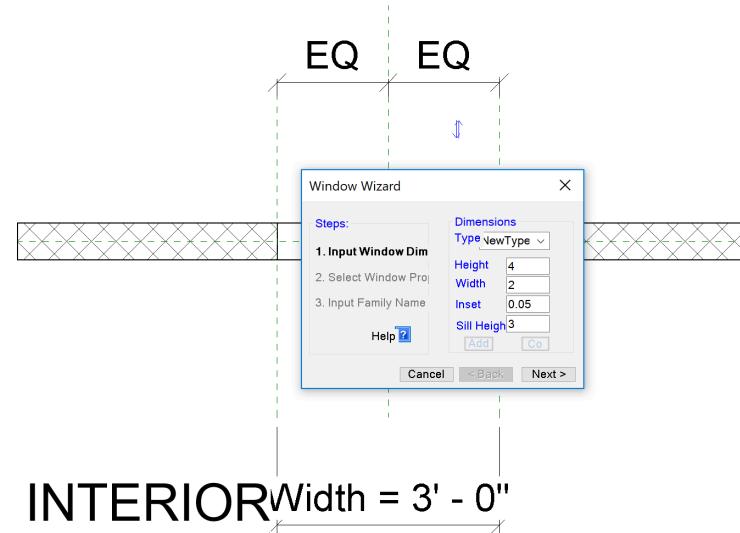
1. Remove the `using` directive to the `Autodesk.Revit.UI` namespace and insert a `using` directive to the `DesignAutomationFramework` namespace in its place.

Revit Design Automation Tutorial

\Revit 2020
SDK\Samples\FamilyCreation\WindowWizard\CS

<https://github.com/Autodesk-Forge/forge-createfamily-revit/tree/master/CreateWindow/PlugIn>

EXTERIOR



Plugin Migration Demo

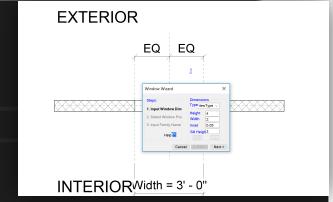
Step 1 Remove UI

- Remove reference to RevitAPIUI.dll,
System.Windows.dll
- Remove Autodesk.Revit.UI and
System.Windows.Forms
- Remove WizardUI.cs
- UIApplication -> Application
- ExternalCommandData -> CreateWindowData

```
public class CreateWindowData {  
    public Application Application { get; set; }  
    public Document Document { get; set; }  
}
```

Step 2 Handle user input by JSON

- NuGet: Newtonsoft.Json
- Create class TypeDAParams and WindowsDAParams



```
internal class TypeDAParams
{
    public String TypeName { get; set; } = "New Type";
    public Double WindowHeight { get; set; } = 4;
    public Double WindowWidth { get; set; } = 2;
    public Double WindowInset { get; set; } = 0.05;
    public Double WindowSillHeight { get; set; } = 3;
}

internal class WindowsDAParams
{
    public TypeDAParams[] Types { get; set; } = { new TypeDAParams() };

    public String WindowStyle { get; set; } = "DoubleHungWindow";
    public String GlassPaneMaterial { get; set; } = "Default";
    public String SashMaterial { get; set; } = "Default";
    public String WindowFamilyName { get; set; } = "Double Hung.rfa";

    static public WindowsDAParams Parse(string jsonPath)
    {
        ...
    }
}

"inputJson": {
    "url": "data:application/json,{ 'Types': [{TypeName': 'My Type', 'WindowHeight': 6, 'WindowWidth': 4, 'WindowInset': 0.05, 'WindowSillHeight': 3}, {'TypeName': 'My New Type', 'WindowHeight': 8, 'WindowWidth': 6, 'WindowInset': 0.05, 'WindowSillHeight': 6}], 'GlassPaneMaterial': 'Glass', 'SashMaterial': 'Maple', 'WindowStyle': 'DoubleHungWindow' }"
},
```

