SD323658

Getting Started on Design Automation for Revit on Forge

Rahul Bhobe
Autodesk

Jason Kunkel
CADD Microsystems
[Speaker Company]

Learning Objectives

- Learn how to modify a Revit add-in to work with Design Automation
- Learn how to manage Revit links and family files on Design Automation
- Learn how to get different results from the same add-in with the use of JSON parameters
- Learn how to debug your cloud application with log messages

Description

At Autodesk, we know there is a need for our customers to bring their work to the cloud for collaboration and automation at scale. If you're a Revit API developer and would like to get started on Forge Design Automation for Revit, this class will cover the full basic workflow. We'll walk through a simple example of a Revit add-in, turning it into a Design Automation application. We'll go through all of the steps necessary to go from the desktop to Forge. This will include code samples as well as a working end-to-end example of both desktop code and Design Automation code. We'll teach you the syntax for using Design Automation. We'll show you common errors and how to avoid them. We'll explain how to troubleshoot problems. You'll walk away knowing all of the building blocks and have a solid starting point for automating your processes on Forge. Come learn from someone who is part of the team that built Design Automation for Revit.

Speaker(s)

Rahul Bhobe
He holds a B. Tech degree in Naval Architecture from IIT Madras and a masters degree in Software Systems. He has been with Autodesk for 12 years working on several features of Revit. Currently he is working on developing Design Automation for Revit. In the past he has implemented Revit Conceptual Modeling features like Adaptive Components, Point Elements and Divided Surface. He has then worked on several Revit features like Family, Groups, Links and Worksharing.

Jason Kunkel
Jason has worked across the design and technology spectrum of the AEC industry for over 20 years first as an architectural designer for a major mid-Atlantic architecture firm specializing in large, public sector projects then migrating to the IT support world, where he worked to help architects and engineers leverage technology in new and exciting ways, and save time in the process. Currently at CADD Microsystems, Jason has been able to apply his knowledge and experience to help a wider range of customers achieve the same goals. He is one of the founders of RevitRVA, a Revit user group in the central Virginia area, and has a wide array of knowledge and experience with both software and hardware to help companies improve their processes and work more effectively.

Step by Step guide:

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, or Revit 2020: 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 end up with the project in *Design-Automation_Version*.

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.

```
using Autodesk.Revit.ApplicationServices;
using Autodesk.Revit.DB;
using DesignAutomationFramework;
```

2. In the file *DeleteWalls.addin*, change `AddIn Type` from `command` to `DBApplication`

```
<?xml version="1.0" encoding="utf-8"?>
  <RevitAddIns>
    <AddIn Type="DBApplication">
      <Name>DeleteWalls</Name>
      <Assembly>.\DeleteWalls.dll</Assembly>
      <AddInId>d7fe1983-8f10-4983-98e2-c3cc332fc978</AddInId>
      <FullClassName>DeleteWalls.DeleteWallsApp</FullClassName>
      <Description>"Deletes Walls"</Description>
      <VendorId>Autodesk</VendorId>
      <VendorDescription>
      </VendorDescription>
    </AddIn>
  </RevitAddIns>
```

Notes:

Whenever you are converting a Revit add-in in general, make sure that you:

- Remove references to `RevitAPIUI` or any code in the `Autodesk.Revit.UI` namespace. These functions are not available in Design Automation and hence cannot be called.
- Remove references to `WPF`, `Windows Forms`, or any other UI-based libraries.

Step 5 - Convert IExternalApplication or IExternalCommand to IExternalDBApplication

Since there is no UI interaction in Design Automation, you can't use the Revit UI to initiate commands. In order to initiate commands with Design Automation, you must implement `OnStartup` and `OnShutdown` in your add-in. These functions receive a `ControlledApplication` instead of a `UIControlledApplication`. The functions return an `ExternalDBApplicationResult` object:
For this task, in `DeleteWalls.cs`, implement `OnStartup` and `OnShutdown` as shown in the following code block.

```
using Autodesk.Revit.ApplicationServices;
using Autodesk.Revit.DB;
using DesignAutomationFramework;
namespace DeleteWalls
{
```

```
[Autodesk.Revit.Attributes.Regeneration(Autodesk.Revit.Attributes.Regenera
tionOption.Manual)]

[Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.Transacti
onMode.Manual)]
   public class DeleteWallsApp : IExternalDBApplication
   {
      public ExternalDBApplicationResult
OnStartup(Autodesk.Revit.ApplicationServices.ControlledApplication app)
      {
         return ExternalDBApplicationResult.Succeeded;
      }

      public ExternalDBApplicationResult
OnShutdown(Autodesk.Revit.ApplicationServices.ControlledApplication app)
      {
         return ExternalDBApplicationResult.Succeeded;
      }
```

Step 6 - Add an event handler for DesignAutomationReady

`DesignAutomationBridge` defines the event `DesignAutomationReadyEvent`. The Revit engine raises the `DesignAutomationReadyEvent` when it's ready to run your add-in. The event handler is the entry point to your code.

1. Set the success/failure argument to `DesignAutomationReadyEventArgs.Succeeded` so that Design Automation knows your code succeeded.

```
public class DeleteWallsApp : IExternalDBApplication
{
   public ExternalDBApplicationResult
OnStartup(Autodesk.Revit.ApplicationServices.ControlledApplication app)
   {
      DesignAutomationBridge.DesignAutomationReadyEvent +=
HandleDesignAutomationReadyEvent;
      return ExternalDBApplicationResult.Succeeded;
   }
   public void HandleDesignAutomationReadyEvent(object sender,
DesignAutomationReadyEventArgs e)
   {
      e.Succeeded = true;
      DeleteAllWalls(e.DesignAutomationData);
   }
```

2. Modify `DeleteAllWalls` to accept `DesignAutomationData`.

```
public static void DeleteAllWalls(DesignAutomationData data)
{
   if (data == null) throw new ArgumentNullException(nameof(data));

   Application rvtApp = data.RevitApp;
   if (rvtApp == null) throw new InvalidDataException(nameof(rvtApp));

   string modelPath = data.FilePath;
   if (String.IsNullOrWhiteSpace(modelPath)) throw new
InvalidDataException(nameof(modelPath));

   Document doc = data.RevitDoc;
   if (doc == null) throw new InvalidOperationException("Could not open
document.");

   using (Transaction transaction = new Transaction(doc))
   {
      FilteredElementCollector col = new
FilteredElementCollector(doc).OfClass(typeof(Wall));
      transaction.Start("Delete All Walls");
      doc.Delete(col.ToElementIds());
      transaction.Commit();
   }

   ModelPath path =
ModelPathUtils.ConvertUserVisiblePathToModelPath("result.rvt");
   doc.SaveAs(path, new SaveAsOptions());
}
```

3. Delete the method `Execute`, which previously called `DeleteAllWalls`. This is no longer necessary.

During the execution of your add-in, all files you load from the disk or write to the disk must go into the Windows current working directory. In Design Automation for Revit, write access is limited to the current working directory and its children.

Step 7 - Handle failures encountered by Revit

When an add-in runs on Revit, the add-in uses the UI to communicate warnings and errors. Since there is no UI interaction in Design Automation, you must use an alternate strategy to handle failures.

