Revit API Intro Labs

Lab1 – Hello World

Created by M. Harada, July 2010
Updated by DevTech AEC WG
Last modified: 10/5/2023

<C#>C# Version</C#>

Objective: In this lab, we will learn the basics of Revit API. We'll learn how to:

- Implement an External Command or a command level add-in
- Implement an External Application or an application level add-in
- Apply attributes to your external command or application definitions
- Write an add-in manifest file to register your external command or application with Revit
- Access a Revit model through the External Command Data

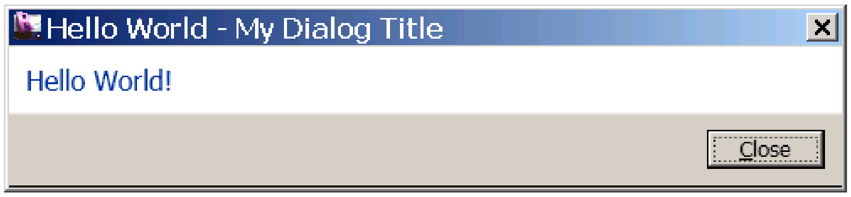
In addition, you will also have a chance to familiarize yourself with the following tools:

- RevitLookup
- Add-In Manager

Tasks: We'll write two commands and one application in this lab:

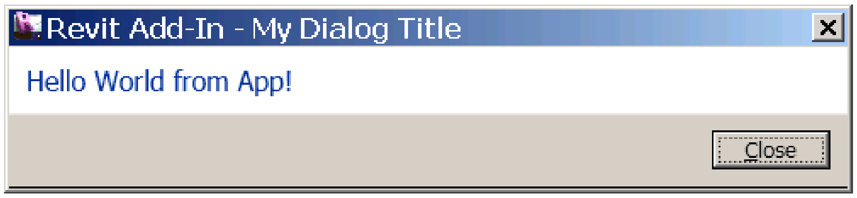
1. "**Hello World**" – a minimum external command that pops up a dialog with a message "Hello World". The command will run from [Add-Ins] tab >> [External Tools] panel.
2. "**Hello World App**" – a minimum external application that pops up a dialog with a message "Hello World from App." The dialog will pops up automatically when Revit starts up.
3. "**Command Data**" – an external command that looks at ExternalCommandData or the first argument of the IExternalCommand.Execute() method, and writes a few information from it.

Figure 1 to 3 shows the sample images from running the commands and application that you will be defining in this lab's exercises:


Hello World - My Dialog Title

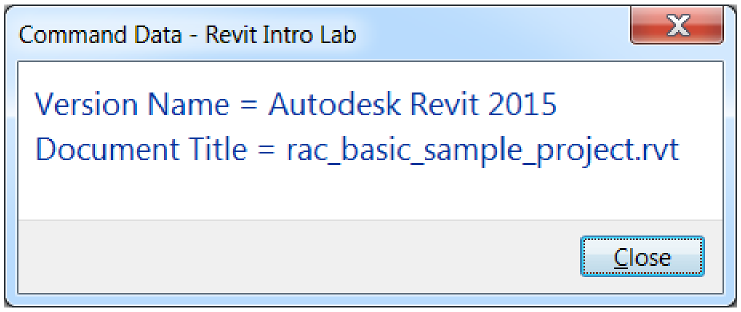
Hello World!