Step 3

IExternalDBApplication

- Nuget:
 - DesignAutomationBridge.dll,
- IExternalDBApplication

```
[Autodesk.Revit.Attributes.Regeneration(Autodesk.Revit.Attributes.RegenerationOption.Manual)]
[Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)]
public class CreateWindowApp : IExternalDBApplication
{
    public ExternalDBApplicationResult OnStartup(ControlledApplication application)
    {
        DesignAutomationBridge.DesignAutomationReadyEvent += HandleDesignAutomationReadyEvent;
        return ExternalDBApplicationResult.Succeeded;
    }

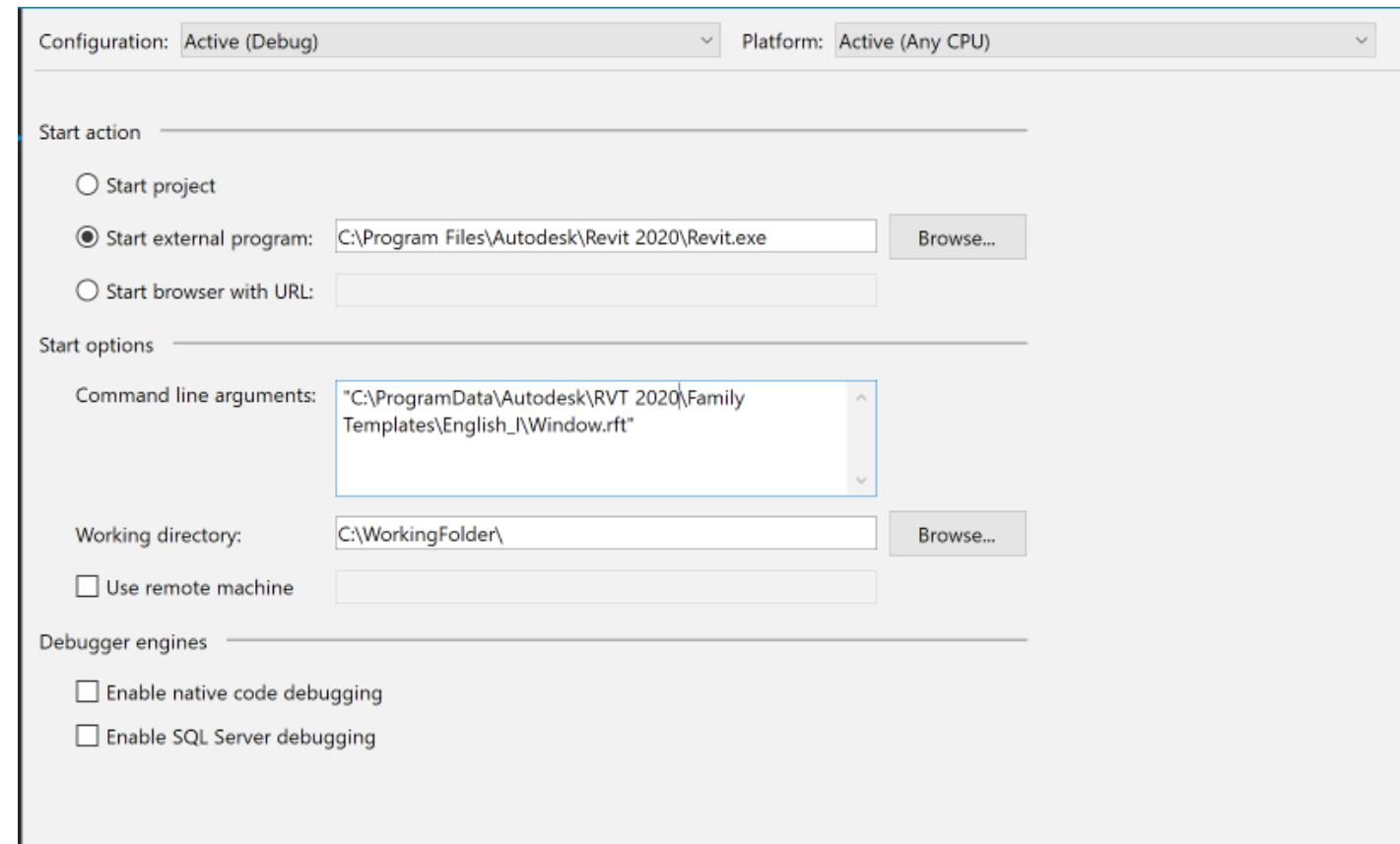
    public void HandleDesignAutomationReadyEvent( object sender, DesignAutomationEventArgs e)
    {
        e.Succeeded = true;
        e.Succeeded = CreateWindowFamily(e.DesignAutomationData);
    }

    protected bool CreateWindowFamily( DesignAutomationData data )
    {
        if (data == null)
            return false;

        Application app = data.RevitApp;
        if (app == null)
            return false;

        string modelPath = data.FilePath;
        if (String.IsNullOrWhiteSpace(modelPath))
            return false;
```

Step 4 Local debug



DesignAutomationHandler

<https://forge.autodesk.com/blog/design-automation-debug-revit-plugin-locally>
<https://github.com/Autodesk-Forge/design.automation-csharp-revit.local.debug.tool>

Step 5 AppBundle

```
WindowWizard.zip
| -- WindowWizard.bundle
|   | -- PackageContents.xml
|   | -- Contents
|     | -- WindowWizard.dll
|     | -- WindowWizard.addin
```

More details

Autodesk-Forge / forge-createfamily-revit

Watch 10 Star 8 Fork 9

Code Issues 2 Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

Branch: master → forge-createfamily-revit / CreateWindow / Plugin / Create new file Upload files Find file History

JohnOnSoftware polish the code Latest commit 4245ee2 on Aug 26, 2019

..

Source polish the code 7 months ago

README.md fix readme 14 months ago

README.md

Create Window Family Addin

Description

This sample demonstrated how to migrate the Revit WindowWizard Plugin to be used within AppBundle of Design Automation for Revit.

Migration Steps

Here is the main steps to migrate the Revit addin, before read the detail steps, please make sure you go through the official [Convert Addin Doc](#) and understand the framework.

Before we start:

- Build the WindowWizard project under \Revit 2019 SDK\Samples\FamilyCreation\WindowWizard\CS, load it into Revit to make sure it works good.

Get rid of all UI Stuff:

- Remove References to all the UI related assembly, including RevitAPIUI.dll, System.Windows.dll;
- Check all the .cs file, and remove all the reference to Autodesk.Revit.UI and System.Windows.Forms;
- Remove WizardUI.cs file, which is the main Window UI part to collect all the input parameter, we will get all these input parameter from a JSON file instead later.
- Create a new class CreateWindowData in namespace Revit.SDK.Samples.WindowWizard.CS to replace Autodesk.Revit.UI.ExternalCommandData, the code should look as follow, and then replace all the usage of ExternalCommandData to CreateWindowData.

```
using Autodesk.Revit.ApplicationServices;
public class CreateWindowData
{
    public Application Application { get; set; }
}
```



AUTODESK®

Make anything™

Autodesk and the Autodesk logo are registered trademarks or trademarks 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 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.

 AUTODESK® FORGE

© 2019 Autodesk. All rights reserved.