For this tutorial, you will use the default error handler. You don't need to add any code to enable the default error handler because it comes by default with the Design Automation Bridge. The default error handler suppresses warnings and resolves errors automatically by applying the default options. If resolution of an error fails, it rolls back the failed action.

For more information refer Handling Revit Failures .

Step 8 - Build the add-in

- In Visual Studio, rebuild *DeleteWalls.dll*.

You should now have a Design Automation capable Revit add-in.

Additional notes

- Use the Design Automation for Revit Debug Tool to test the add-in locally. The Readme file in the Git repository provides instructions on how to test the add-in. You can also follow a video tutorial on how to use this tool on YouTube.
- Your application cannot use the network or write to any files outside of the current working directory (or a child folder of the working directory). Restrictions on Design Automation for Revit can be found here.
- If step 8 fails, you can download a Design Automation capable version of *DeleteWalls.dll* from here and continue with task 2.

Task 2 – Obtain an Access Token

This task produces a two-legged token with a scope sufficient to authenticate the remaining tasks in this tutorial.

Expected task outcome

By the end of this task, you will know how to obtain a two-legged access token when the Client ID and Client Secret is known.

Endpoints used in this task

You will use the following API endpoint in this task:

Command	Endpoint URL	Description
POST	**/v1/authenticate**	Get a two-legged access token.

- Base URL: **https://developer.api.autodesk.com/authentication**
- More information can be found here.

Step 1 - Create a Forge App

Follow the instructions on Get Started with Forge in Three Steps to create a Forge App for this tutorial. In the *Add Services to Forge* stage select "Design Automation API V3" and "Data Management API".

Step 2 - Get an Access Token

You must use the Client ID and Client Secret of the Forge App from step 1 to obtain an access token. The access token acts as your security credentials, which authenticates the requests you send Forge for the remainder of this tutorial.

- Replace **`CLIENT_ID`** and **`CLIENT_SECRET`** in the following example with the Client ID and Client Secret you obtained in step 1, and send the request.

Request

```
curl -i -X POST \
    'https://developer.api.autodesk.com/authentication/v1/authenticate' \
    -H 'Content-Type: application/x-www-form-urlencoded' \
    -d 'client_id=CLIENT_ID' \
    -d 'client_secret=CLIENT_SECRET' \
    -d 'grant_type=client_credentials' \
    -d 'scope=code:all data:write data:read bucket:create bucket:delete'
```

Response

```
{
  "access_token":"YOUR_ACCESS_TOKEN",
  "token_type":"Bearer",
  "expires_in":3599
}
```

Notes:

- Jot down the access token (indicated by **`YOUR_ACCESS_TOKEN`** in the preceding example) in the response. You use this value for all subsequent requests in this tutorial.
- The access token expires in the number of seconds specified by the **`expires_in`** attribute.
- Although the scope specified in the request is **`code:all`** **`data:write`** **`data:read`** **`bucket:create`** **`bucket:delete`**, Design Automation requires only a scope of **`code:all`**. The scopes **`data:read`** **`bucket:create`** **`bucket:delete`** are for HTTP requests to the Forge Data Management API. These requests are discussed under Task 6 - Prepare Cloud Storage.

Task 3 – Create a Nickname for the Forge App

Forge uses the Client ID that was generated in the previous task to uniquely identify the app you created. The Client ID can be long and cryptic, and hence a source of irritation when you reference data you add to your app.

A Nickname is a way to map a Forge App Client ID to an easy-to-use name that you can use in place of the Client ID.

You can assign a nickname to an app only if there is no data associated with that app. That is why we are creating the nickname before we do anything else with the app.

Expected task outcome

By the end of this task, you will know how to create a nickname to reference an app.

Endpoints used in this task

You will use the following API endpoint in this task:

Command	Endpoint URL	Description
PATCH	**/v3/forgeapps/:id**	Creates/updates the nickname for the current Forge App.

- Base URL: **https://developer.api.autodesk.com/da/us-east/**
- More information can be found here.

Request

```
curl -X PATCH \
  https://developer.api.autodesk.com/da/us-east/v3/forgeapps/me \
  -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
  -H 'Content-Type: application/json' \
```

```
  -d ' {"nickname": "YOUR_NICKNAME"}'
```

Notes:

- `YOUR_ACCESS_TOKEN` is the Access Token returned by the authentication request in the previous task.
- If your Forge App doesn't have any data, you can map the Forge App to another Nickname. The new nickname will overwrite the old one. If your Forge App has data, you cannot `PATCH` a Nickname for your Forge App anymore. This is true even if you have not yet assigned a Nickname for the app. The only way you can assign a Nickname to an app with data is by first calling [DELETE] /forgeapps/me. This deletes all data associated with that app, including the Nickname.
- Nicknames must be globally unique. If the Nickname you provided is already in use, even by someone else, the PATCH request will return a `409 Conflict` error.

Response

```
Date →Fri, 19 Jul 2019 10:08:35 GMT
Strict-Transport-Security →max-age=31536000; includeSubDomains
Via →1.1 58b224f0fcba4846d5699ecad6c6829f.cloudfront.net (CloudFront)
x-amz-apigw-id →dER5lECTIAMFeZQ=
X-Amz-Cf-Id →hgM0PhBQoDdAocVAEZx48V-T1Iyu5MeAwNPVHF4OWDD-y1dBUsVgwA==
X-Amz-Cf-Pop →SEA19
x-amzn-RequestId →2bd002d9-aa0d-11e9-bdcc-db9443c1afd8
X-Amzn-Trace-Id →Root=1-5d3196a3-e2043246cf1cfa27b9d85aee
X-Cache →Miss from cloudfront
Content-Length →0
Connection →keep-alive
```

Task 4 – Upload an AppBundle to Design Automation

An AppBundle is a package of binaries and supporting files that make a Revit add-in.

Expected task outcome

By the end of this task you will be able to:

- Put together an AppBundle.
- Upload an AppBundle to Design Automation.
- Create an alias for the AppBundle.
- Create a new version of the AppBundle.
- Point the alias to the new version of the AppBundle.

Endpoints used in this task

You will use the following API endpoints to handle AppBundles in this task:

Command	Endpoint URL	Description
POST	**/v3/appbundles**	Registers a new AppBundle.
POST	**/v3/appbundles/{id}/aliases**	Creates a new alias for the AppBundle.
POST	**/v3/appbundles/{id}/versions**	Creates a new version of the AppBundle.
PATCH	**/v3/appbundles/{id}/aliases/{aliasId}**	Modify alias details.

- Base URL: **https://developer.api.autodesk.com/da/us-east**
- More information on AppBundles can be found here.

Step 1 - Understand the structure of an AppBundle

A Design Automation for Revit AppBundle file is a zip file that contains specific contents stored according to a specific structure.

Download the example AppBundle for this exercise, DeleteWallsApp.zip, from this repository.