Figure 1. "Hello World" command

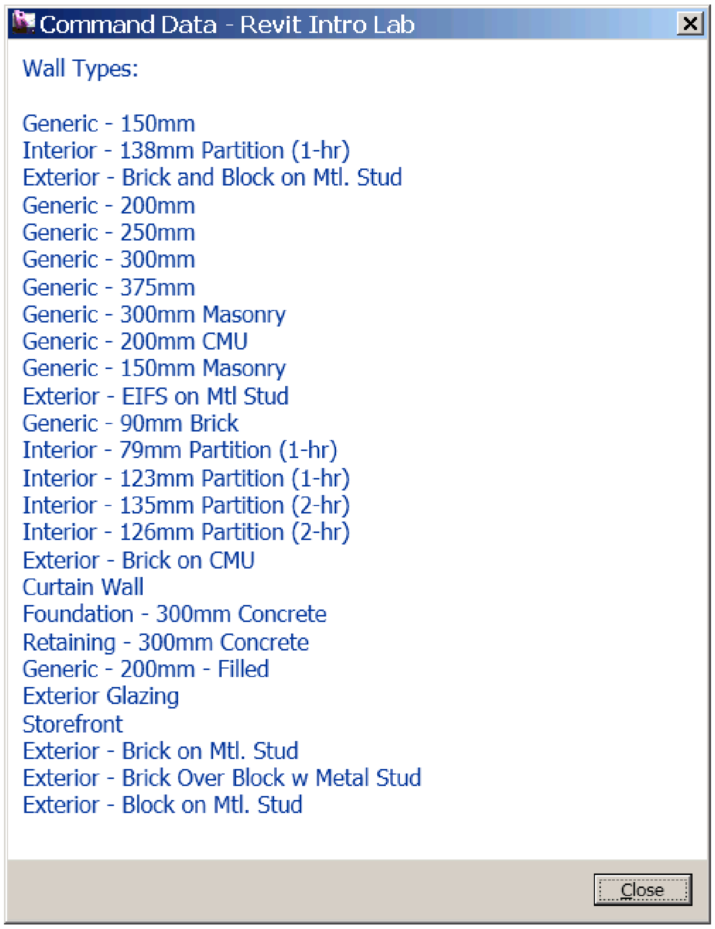


```
Hello World from App!
```

Figure 2. "Hello World App" application



```
Version Name = Autodesk Revit 2015
Document Title = rac_basic_sample_project.rvt
```



```
Wall Types:

Generic - 150mm
Interior - 138mm Partition (1-hr)
Exterior - Brick and Block on Mtl. Stud
Generic - 200mm
Generic - 250mm
Generic - 300mm
Generic - 375mm
Generic - 300mm Masonry
Generic - 200mm CMU
Generic - 150mm Masonry
Exterior - EIFS on Mtl Stud
Generic - 90mm Brick
Interior - 79mm Partition (1-hr)
Interior - 123mm Partition (1-hr)
Interior - 135mm Partition (2-hr)
Interior - 126mm Partition (2-hr)
Exterior - Brick on CMU
Curtain Wall
Foundation - 300mm Concrete
Retaining - 300mm Concrete
Generic - 200mm - Filled
Exterior Glazing
Storefront
Exterior - Brick on Mtl. Stud
Exterior - Brick Over Block w Metal Stud
Exterior - Block on Mtl. Stud
```

Figure 3. "Command Data" command reports some information from ExternalCommandData.

The following is the breakdown of step by step instructions in this lab:

1. "Hello World": an External Command
2. Add-in Manifest File
3. Testing Your Command
4. "Hello World" with Simplified Namespaces
5. "Hello World App": an External Application.
6. Command Data and Revit Object Model

1. "Hello World": an External Command

A Revit add-in is defined as a **Class Library** in the Microsoft Visual Studio 2017 (MSVS). If you are already familiar with defining a class library using MSVS, you can continue with the section 1.1 below.

If you are not yet familiar with MSVS or class library, Revit API Developer Guide on Autodesk Help provides good walkthroughs for a Hello World in both VB.NET and C#.

If this is an instructor-lead, classroom training, your instructor will demonstrate for you.

1.1 Create a new Visual Studio project using Class Library template with the language of your choice (VB.NET or C#). Let's name them as follows:

- Solution name: **IntroCs (or IntroVb)**
- Project name: **IntroCs (or IntroVb)**
- File name: **1_HelloWorld.cs (or .vb)**
- Command class name: **HelloWorld** (We'll define this shortly.)

(You may choose to use any names you want here. When you do so, just remember what you are calling your own project, and substitute these names as needed while reading the instruction in this document.)

Required References:
Add references. We will need the following reference at least:

- System.dll
- System.Core.dll (this is for LINQ query)
- RevitAPI.dll (this is in <Revit install>\ folder)
- RevitAPIUI.dll (same as above)

Make sure to set "Copy Local" to False when you reference them.

Required Namespaces:

In addition to the default setting, add the following namespaces to your projects:

- System.Linq
- Autodesk.Revit.DB
- Autodesk.Revit.UI
- Autodesk.Revit.Attributes
- Autodesk.Revit.ApplicationServices (for Revit Application)

Note: about VB.NET vs. C# behavior difference
If you are using VB.NET, you can set the above namespaces under: [Project] >> [Properties] >> [References] >> [Imported namespaces]. In fact, we recommend setting namespaces in the project properties for VB.NET project instead of using "Imports" keywords in each .vb file. Otherwise, you may encounter some error messages like the following:

>> '<class name>' is not accessible in this context because it is 'Friend'.