The following code block shows the structure of the AppBundle DeleteWallsApp.zip.

```
DeleteWallsApp.zip
|-- DeleteWalls.bundle
|   |-- PackageContents.xml
|   |-- Contents
|   |   |-- DeleteWalls.dll
|   |   |-- DeleteWalls.addin
```

The top-level folder is named `*.bundle`. This folder contains a file named `PackageContents.xml`. This file contains the details of the AppBundle, the relative path to its add-in file, and its runtime requirements, as shown in the following code block.

```
<?xml version="1.0" encoding="utf-8" ?>
<ApplicationPackage>
  <Components Description="Delete Walls">
    <RuntimeRequirements OS="Win64"
                         Platform="Revit"
                         SeriesMin="R2018"
                         SeriesMax="R2018" />
    <ComponentEntry AppName="DeleteWalls"
                    Version="1.0.0"
                    ModuleName="./Contents/DeleteWalls.addin"
                    AppDescription="Deletes walls"
                    LoadOnCommandInvocation="False"
                    LoadOnRevitStartup="True" />
  </Components>
</ApplicationPackage>
```

`SeriesMin` and `SeriesMax` both refer to Revit 2018 as `R2018`. Design Automation for Revit currently support AppBundles that run on Revit `R2018` , `R2019` and `R2020`. For more information on `PackageContents.xml`, see PackageContents.xml Format Reference

The `*.bundle\Contents` folder contains the add-in file, the application DLL, and its dependencies. The following code block shows the content of *DeleteWalls.addin*.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<RevitAddIns>
  <AddIn Type="DBApplication">
    <Name>DeleteWalls</Name>
    <Assembly>.\DeleteWalls.dll</Assembly>
    <AddInId>d7fe1983-8f10-4983-98e2-c3cc332fc978</AddInId>
    <FullClassName>DeleteWalls.DeleteWallsApp</FullClassName>
    <Description>"Walls Deleter"</Description>
    <VendorId>Autodesk</VendorId>
    <VendorDescription>
    </VendorDescription>
  </AddIn>
</RevitAddIns>
```

Notes:

- **`Type`** must be **`DBApplication`**. Design Automation for Revit doesn't support applications that need Revit's UI functionality. **`Assembly`** must be a relative path to the DLL.

  You can find examples of the *bundle* folder and *PackageContent.xml* file in the presentation on Autodesk Exchange Revit Apps on this site.

  You can use PackageContents.xml from any existing Autodesk Exchange Revit app on Design Automation for Revit. However, Design Automation for Revit reads only the **`RuntimeRequirements`** and **`ComponentEntry`** blocks, which are circled in the image shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<ApplicationPackage SchemaVersion="1.0"
                    AutodeskProduct="Revit"
                    ProductType="Application"
                    Name="File Upgrader"
                    AppVersion="2.0.0"
                    Description="ADN Plugin of the Month: File Upgrader"
                    Author="Saikat Bhattacharya"
                    Icon="./Contents/2014/Resources/FileUpgrader_Thumbnail.png"
                    OnlineDocumentation="http://labs.autodesk.com/utilities/ADN_Plugins"
                    HelpFile="./Contents/2014/Resources/ADNFileUpgraderHelp.htm"
                    ProductCode="{F23B85C8-D5DE-45B9-977E-D860120D27B1}"
                    UpgradeCode="{5D9F89AD-3CC0-4769-B90D-60BFB4EE90DB}"
                    FriendlyVersion="2.0.0"
                    SupportedLocales="Enu"
                    AppNameSpace="appstore.exchange.autodesk.com">

  <CompanyDetails Name="Autodesk"
                  Url="http://labs.autodesk.com/utilities/ADN_Plugins"
                  Email="adn.apps@autodesk.com"
                  Phone=" " />

  <RuntimeRequirements OS="Win32|Win64"
                       Platform="Revit|Revit Architecture|Revit Structure|Revit MEP"
                       SeriesMin="R2014"
                       SeriesMax="R2014" />

  <Components Description="2014 parts">
    <RuntimeRequirements OS="Win32|Win64"
                         Platform="Revit|Revit Architecture|Revit Structure|Revit MEP"
                         SeriesMin="R2014"
                         SeriesMax="R2014" />
    <ComponentEntry AppName="FileUpgrader"
                    Version="2.0.0"
                    ModuleName="./Contents/2014/ADNPlugin-FileUpgrader.addin"
                    AppDescription="FileUpgrader" />
  </Components>

</ApplicationPackage>
```


The source code and dependent library associated with this AppBundle are:

- DeleteWalls sample
- DesignAutomationBridge

Step 2 - Register the AppBundle

Before you upload the AppBundle to Design Automation, you must register the AppBundle.

- Register an AppBundle named **`DeleteWallsApp`** as per the following example. It uses Revit 2018 as the target engine:

Request

```
curl -X POST \
  https://developer.api.autodesk.com/da/us-east/v3/appbundles \
  -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
  -H 'Content-Type: application/json' \
  -d '{
  "id": "DeleteWallsApp",
  "engine": "Autodesk.Revit+2018",
  "description": "Delete Walls AppBundle based on Revit 2018"
}'
```

Attribute	Description
`id`	The name given to the AppBundle.
`engine`	The engine used by the AppBundle.

Response

```
{
    "uploadParameters": {
        "endpointURL": "https://dasprod-store.s3.amazonaws.com",
        "formData": {
            "key": "apps/Revit/DeleteWallsApp/1",
            "content-type": "application/octet-stream",
            "policy": "eyJleHBpcmF0aW9uIjoiMjAxOC... (truncated)",
            "success_action_status": "200",
            "success_action_redirect": "",
            "x-amz-signature": "6c68268e23ecb8452... (truncated)",
            "x-amz-credential": "ASIAQ2W... (truncated)",
            "x-amz-algorithm": "AWS4-HMAC-SHA256",
            "x-amz-date": "20180810... (truncated)",
            "x-amz-server-side-encryption": "AES256",
```

```
            "x-amz-security-token": "FQoGZXIvYXdzEPj//////////wEaDHavu...
(truncated)"
        }
    },
    "engine": "Autodesk.Revit+2018",
    "description": "Delete Walls AppBundle based on Revit 2018",
    "version": 1,
    "id": "YOUR_NICKNAME.DeleteWallsApp"
}
```

Attribute	Description
`endpointURL`	**This is the URL you must upload your AppBundle zip file to.**
`version`	The version number for the AppBundle created by the POST request. For new AppBundles, the returned version is always **`1`**.
`formData`	The form data that needs to accompany your AppBundle upload. The formData expires in **`3600`** seconds.

Step 3 - Upload the AppBundle

Upload the AppBundle to the signed URL returned by **`endpointURL`** in the previous step.

```
curl -X POST \
  https://dasprod-store.s3.amazonaws.com \
  -H 'Cache-Control: no-cache' \
  -F key=apps/Revit/DeleteWallsApp/1 \
  -F content-type=application/octet-stream \
  -F policy=eyJleHBpcmF0aW9uIjoiMjAxOC... (truncated) \
  -F success_action_status=200 \
  -F success_action_redirect= \
  -F x-amz-signature=6c68268e23ecb8452... (truncated) \
  -F x-amz-credential=ASIAQ2W... (truncated) \
  -F x-amz-algorithm=AWS4-HMAC-SHA256 \
  -F x-amz-date=20180810... (truncated) \
  -F x-amz-server-side-encryption=AES256 \
  -F 'x-amz-security-token=FQoGZXIvYXdzEPj//////////wEaDHavu...
(truncated)' \
  -F 'file=@path/to/your/app/zip'
```

Note: Ensure that all the form-data from the create AppBundle response is included in your request.

Step 4 - Create an alias for the AppBundle

When you registered the AppBundle in step 2, it was registered as version 1 of the AppBundle. In this step, you create an alias named `test` to reference that version.

```
curl -X POST \
  https://developer.api.autodesk.com/da/us-
east/v3/appbundles/DeleteWallsApp/aliases \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
  -d '{
      "version": 1,
      "id": "test"
    }'
```

Step 5 - Update an existing AppBundle

To update an existing AppBundle, you must register a new version for the AppBundle and then upload the updated AppBundle for that version. If you try to overwrite an existing AppBundle, Design Automation for Revit returns a `409 Conflict` error.

To register a new version of the AppBundle DeleteWallsApp:

Request

```
curl -X POST \
  https://developer.api.autodesk.com/da/us-
east/v3/appbundles/DeleteWallsApp/versions\
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
  -d '{
      "id": null,
      "engine": "Autodesk.Revit+2018",
      "description": "Delete Walls AppBundle based on Revit 2018 Update"
    }'
```

Note: You can omit `id` from the request body. If you include `id` in the request body, set it to `null`. If you don't set it to `null`, Design Automation for Revit returns an error.

Response


```
{
    "uploadParameters": {
        "endpointURL": "https://dasprod-store.s3.amazonaws.com",
        "formData": {
            "key": "apps/Revit/DeleteWallsApp/2",
            "content-type": "application/octet-stream",
            "policy": "eyJleHBpcmF0aW9uIjoiMjAxOC... (truncated)",
            "success_action_status": "200",
            "success_action_redirect": "",
            "x-amz-signature": "6c68268e23ecb8452... (truncated)",
            "x-amz-credential": "ASIAQ2W... (truncated)",
            "x-amz-algorithm": "AWS4-HMAC-SHA256",
            "x-amz-date": "20180810... (truncated)",
            "x-amz-server-side-encryption": "AES256",
            "x-amz-security-token": "FQoGZXIvYXdzEPj//////////wEaDHavu...
(truncated)"
        }
    },
    "engine": "Autodesk.Revit+2018",
    "description": "Delete Walls AppBundle based on Revit 2018",
    "version": 2,
    "id": "YOUR_NICKNAME.DeleteWallsApp"
}
```


The response to the AppBundle version post includes:

Attribute	Description
This is the signed URL to which you must upload the updated AppBundle.	
The new version number for the AppBundle created by the above POST request.	

Step 6 - Upload the updated AppBundle

Follow the procedure outlined in Step 3 to upload the AppBundle to the signed URL returned by `endpointURL`.

Step 7 - Assign an existing alias to the updated AppBundle

Currently, the alias *test* points to version 1 of the AppBundle.

id	alias	version
DeleteWallsApp	test	1
DeleteWallsApp		2

You can reassign the alias *test* to version 2 of the AppBundle DeleteWallsApp.

id	alias	version
DeleteWallsApp		1
DeleteWallsApp	test	2

To reassign the alias, you can either:

- Delete the existing alias and recreate it to point to the desired version.
- Send a PATCH request.

To send a PATCH request:

Request

```
curl -X PATCH \
https://developer.api.autodesk.com/da/us-
east/v3/appbundles/DeleteWallsApp/aliases/test \
-H 'Content-Type: application/json' \
```

```
-H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
-d '{
            "version": 2
        }'
```

Notes:

- `version` - Refers to the version number the alias must point to.
- You can use this technique to make sure that an alias always points to the latest version of an AppBundle.

Additional notes

- Each AppBundle POST request specifies an `engine` on which the application runs. The following table shows the keywords to specify for the available Revit engines.
- The `engine` must be match the `SeriesMin` and `SeriesMax` settings specified in the AppBundle's PackageContent.xml.
- The active engine version aliases are:

Engine	Description	JSON in AppBundle post	DesignAutomationBridge DLL
`Autodesk.Revit+2018`	Revit 2018.3.3	"engine": "Autodesk.Revit+2018"	DesignAutomationBridge.dll for 2018.
`Autodesk.Revit+2019`	Revit 2019.2.	"engine": "Autodesk.Revit+2019"	DesignAutomationBridge.dll for 2019.
`Autodesk.Revit+2020`	Revit 2020.1	"engine": "Autodesk.Revit+2020"	DesignAutomationBridge.dll for 2020.

Notes:

- The Endpoints for `engines` are listed here.
- If there is an error, refer the section on troubleshooting.

Task 5 – Publish an Activity

An Activity is an action that can be executed in Design Automation. You create and post Activities to run specific AppBundles.

Expected task outcome

By the end of this task, you will know:

- What an Activity is.
- How to create an Activity.
- How to create new versions of an Activity.
- How to reference a specific version of an Activity by an alias.

Endpoints used in this task

You will use the following API endpoints to handle Activities for this task:

Command	Endpoint URL	Description
POST	**/v3/activities**	Creates a new Activity.
POST	**/v3/activities/{id}/aliases**	Creates a new alias for this Activity.
POST	**/v3/activities/{id}/versions**	Creates a new version of the Activity.
PATCH	**/v3/activities/{id}/aliases/{aliasId}**	Modifies alias details.

- Base URL: **https://developer.api.autodesk.com/da/us-east**

- More Endpoints for Activities can be found here.

Step 1 - Create a new Activity

To create a new Activity named DeleteWallsActivity, post this request:

Request

```
curl -X POST \
  https://developer.api.autodesk.com/da/us-east/v3/activities \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
  -d '{
            "id": "DeleteWallsActivity",
            "commandLine": [ "$(engine.path)\\\\revitcoreconsole.exe /i
$(args[rvtFile].path) /al $(appbundles[DeleteWallsApp].path)" ],
            "parameters": {
              "rvtFile": {
                "zip": false,
                "ondemand": false,
                "verb": "get",
                "description": "Input Revit model",
                "required": true,
                "localName": "$(rvtFile)"
              },
              "result": {
                "zip": false,
                "ondemand": false,
                "verb": "put",
                "description": "Results",
                "required": true,
                "localName": "result.rvt"
              }
            },
            "engine": "Autodesk.Revit+2018",
            "appbundles": [ "YOUR_NICKNAME.DeleteWallsApp+test" ],
            "description": "Deletes walls from Revit file."
    }'
```

Attribute	Description
`id`	The name given to your new Activity.
`commandLine`	The command run by this Activity. - `$(engine.path)\\\\revitcoreconsole.exe` - The full path to the folder from which the engine for Revit executes. The engine is defined in the request body as `"engine": "Autodesk.Revit+2018"`. More information about engines can be found in the previous step. **Do not edit or alter this "commandLine" in the request body of Activity posts.** - `$(args[rvtFile].path)` - The full path to the folder that contains the input Revit model. `rvtFile` is the parameter that represents the Revit model to which the Activity `DeleteWallsActivity` applies the AppBundle. The AppBundle is defined in the request body as `"appbundles": [ "YOUR_NICKNAME.DeleteWallsApp\ +test" ]`.
`parameters`	The parameters that must be passed to this Activity, when it is executed (via a WorkItem). You will specify values for these parameters in task 6, at the time you submit a WorkItem to execute this Activity
`engine`	The engine on which your Activity runs. The available engine versions are described in the Additional notes section in Task 4.