This error can be avoided if you use project level import settings.
C# does not have the same option of importing namespaces at project level. You will need to use "using" keyword in each .cs files to import namespaces. i.e.,

```
using System;
using Autodesk.Revit.DB;
using Autodesk.Revit.UI;
using Autodesk.Revit.Attributes;
```

1.2 Add the following code to 1_HelloWorld.cs (or .vb). This defines a minimum external command in Revit:

<C#>
```
/// <summary>
/// Hello World #1 - A minimum Revit external command.
/// </summary>
[Autodesk.Revit.Attributes.Transaction( Autodesk.Revit.Attributes.Transaction
Mode.ReadOnly )]
public class HelloWorld : Autodesk.Revit.UI.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 Autodesk.Revit.UI.Result.Succeeded;
  }
}
```
</C#>

Our command class is called "HelloWorld" and it is defined as following:

- Command class is derived from `IExternalCommand`.
- You will need to implement `Execute()` method. It takes those three arguments, and returns the Result of this command which is either "Succeeded", "Failed" or "Cancelled."

In this example, we simply display a message using a TaskDialog, which is a Revit style dialog.

Notice a line at the top: Attribute for Transaction. These are not an optional. Without them Revit will not recognize your command.

- Transaction mode – controls the behavior of transaction, either by Automatic (not recommended), Manual or ReadOnly (no transaction).

(Note: The attributes are added since Revit 2011. In Revit 2011, there was Regeneration option – controls whether the API framework automatically regenerates graphics or not, i.e., Automatic or Manual. Since 2012, regeneration is always manual.)

Your code is ready to build at this point. Go ahead and build the code.

2. Add-in Manifest File

Revit uses an add-in manifest file (**.addin**) to register a command and an application. Revit will read manifest files automatically at startup. There are two places where you can place manifest files on your computer: for All Users and for your user specific location.

For Windows 7 and 8:

- C:\ProgramData\Autodesk\Revit\Addins\2022\
- C:\Users\<user>\AppData\Roaming\Autodesk\Revit\Addins\2022\

Locate or create the folder that matches your system and the user requirement. Create a new text file, and name it "**AdnpIntroCs.addin**". Here is the content of the .addin file:

```
<?xml version="1.0" encoding="utf-8"?>
<RevitAddIns>
  <AddIn Type="Command">
    <Text>Hello World</Text>
    <FullClassName>IntroCs.HelloWorld</FullClassName>
    <Assembly>C:\...\ IntroCs.dll</Assembly>
    <AddInId> 8195FDC2-5B44-43D8-9637-51E0967A9562</AddInId>
    <VendorId>ADNP</VendorId>
    <VendorDescription>Autodesk, Inc. www.autodesk.com</VendorDescription>
  </AddIn>
</RevitAddIns>
```

This includes information about:

- Type of your add-in, command or application
- Text that appears under [Add-Ins] tab >> [External Tools] panel
- Full class name including namespaces
- Full path to the dll or assembly module
- GUID or a unique identifier of your command.
- Vendor Id - your Registered Developer Symbol (RDS)
- Vendor Description - name of your company

Make necessary adjustment to match with your environment, i.e., the namespaces and the name of the class, and full path and name of the assembly. (e.g., the above example shows with CS suffix.) This developer page has a table that describes each tag and more tag options. You can use a tool to produce a new GUID, e.g. Microsoft Visual Studio > Tools > Create GUID.

3. Testing Your Command

Once you have successfully built your dll and have the .addin manifest in place, you are ready to test your command.

1. Start Revit. Start a new project. You should see [Add-Ins] tab. ([Add-Ins] is not visible when there is not add-ins registered.)
2. Under [External Tools] panel, you should see your command [Hello World]. (Figure 4.)

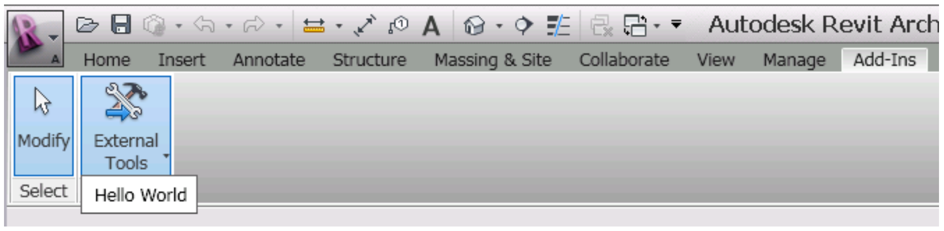


Figure 4. Your "Hello World" command should be under [External Tools] of [Add-Ins] tab.