Attribute	Description
`appbundles`	The fully qualified id of the AppBundle that this Activity applies to the input rvt file. It is currently defined as `[ "YOUR_NICKNAME.DeleteWallsApp\ +test" ]`, where `YOUR_NICKNAME` represents the Client ID of the Forge App the AppBundle `DeleteWallsApp` was uploaded to.

Response

```
{
    "commandLine": [
        "$(engine.path)\\\\revitcoreconsole.exe /i $(args[rvtFile].path)
/al $(appbundles[DeleteWallsApp].path)"
    ],
    "parameters": {
        "rvtFile": {
            "verb": "get",
            "description": "Input Revit model",
            "required": true,
            "localName": "$(rvtFile)"
        },
        "result": {
            "verb": "put",
            "description": "Results",
            "required": true,
            "localName": "result.rvt"
        }
    },
    "engine": "Autodesk.Revit+2018",
    "appbundles": [
        "YOUR_NICKNAME.DeleteWallsApp+test"
    ],
    "description": "Delete walls from Revit file.",
    "version": 1,
    "id": "YOUR_NICKNAME.DeleteWallsActivity"
}
```

The response includes:

Attribute	Description
`version`	The version number for the Activity created by the post request. A post request that creates a new Activity will get version number `1`.

Step 2 - Create an alias to the Activity

Design Automation does not let you reference an Activity by its `id`. You must always reference an Activity by an alias. Note that an alias points to a specific version of an Activity and not the Activity itself.
To create an alias named `test`, which refers to version `1` of the `DeleteWallsActivity`:

Request

```
curl -X POST \
  https://developer.api.autodesk.com/da/us-
east/v3/activities/DeleteWallsActivity/aliases \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
  -d '{
      "version": 1,
      "id": "test"
    }'
```

Response

```
{
    "version": 1,
    "id": "test"
}
```

Step 3 - Update an existing Activity

Design Automation does not let you overwrite an Activity once you have created it. If you want to modify/update an existing Activity, you must update it as a new version. If you try to overwrite an existing Activity, Design Automation for Revit returns a `409 Conflict` error.

To create a new version of an Activity:

Request

```
curl -X POST \
 https://developer.api.autodesk.com/da/us-
east/v3/activities/DeleteWallsActivity/versions \
 -H 'Content-Type: application/json' \
 -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
 -d '{
        "id": null,
        "commandLine": [ "$(engine.path)\\\\revitcoreconsole.exe /i
$(args[rvtFile].path) /al $(appbundles[DeleteWallsApp].path)" ],
        "parameters": {
          "rvtFile": {
            "zip": false,
            "ondemand": false,
            "verb": "get",
            "description": "Input Revit model",
            "required": true,
            "localName": "$(rvtFile)"
          },
          "result": {
            "zip": false,
            "ondemand": false,
            "verb": "put",
            "description": "Results",
            "required": true,
            "localName": "result.rvt"
          }
        },
        "engine": "Autodesk.Revit+2018",
        "appbundles": [ "YOUR_NICKNAME.DeleteWallsApp+test" ],
        "description": "Delete walls from Revit file Updated."
}'
```

Note: You can omit `id` from the request body. If you include `id` in the request body, set it to `null`. If you don't set it to `null`, Design Automation for Revit returns an error.

Response

```
{
    "commandLine": [
        "$(engine.path)\\\\revitcoreconsole.exe /i $(args[rvtFile].path)
/al $(appbundles[DeleteWallsApp].path)"
    ],
    "parameters": {
        "rvtFile": {
            "verb": "get",
            "description": "Input Revit model",
            "required": true,
```

```
            "localName": "$(rvtFile)"
        },
        "result": {
            "verb": "put",
            "description": "Results",
            "required": true,
            "localName": "result.rvt"
        }
    },
    "engine": "Autodesk.Revit+2018",
    "appbundles": [
        "YOUR_NICKNAME.DeleteWallsApp+test"
    ],
    "description": "Delete walls from Revit file Updated.",
    "version": 2,
    "id": "YOUR_NICKNAME.DeleteWallsActivity"
}
```

Step 4 - Assign an existing alias to the updated Activity

Currently, the alias *test* points to version 1 of the Activity.

id	alias	version
DeleteWallsActivity	test	1
DeleteWallsActivity		2

You can reassign the alias *test* to point to version 2 of the Activity.

id	alias	version
DeleteWallsActivity		1
DeleteWallsActivity	test	2

To update the alias, you can either:

- Delete the existing alias and recreate it with the version you want to label.
- Send a PATCH request.

To send a PATCH request:

Request

```
curl -X PATCH \
https://developer.api.autodesk.com/da/us-
east/v3/activities/DeleteWallsActivity/aliases/test \
-H 'Content-Type: application/json' \
-H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
-d '{
        "version": 2
        }'
```

Notes:

- **`version`** - Refers to the version number the alias labels.

Additional notes

By default Design Automation runs the Revit engine in English. However, you can request Design Automation to run the Revit engine in a specific language using the **`/l`** specifier on **`commandLine`**. The list of supported languages are listed in [this article] for the full list of language codes.

The following example instructs Design Automation to launch the Revit engine in German. This feature is useful when you want the built-in elements to be labelled in German.

```
{
    "commandLine": [
        "$(engine.path)\\\\revitcoreconsole.exe /i $(args[rvtFile].path)
/al $(apps[DeleteWallsApp].path) /l DEU"
    ],
...
}
```

Task 6 – Prepare Cloud Storage

When you execute an Activity (via a WorkItem) Design Automation takes a Revit file as input, processes it, and uploads the result to a remote server. While you can use any cloud storage service, in this task. you use the Data Management API to access the Forge Object Storage Service (OSS) to store the input Revit file as well as the output Revit file.

The OSS is a cloud storage service that uses "Buckets" as containers of data. The input and output files are stored as "Objects" in a Bucket. For more information on the OSS, refer the Data Management API documentation.

Expected task outcome

By the end of this task, you will be able to:

- Create a Bucket to store the input and output files.
- Upload a file to a Bucket.
- Obtain a temporary signed URL that enables Design Automation to download or upload a file.

Endpoints used in this task

You will use the following API endpoints in this task:

Command	Endpoint URL	Description
POST	**/buckets**	Creates an OSS Bucket.
PUT	**/buckets/:bucketKey/objects/:objectName**	Uploads a file to a Bucket.
PUT	**/buckets/:bucketKey/objects/:objectName/signed**	Create a signed url.

- Base URL: **https://developer.api.autodesk.com/oss/v2**
- More information on endpoints for Buckets and Objects can be found here.

Step 1 - Create a Bucket

The first thing to do when using the OSS service is to create a Bucket. Once a Bucket is created, you can store the input and output files inside of it as objects.