3. Run the command. You should see a dialog greeting "Hello World! (Figure 1).

Congratulations! You have just created your first Revit External Command.

4. "Hello World" with Simplified Namespaces

Let's go back to our "Hello World" code once again. In our earlier "Hello World" code, we have used full namespaces. We have done so intentionally because we wanted to give you an idea about structure and kinds of namespaces that we are using. Once you get the basics idea, you may want to write code without spelling out full namespaces. That will certainly save your typing and make your code more readable. Here is the simplified version of "Hello World":

<C#>
```
// Hello World #2 - simplified without full namespace.
//
```

```
[Transaction(TransactionMode.ReadOnly)]
public class HelloWorldSimple : IExternalCommand
{
    public Result Execute(
        ExternalCommandData commandData,
        ref string message,
        ElementSet elements)
    {
        TaskDialog.Show("My Dialog Title", "Hello World Simple!");

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

Make sure that you have imported necessary namespaces to make this work. (And if you are writing in VB.NET, make sure you import namespaces in the project property.)

You can try simplifying the code that you wrote earlier and try building it again.

5. "Hello World App": an External Application

Now that we learned how to write a minimum External Command and how to register it with Revit, let's take a look at another type of Revit Add-Ins, External Application. You can use an External Application when you want to add a functionality that needs to be executed at application level, such as when Revit starts up or when shutting down. For example, if you want to add your own Ribbon panel to the Add-Ins tab, you will call a function to add a Ribbon panel when Revit starts.

5.1 Add code for "Hello World App" External Application

The following is the minimum code for an External Application:

<C#>
```
// Hello World #3 - minimum external application
public class HelloWorldApp : IExternalApplication
{
    // OnShutdown() - called when Revit ends.
    public Autodesk.Revit.UI.Result OnShutdown(
        Autodesk.Revit.UI.UIControlledApplication application)
    {
        return Result.Succeeded;
    }

    // OnStartup() - called when Revit starts.
    public Autodesk.Revit.UI.Result OnStartup(
        Autodesk.Revit.UI.UIControlledApplication application)
    {
        TaskDialog.Show("My Dialog Title", "Hello World from App!");
        return Result.Succeeded;
    }
}
</C#>
```

Notice that this time we are deriving a class from **IExternalApplication**. There are two functions that you can override:

- **OnStartup**() – is called when Revit starts up
- **OnShutdown**() – is called when Revit shuts down

For this exercise, we simply display a dialog with "Hello World from App!" message to the user when the Revit starts up.

You can add the code to your current document (1_HelloWorld.vb or .cs). Build your project. Make sure your code come out clean without any build errors.

5.2 Add Add-in Manifest File for an External Application

Here is an add-in manifest for an external application:

```
  <AddIn Type="Application">
    <Name>Hello World App</Name>
    <FullClassName>IntroCs.HelloWorldApp</FullClassName>
    <Assembly>C:\...\IntroCs.dll</Assembly>
    <AddInId>49E64932-F4DA-4053-B795-3B4B7AE5C12C</AddInId>
    <VendorId>ADNP</VendorId>
    <VendorDescription>Autodesk, Inc. www.autodesk.com</VendorDescription>
  </AddIn>
```

Notice that the add-in type is "Application", and use "Name" instead of "Text". Other than that, tags are the same as External Command. You add this to the existing AsdkRevitInto.addin file after your previous definition of <Addin></AddIn>, which now look like the following. You will need to adjust it to fit your environment, of course:

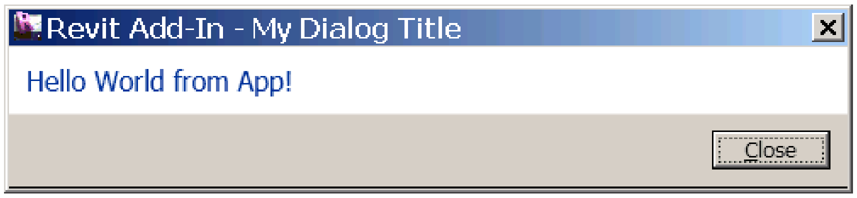
```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<RevitAddIns>
  <AddIn Type="Command">
    <Text>Hello World</Text>
    <FullClassName>IntroCs.HelloWorld</FullClassName>
    <Assembly> C:\...\IntroCs\RevitIntroVB.dll</Assembly>
    <AddInId>0B997216-52F3-412a-8A97-58558DC62D1E</AddInId>
  </AddIn>

  <AddIn Type="Application">
    <Name>Hello World App</Name>
    <FullClassName>IntroCs.HelloWorldApp</FullClassName>
    <Assembly>C:\...\IntroCs.dll</Assembly>
    <AddInId>49E64932-F4DA-4053-B795-3B4B7AE5C12C </AddInId>
    <VendorId>ADNP</VendorId>
    <VendorDescription>Autodesk, Inc. www.autodesk.com</VendorDescription>
  </AddIn>
```

```
</RevitAddIns>
```

Alternatively, you can create a separate .addin file if you prefer.

5.3 Test Your Application.

You are ready to test your application. Simply start Revit. You should see a message “Hello World from App” automatically popping up when Revit starts (figure 5)


Revit Add-In - My Dialog Title

Hello World from App!

Figure 5. A dialog message from your external application at Revit startup.

6. Command Data and Revit Object Model

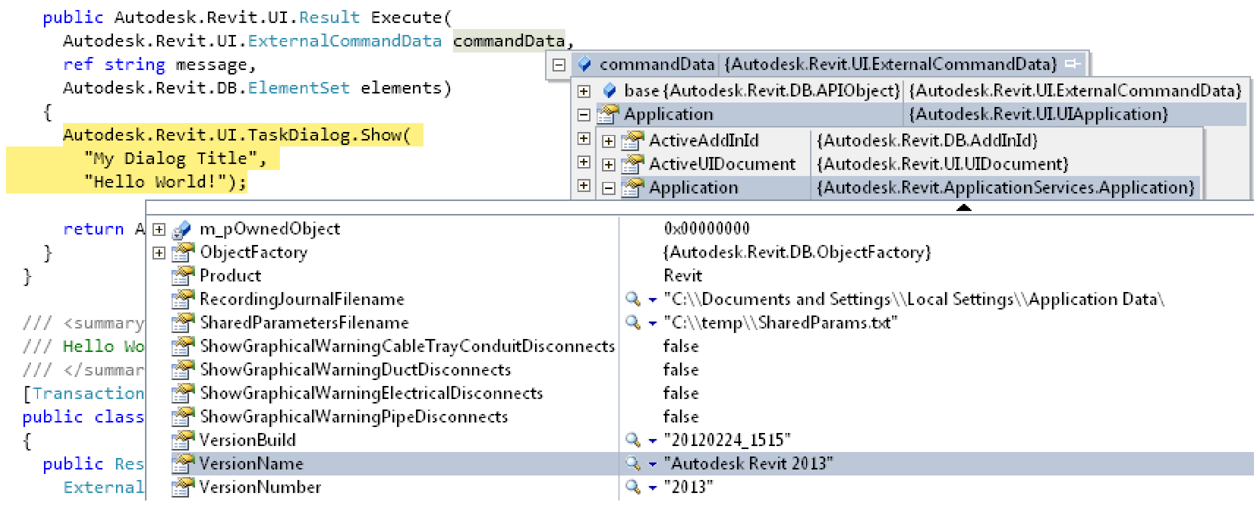
Now that we have understand how to define an external command and application in Revit, let’s go back to our external command and look at the command arguments for `Execute()` method more closely. You see three arguments passed through `Execute()`:

1. commandData (ExternalCommandData) – the top most object and entry point to access Revit model (Input)
2. message (String) – a message to the user when the command fails (Output)
3. elements (ElementSet) – a list of elements you want to highlight when a command fails (Output)

6.1 Examine commandData in the Debugger
The most important object of all is the commandData. Accessing to the Revit model starts here. To see what’s in there, place a breakpoint somewhere in the “Hello World”, for example, at the line of TaskDialog.Show(), and run the command again. When you hit the breakpoint, take a look at commandData. You can drill down from the commandData and see what kind of information is accessible from there. For example, by “drilling down” the hierarchy,

commandData >> Application >> Application >> VersionXxx

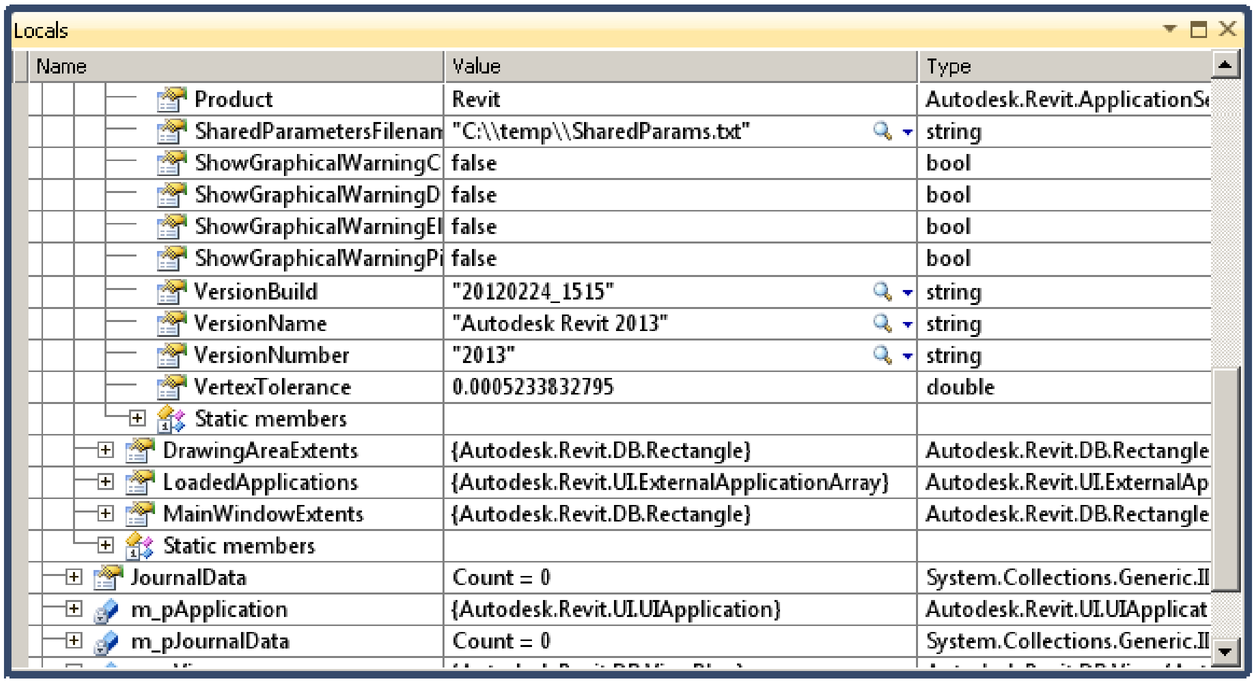
you can obtain the version name, version number and built number of Revit that you are currently running. Figure 6 and 7 below show sample images from a drill down in a code and in the Locals view.