Request

```
curl -X POST \
    'https://developer.api.autodesk.com/oss/v2/buckets' \
    -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
    -H 'Content-Type: application/json' \
        -d '{
            "bucketKey": "YOUR_BUCKET_KEY",
            "access": "full",
            "policyKey": "transient"
        }'
```

Notes:

- You must specify a name for your Bucket. Replace `YOUR_BUCKET_KEY` with the name you chose.
- The Bucket Key must be unique throughout all of the OSS service. If the Bucket Key is already in use (even by another user) Forge returns a `409 Conflict` error. In such a case, retry with another name.
- Bucket Keys must consist of only lower case characters, numbers 0-9, and the underscore (_) character

Response

```
{
    "bucketKey": "YOUR_BUCKET_KEY",
    "bucketOwner": "YOUR_FORGE_APP_CLIENT_ID",
    "createdDate": 156095829931,
    "permissions": [
        {
            "authId": "YOUR_FORGE_APP_CLIENT_ID",
            "access": "full"
        }
    ],
    "policyKey": "transient"
}
```

Step 2 - Upload input file to OSS

Once the Bucket is created, you must upload the input file to the Bucket.

1. Download the file *DeleteWalls.rvt* from https://github.com/Autodesk-Forge/forge-tutorial-postman/blob/master/DA4Revit/tutorial_data and zip it up.
2. Send a HTTP request to Forge, to upload the file to the Bucket.

Request

```
curl -X PUT \

'https://developer.api.autodesk.com/oss/v2/buckets/YOUR_BUCKET_KEY/objects
/OBJECT_KEY_4_INPUT_FILE' \
    -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
    -H 'accept-encoding: gzip, deflate' \
    --data-binary '@PATH_TO_YOUR_FILE_TO_UPLOAD'
```

Notes:

- You will need to replace **`PATH_TO_YOUR_FILE_TO_UPLOAD`** with the path to the zip file you created earlier.
- Pay attention to the URI. - You must replace the URI Parameter **`YOUR_BUCKET_KEY`** with the Bucket Key you specified in the previous step. - Replace **`OBJECT_KEY_4_INPUT_FILE`** with a name that you will use to refer to the input file. - Object keys must consist of only lower case characters, numbers 0-9, and the underscore (_) character.

Response

```
{
    "bucketKey": "YOUR_BUCKET_KEY",
    "objectId":
"urn:adsk.objects:os.object:YOUR_BUCKET_KEY/OBJECT_KEY_4_INPUT_FILE",
    "objectKey": "OBJECT_KEY_4_INPUT_FILE",
    "sha1": "1e0c312608ece2a21d6ecd3c316e40ebec574011",
    "size": 55998,
    "contentType": "text/plain",
    "location":
"https://developer.api.autodesk.com/oss/v2/buckets/YOUR_BUCKET_KEY/objects
/YOUR_OBJECT_KEY"
```

```
}
```

Step 3 - Get temporary download URL

The next step is to create a temporary url for the object you created in step 2. Design Automation will use this temporary URL to download the input file.

Request

```
curl -X POST \

'https://developer.api.autodesk.com/oss/v2/buckets/YOUR_BUCKET_KEY/objects
/OBJECT_KEY_4_INPUT_FILE/signed' \
    -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
    -H 'Content-Type: application/json' \
    -d '{}'
```

Notes:

- Pay attention to the URI. - You must replace the URI Parameter `YOUR_BUCKET_KEY` with the Bucket Key you specified at the time you created the Bucket. - You must replace the URI Parameter `OBJECT_KEY_4_INPUT_FILE` with the Object Key you specified in the previous step.

Response

```
{
    "signedUrl":
"https://developer.api.autodesk.com/oss/v2/signedresources/fd41e212-0fa1-
4ef0-91fe-64373a17bdb1?region=US",
    "expiration": 1560964242534,
    "singleUse": false
}
```

Note down the value you for *signedUrl*. You will use this value (referred to as `URL_TO_INPUT_FILE`) when you submit a WorkItem in Task 7. The URL expires in 1 hour.

Step 4 - Get temporary upload URL

You must get a URL to enable Design Automation upload the results.

Request

```
curl -X POST \

'https://developer.api.autodesk.com/oss/v2/buckets/YOUR_BUCKET_KEY/objects
/OBJECT_KEY_4_OUTPUT_FILE/signed?access=readwrite' \
    -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
    -H 'Content-Type: application/json' \
    -d '{}'
```

Notes:

- Pay attention to the URI. You must replace the URI Parameter **`YOUR_BUCKET_KEY`** with the Bucket Key you specified in the previous step.
- Replace **`OBJECT_KEY_4_OUTPUT_FILE`** with a name that you will use to refer to the output file.

Response

```
{
    "signedUrl":
"https://developer.api.autodesk.com/oss/v2/signedresources/4bf02d31-0b39-
1706-4lch-73r341?region=US",
    "expiration": 1560970633351,
    "singleUse": false
}
```

Note down the value you for *signedUrl*. You will use this value (referred to as **`SIGNED_URL_TO_RESULT`**) when you submit a WorkItem in Task 7.

Task 7 – Submit a WorkItem

When you post a WorkItem to Design Automation, you are instructing Design Automation to execute an Activity.

The relationship between an Activity and a WorkItem can be thought of as a “function definition” and “function call”, respectively. Named parameters of the Activity have corresponding named arguments of the WorkItem. Like in function calls, optional parameters of the Activity can be skipped and left unspecified while posting a WorkItem.

Expected task outcome

By the end of this task, you will be able to:

- Create a WorkItem to execute an Activity.
- Check if execution succeeded or failed.
- Get the URL to the execution log file.

Endpoints used in this task

You will use the following API endpoints to work with WorkItems in this task:

Command	Endpoint URL	Description
POST	**/v3/workitems**	Creates a new WorkItem and queues it for processing.
GET	**/v3/workitems/{id}**	Gets the status of a specific WorkItem.

- Base URL: **https://developer.api.autodesk.com/da/us-east**
- For more information on these endpoints see the API Reference.

Step 1 - Create a WorkItem

To create a work item to execute the Activity DeleteWallsActivity:

Request

```
curl -X POST \
  https://developer.api.autodesk.com/da/us-east/v3/workitems \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
  -d '{
        "activityId": "YOUR_NICKNAME.DeleteWallsActivity+test",
        "arguments": {
          "rvtFile": {
            "url": "SIGNED_URL_TO_INPUT_FILE",
            "pathInZip": "PATH_TO_RVT_FILE_WITHIN_ZIP_FILE"
          },
          "result": {
            "verb": "put",
            "url": "SIGNED_URL_TO_RESULT"
          }
        }
      }'
```

Note

Attribute	Description
`activityId`	The target Activity defined by "owner.activity+alias"(**`YOUR_NICKNAME.DeleteWallsActivity+test`**) this WorkItem will execute.
`arguments`	The argument list that is required by the Activity (**`DeleteWallsActivity`**): - **`rvtFile`** - The URL to get the input file that will be processed by the WorkItem.

Attribute	Description
	- `result` - The URL to which the output must be “put” (uploaded).

The response contains the `id` of the posted WorkItem:

```
{
    "status": "pending",
    "stats": {
        "timeQueued": "2018-04-16T21:45:08.1357163Z"
    },
    "id": "YOUR_WORKITEM_ID"
}
```

Step 2 - Check status of a WorkItem

Design Automation WorkItems are queued before they are processed. A WorkItem’s processing time will vary depending on the size and complexity of the input files, the type of processing done by the AppBundle, and the size of the output files.