```
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 A
  }
}

/// <summary
/// Hello Wo
/// </summar
[Transaction
public class
{
  public Res
    External
```

```
commandData {Autodesk.Revit.UI.ExternalCommandData}
  base {Autodesk.Revit.DB.APIObject}   {Autodesk.Revit.UI.ExternalCommandData}
  Application                          {Autodesk.Revit.UI.UIApplication}
    ActiveAddInId                      {Autodesk.Revit.DB.AddInId}
    ActiveUIDocument                   {Autodesk.Revit.UI.UIDocument}
    Application                        {Autodesk.Revit.ApplicationServices.Application}
      m_pOwnedObject                                  0x00000000
      ObjectFactory                                   {Autodesk.Revit.DB.ObjectFactory}
      Product                                         Revit
      RecordingJournalFilename                        "C:\\Documents and Settings\\Local Settings\\Application Data\
      SharedParametersFilename                        "C:\\temp\\SharedParams.txt"
      ShowGraphicalWarningCableTrayConduitDisconnects false
      ShowGraphicalWarningDuctDisconnects             false
      ShowGraphicalWarningElectricalDisconnects       false
      ShowGraphicalWarningPipeDisconnects             false
      VersionBuild                                    "20120224_1515"
      VersionName                                     "Autodesk Revit 2013"
      VersionNumber                                   "2013"
```

Figure 6. Examining commandData in the code



Locals

Name	Value	Type
Product	Revit	Autodesk.Revit.ApplicationSe
SharedParametersFilenan	"C:\\temp\\SharedParams.txt"	string
ShowGraphicalWarningC	false	bool
ShowGraphicalWarningD	false	bool
ShowGraphicalWarningEl	false	bool
ShowGraphicalWarningPi	false	bool
VersionBuild	"20120224_1515"	string
VersionName	"Autodesk Revit 2013"	string
VersionNumber	"2013"	string
VertexTolerance	0.0005233832795	double
Static members		
DrawingAreaExtents	{Autodesk.Revit.DB.Rectangle}	Autodesk.Revit.DB.Rectangle
LoadedApplications	{Autodesk.Revit.UI.ExternalApplicationArray}	Autodesk.Revit.UI.ExternalAp
MainWindowExtents	{Autodesk.Revit.DB.Rectangle}	Autodesk.Revit.DB.Rectangle
Static members		
JournalData	Count = 0	System.Collections.Generic.II
m_pApplication	{Autodesk.Revit.UI.UIApplication}	Autodesk.Revit.UI.UIApplicat
m_pJournalData	Count = 0	System.Collections.Generic.II

Figure 7. Examining commandData in Locals view

Let's define a new command to show a few data from the commandData.

6.2 Define a new external command class.

Let's name your new external command class as "CommandData", for example.

- Command class name: **CommandData**

Using the debugger, explore what kind of information you can obtain from the commandData. Write a code to print out some information found there. (Note: At this point, we haven't looked at the detail of objects that are specific to Revit API. Please keep it simple. Try choosing data that is easy to printout, such as ones that represented as a String data.)

The following is example of the code where we print out version information and names of wall types in the Revit file:

<C#>