In this tutorial, you will be checking the WorkItem status to see if it has completed. However, the best practice is to use the `onComplete` argument when submitting a WorkItem. This onComplete argument enables you to specify a callback URL, which will be called once the WorkItem is completed. For more information on the `onComplete` argument see “Output arguments: onComplete callback” under the Additional notes section below.
You can check the status of a WorkItem by calling `[GET] /workitems/{id}`:

Request

```
curl -X GET \
  https://developer.api.autodesk.com/da/us-
east/v3/workitems/YOUR_WORKITEM_ID \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer YOUR_ACCESS_TOKEN'
```

Response

```
{
    "status": "success",
    "reportUrl":  "https://dasprod-
store.s3.amazonaws.com/workItem/Revit/YOUR_WORKITEM_ID/report.txt?XXXXXXXX
X",
```

```
    "stats": {
        "timeQueued": "2018-04-13T03:15:15.9772282Z",
        "timeDownloadStarted": "2018-04-13T03:15:17.2960823Z",
        "timeInstructionsStarted": "2018-04-13T03:15:20.2803318Z",
        "timeInstructionsEnded": "2018-04-13T03:15:41.6075799Z",
        "timeUploadEnded": "2018-04-13T03:15:42.0450494Z"
    },
    "id": "YOUR_WORKITEM_ID"
}
```

Attribute	Description
`status`	Indicates if execution is pending, successful, failed or cancelled.
`reportUrl`	The URL to get the report log for this WorkItem's execution.

Additional notes

Input arguments: Embedded JSON

If an input argument of an Activity requires JSON values, the JSON values can be embedded in the WorkItem itself.

For example, the Activity `CountItActivity` requires a parameter named `countItParams`. The Activity expects the argument value to be a JSON file. The WorkItem is able to embed the JSON values in the WorkItem itself as shown below. By prefixing those values with `data:application/json`, you instruct the Design Automation framework to handle them as a JSON stream and save them as a JSON file:

Request

```
curl -X POST \
  https://developer.api.autodesk.com/da/us-east/v3/workitems \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
  -d '{
          "activityId": "YOUR_NICKNAME.CountItActivity+test",
          "arguments": {
            "rvtFile": {
```

```
              "url": "https://s3.amazonaws.com/revitio-dev/test-
data/CountIt.rvt"
            },
            "countItParams": {
              "url": "data:application/json,{'walls': false, 'floors':
true, 'doors': true, 'windows': true}"
            },
            "result": {
              "verb": "put",
              "url": "SIGNED_URL_TO_RESULT"
            }
          }
      }'
```

Input arguments: eTransmit files

Design Automation is capable of processing outputs from eTransmit for Revit, so long as you first create a zip file from those outputs:

Request

```
curl POST \
  https://developer.api.autodesk.com/da/us-east/v3/workitems \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
  -d '{
        "activityId" : "YOUR_NICKNAME.CountItActivity+test",
        "arguments": {
          "rvtFile": {
            "url": "https://s3.amazonaws.com/revitio-dev/test-
data/TopHost.zip"
            "pathInZip": "CountIt.rvt"
          },
          "countItParams": {
            "url": "data:application/json,{'walls': true, 'floors': true,
'doors': true, 'windows': true}"
          },
          "result": {
            "verb": "put",
            "url": "SIGNED_URL_TO_RESULT"
          }
        }
    }'
```

A sample eTransmit file **`TopHost.zip`** is available at TopHost.zip.

The name of the “Root Model” is read from the manifest file. The root model is then found in the zip.

Host RVT file with linked models

Request

```
curl POST \
  https://developer.api.autodesk.com/da/us-east/v3/workitems \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
  -d '{
        "activityId": "YOUR_NICKNAME.CountItActivity+test",
        "arguments": {
          "rvtFile": {
            "url": "https://s3.amazonaws.com/revitio-dev/test-
data/TopHost.rvt",
            "references": [
              {
                "url": "https://s3.amazonaws.com/revitio-dev/test-
data/LinkA.rvt",
                "references": [
                  {
                    "url": "https://s3.amazonaws.com/revitio-dev/test-
data/LinkA1.rvt"
                  },
                  {
                    "url": "https://s3.amazonaws.com/revitio-dev/test-
data/LinkA2.rvt"
                  }
                ]
              },
              {
                "url": "https://s3.amazonaws.com/revitio-dev/test-
data/LinkB.rvt"
              }
            ]
          },
          "countItParams": {
            "url": "data:application/json,{'walls': true, 'floors': true,
'doors': true, 'windows': true}"
          },
          "result": {
            "verb": "put",
            "url": "https://myWebsite/signed/url/to/result"
          }
        }
```

```
    }'
```

The root model in this example is **TopHost.rvt** and it contains **LinkA.rvt** and **LinkB.rvt**. The file **LinkA.rvt** in turn contains **LinkA1.rvt** and **LinkA2.rvt**. Each file is uploaded to a cloud location and the path is provided for each individually:

```
TopHost.rvt
|-- LinkA.rvt
|   |-- LinkA1.rvt
|   |-- LinkA2.rvt
|
|-- LinkB.rvt
```

RvtLinks in sub-folders

The WorkItem's **localName** variable can be used to create a folder structure inside the working directory. For example, a Revit file **Host.rvt** containing a relative link **SubFolder/Link.rvt** can be defined in this way for **rvtFile** in the WorkItem:

```
{
  "url": "https://s3.amazonaws.com/revitio-dev/test-
data/TestForSubFolders/Host.rvt",
  "references": [
    {
      "url": "https://s3.amazonaws.com/revitio-dev/test-
data/TestForSubFolders/Link.rvt",
      "localName": "SubFolder/Link.rvt"
    }
  ]
}
```

This will create the directory/file structure in the current working directory (CWD):

```
{CWD}/Host.rvt
{CWD}/SubFolder/Link.rvt
```

Because you are not allowed to create a folder structure outside of your current working directory, if the host file has linked files with relative paths like **../ParallelFolder/Link.rvt**, you can move the entire structure down one level by creating a top level folder of your own. The same **localName** variable can be used for the top host link you use for linked files. Here is an example json:

```
{
 "url": "https://path/to/Host.rvt",
 "localName": "TopFolder/Host.rvt",
 "references": [
   {
     "url": "https://path/to/Link.rvt",
     "localName": "ParallelFolder/Link.rvt"
   }
```

```
 ]
}
```

This will create the directory/file structure:

```
{CWD}/TopFolder/Host.rvt
{CWD}/ParallelFolder/Link.rvt
```

Input arguments: Zip file

To increase download speed, Design Automation provides the ability to use a zip file for input arguments. To specify the path to the host Revit file, use the `"pathInZip":` option:

```
"arguments": {
  "rvtFile": {
    "url": "https://s3.amazonaws.com/revitio-dev/test-data/CountIt.zip",
    "pathInZip": "CountIt.rvt"
  },
}
```

Notes:

- When providing a zip file as input argument, the `"pathInZip"` option can be specified. In our example, we specified `CountIt.rvt`:

```
CountIt.zip
|-- CountIt.rvt
...
```

Output arguments: onComplete callback

Each WorkItem supports a special output argument named `onComplete`. When provided, Design Automation calls the callback URL when it completes processing the WorkItem. Here is an example of how to call `[POST] /workitems` with the `onComplete` argument added to the example in Step 1:

Example Request

```
curl -X POST \
  https://developer.api.autodesk.com/da/us-east/v3/workitems \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer YOUR_ACCESS_TOKEN' \
  -d '{
        "activityId": "YOUR_NICKNAME.DeleteWallsActivity+test",
        "arguments": {
          "rvtFile": {
            "url": "https://s3.amazonaws.com/revitio-dev/test-
data/DeleteWalls.rvt"
```

```
          },
          "result": {
            "verb": "put",
            "url": "SIGNED_URL_TO_RESULT"
          },
          "onComplete": {
            "verb": "post",
            "url": "https://myWebsite/callback"
          }
        }
      }'
```

This argument is optional for the `[POST] /workitems` call.
Once the WorkItem is processed, the specified URL is called with a payload identical to the response received on `[GET] /workitems/{id}` call.

The implementation of the callback URL is similar to how you implement a callback URL for the Webhooks API. Refer to the Webhooks API documentation for information on specifying the callback URL. Additional documentation for configuring local server is also available.

Notes:

- Revit 2018 supports `Open IFC` and `Export IFC` functionality.
- Revit 2019-2020 supports `Open IFC` , `Export IFC` and `Link IFC` functionality.

Decoding Activity Definition

```
{
    "id": "DeleteWallsActivity",
    "commandline": [ "$(engine.path)\\revitcoreconsole.exe /i $(args[rvtFile].path)
/al $(appbundles[DeleteWallsBundle].path)" ],
    "parameters": {
      "rvtfile": {
        "zip": false,
        "ondemand": false,
        "verb": "get",
        "description": "input revit model",
        "required": true
      },
      "result": {
        "zip": false,
        "ondemand": false,
        "verb": "put",
        "description": "results",
        "required": true,
        "localName": "result.rvt"
      }
    },
    "engine": "Autodesk.Revit+2018",
    "appbundles": [ "Revit.DeleteWallsBundle+prod" ],
    "description": "Delete walls from Revit file."
}
```

The activity definition defines the `commandline` field. This is essentially the command that will run on the worker machine. Currently we provide only one built in executable `revitcoreconsole.exe` for a given engine version.

This executable takes an optional input value `/i $(args[rvtFile].path)`. Such an argument will expand at runtime to the value of the file path saved on disk on the worker machine for the workitem argument `rvtFile`.

A required argument of `/al $(appbundles[DeleteWallsBundle].path)` needs to be provided for the console executable to succeed. Currently we only allow one appbundle per excecution.

For each parameter specified in the activity definition, a corresponding argument with the same name needs to be provided while posting the workitem. However, if the parameter is defined with the field `required = false`, then the corresponding workitem argument can be skipped.

A `verb = get` indicates an input argument and the corresponding url will be called before the execution of the job to download the corresponding file. The HTTP methods of such urls need to be of the kind `GET`. For arguments, `verb = put` or `verb = post` can be specified. Such urls will be called after the execution of the job. The HTTP methods should be of the kind `PUT` or `POST` and must match the corresponding `verb` of the parameter.

The value of `localName` field indicates the filename on the disk saved in the current working folder at the time of the execution of the job. For input arguments, the files are downloaded to this location and for output arguments the file is chosen from this location to be uploaded to the url provided.

Report log

Each workitem execution generates a text based report log. The link to this report log is provided in the response to `GET workitem` endpoint. Anything printed to the standard output (console) will be captured in this log. As an example for `C#` based appbundle, one can log messages with a call to `System.Console.WriteLine()`.

Such logging strategy could be used to debug/troubleshoot any problems you might have in your code.

Workshared files

To open workshared files the following strategy could be used:

1) Do not specify the optional `/i $(args[rvtFile].path` in the command line. So we shall not open the file for you.

```
 "commandLine": [ "$(engine.path)\\\\revitcoreconsole.exe /al
$(appbundles[DeleteWallsBundle].path)" ],
```

2) In the activity parameter specify a hard coded local name for your input Revit file

```
 "rvtFile": {
    "zip": false,
    "ondemand": false,
    "verb": "get",
    "description": "Input Revit model",
```

```
  "required": true,
  "localName": "input.rvt"
 }
```

3) In your app bundle code, open the document from known path input.rvt like so

```
 public void HandleDesignAutomationReadyEvent(object sender,
DesignAutomationReadyEventArgs e)
 {
  var application = e.DesignAutomationData.RevitApp.RevitApp;
  var document = application.OpenDocumentFile(
   ModelPathUtils.ConvertUserVisiblePathToModelPath("input.rvt"),
   new OpenOptions { DetachFromCentralOption =
DetachFromCentralOption.DetachAndDiscardWorksets });
 }
```

We currently do not support live RCW and RCM files. We also do not support local files.

The above strategy will work for central models, published RCW models, etransmitted central models.

Zip files

A combination of activity parameter field zip = true/false and the workitem argument field can determine the behavior of how zip files are handled.

The following table describes the behavior:

Activity	Workitem	Arg direction	Comments
zip==true	pathInZip!=null	input	Zip is uncompressed to the folder specified in localname. Any path reference to this argument will expand to full path of pathInZip.
zip==false	pathInZip!=null	input	Zip is uncompressed to the folder specified in localname. Any path reference to this argument will expand to full path of pathInZip.
zip==true	pathInZip==null	input	If zip is provided then it is uncompressed to the folder specified in localname. Any path reference

			to this argument will expand to full path of localName.
zip==false	pathInZip==null	input	If zip is provided then it is left compressed. Any variable referencing this argument will expand to full path of localName.
zip==true	pathInZip!=null	output	Workitem will be rejected.
zip==false	pathInZip!=null	output	Workitem will be rejected.
zip==true	pathInZip==null	output	Output(s) at localName will be zipped if localName is a folder.
zip==false	pathInZip==null	output	Output at localName will not be zipped.

For a zip package containing an eTransmit manifest file, the path to `RootModel` is used from the manifest file.

OnDemand feature

Conceptually, `onDemand` inputs allow your AppBundle to access additional resources based on the actual run of a given WorkItem, using a url that can be parameterized to query exactly the data you need right now. It can either access additional specific design files on your storage, or call your own server's http API to query for json data etc. Note that onDemand can only be used when the parameter verb is get.

There are several steps you need to take in order to be able to use the `onDemand` input.

1) In the activity, you must specify `ondemand=true` for that input parameter:

```
    "parameters": {
      "myparam": {
        "zip": false,
        "ondemand": true,
        "verb": "get",
        "description": "On demand input",
        "required": true
      }
    },
```

2) Call the workitem with a base url for that argument:

```
    "parameters": {
      "myparam": "https://myhost.com/mypath"
    },
```

3) In your addin code provide the suffix to the url and specify the filename to save the response.

```
Console.WriteLine($"!ACESAPI:acesHttpOperation(myparam,qs?a=1&b=2,,,file://myfile.txt");
```

This will save the response of `https://myhost.com/mypath/qs?a=1&b=2` to the file `myfile.txt`.