```
// Command Arguments and Revit Object Model
  [Transaction(TransactionMode.ReadOnly)]
  public class CommandData : IExternalCommand
  {
    public Result Execute(
      ExternalCommandData commandData,
      ref string message,
      ElementSet elements)
    {
      // The first argument, commandData, provides access to the top most object model.
      // You will get the necessary information from commandData.
      // To see what's in there, print out a few data accessed from commandData
      //
      // Exercise: Place a break point at commandData and drill down the data.

      UIApplication rvtUiApp = commandData.Application;
      Application rvtApp = rvtUiApp.Application;
      UIDocument rvtUiDoc = rvtUiApp.ActiveUIDocument;
      Document rvtDoc = rvtUiDoc.Document;

      // Print out a few information that you can get from commandData
      string versionName = rvtApp.VersionName;
      string documentTitle = rvtDoc.Title;

      TaskDialog.Show(
        "Revit Intro Lab",
        "Version Name = " + versionName
        + "\nDocument Title = " + documentTitle);

      // Print out a list of wall types available in the current rvt project:

      FilteredElementCollector collector = new FilteredElementCollector(rvtDoc);
      collector.OfClass(typeof(Autodesk.Revit.DB.WallType));

      string s = "";
      foreach (WallType wallType in collector)
      {
        s += wallType.Name + "\r\n";
      }

      // Show the result:

      TaskDialog.Show(
        "Revit Intro Lab",
        "Wall Types (in main instruction):\n\n" + s);

      // 2nd and 3rd arguments are when the command fails.
      // 2nd - set a message to the user.
      // 3rd - set elements to highlight.

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

Build the code. Make sure it was compiled without any error.

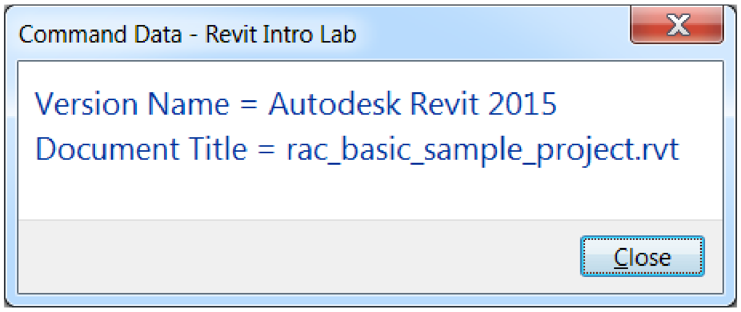
6.3 Add an add-in manifest file

You must know how add-in manifest file works by now (cf. section 2. "Add-In Manifest File" above.) Here is an example:

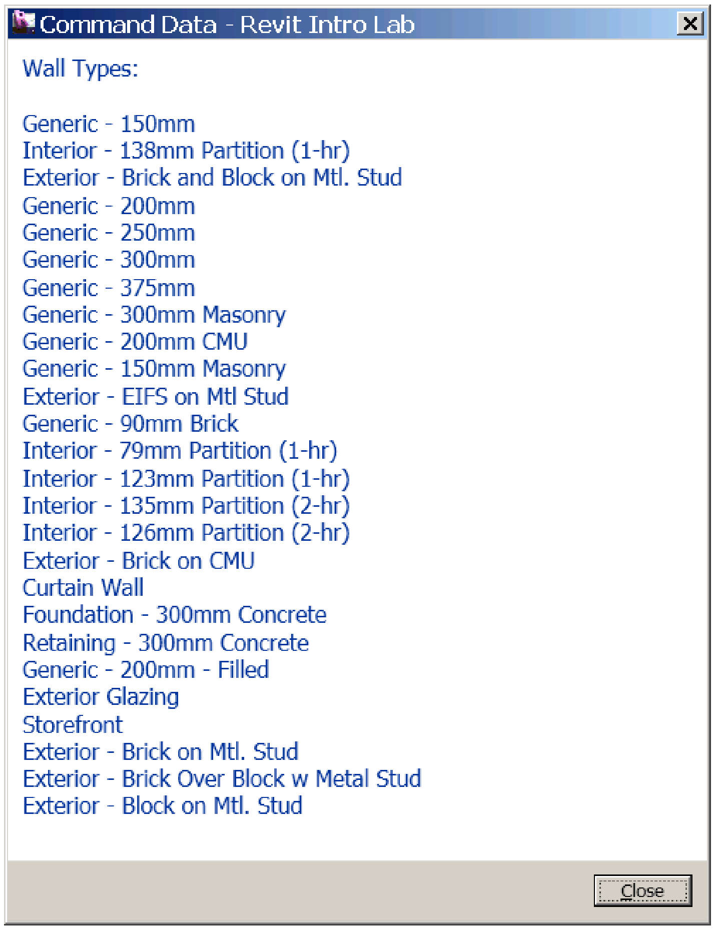
```
  <AddIn Type="Command">
    <Text>Command Data</Text>
    <FullClassName> IntroCs.CommandData</FullClassName>
    <Assembly>C:\...\IntroCs.dll</Assembly>
    <AddInId>AA90426A-E8AA-45b4-84AF-861E865871AE</AddInId>
    <VendorId>ADNP</VendorId>
    <VendorDescription>Autodesk, Inc. www.autodesk.com</VendorDescription>
  </AddIn>
```

6.4 Run and test the command

If everything goes well, you should see dialogs showing some information obtained from the commanddata when you run "Command Data". Figure 8 below shows images from the sample execution when running the command using Revit project with DefaultMetric template.



Version Name = Autodesk Revit 2015
Document Title = rac_basic_sample_project.rvt



```
Wall Types:

Generic - 150mm
Interior - 138mm Partition (1-hr)
Exterior - Brick and Block on Mtl. Stud
Generic - 200mm
Generic - 250mm
Generic - 300mm
Generic - 375mm
Generic - 300mm Masonry
Generic - 200mm CMU
Generic - 150mm Masonry
Exterior - EIFS on Mtl Stud
Generic - 90mm Brick
Interior - 79mm Partition (1-hr)
Interior - 123mm Partition (1-hr)
Interior - 135mm Partition (2-hr)
Interior - 126mm Partition (2-hr)
Exterior - Brick on CMU
Curtain Wall
Foundation - 300mm Concrete
Retaining - 300mm Concrete
Generic - 200mm - Filled
Exterior Glazing
Storefront
Exterior - Brick on Mtl. Stud
Exterior - Brick Over Block w Metal Stud
Exterior - Block on Mtl. Stud
```

Figure 8. “Command Data” command reports some information from ExternalCommandData.

6. Summary

Congratulations!! You have completed your first lab.

In this lab, you have learned the basics of Revit API. We have learned how to:

- Implement an External Command or a command level add-in
- Implements an External Application or an application level add-in
- Apply attributes to your external command or application definitions
- Write an add-in manifest file to register your external command or application with Revit
- Access to a Revit model through the External Command Data

In the next lab, we will take a look at each Revit element more closely and learn how Revit element is represented in the product.

Tools

Before we go to the next lab, we want to point to a few tools that will be useful when learning and debugging Revit API.

- **RevitLookup** – this is a tool that allows you to “snoop” into the Revit database structure. You can find it in the SDK folder. We’ll be using it through out this lab. This is the “must have” for any Revit API programmers.
- **Add-In Manager** – this tools allows you to load and unload your dll while running Revit. You can find this in the SDK folder.
- **SDKSamples2022.sln** – this solution file, found in the /Samples/ folder, is provided to make it easier to build all the sample projects at once. There are more than 100 samples in the Revit SDK now. It will be a tedious task if you want to all the build samples. This becomes handy when you want to build and explore many samples. In conjunction with SDKSamples2022.sln, **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** – this is one of samples found in the /Samples/ folder. This sample integrates all other samples excluding ExternalApplications into a panel named “RvtSamples” under menu “Add-Ins”. Convenient to have for quickly test verious SDK samples.

If this is an instructer-lead classroom training, your instructer may demonstrate them. Please take a look at them and have at least RevitLookup installed for the subsequent labs.