FDC226894

Revit Data on Forge – How can Design Automation for Revit API Help Me?

Sasha Crotty
Autodesk

Diane Christoforo
Ryan Duell
Autodesk

Learning Objectives

- Discover and understand ways to access and work with Revit data in the cloud
- Discover the kinds of problems that design automation can solve for your company
- Understand how the Design Automation API for Revit can help manage BIM data challenges
- Learn how you can use these building blocks to automate your company's workflows

Description

The Forge Design Automation API for Revit allows you to build web applications that can create, read, and modify Revit models in the cloud. No longer is access to Revit data trapped on the desktop. Learn how Forge can help you manage and create Revit data in cloud-native applications. We'll review the different kinds of apps you will be able to build using the Design Automation API to solve your company's challenges as well as demonstrate some sample applications using the service.

Speaker(s)

Sasha Crotty joined Autodesk, Inc., in 2005 as a developer for Revit Structure software. She went on to lead the Revit Structure Development Team before switching gears into product management. As the Revit Platform Services product manager, she is responsible for the direction and evolution of Revit's multi-disciplinary tools, Collaboration for Revit, performance, and the Revit API. Sasha holds a BA in Architecture and a BS in Electrical Engineering and Computer Science from the University of California, Berkeley, as well as an MBA from Boston University. In her spare time Sasha enjoys growing miniature orchids and traveling around the world.

Diane Christoforo has been a software developer at Autodesk for thirteen years. She has worked on a variety of aspects of Autodesk Revit, including linked files, Autodesk Revit eTransmit, the API, shared coordinates, and Forge Design Automation for Revit. She graduated with a degree in computer science from MIT in 2005. She square dances in her spare time.

Ryan Duell started his career at an architectural firm in Boston Massachusetts, working on a variety of project teams and functioning in the BIM Manager role. He joined Autodesk in 2008 as a member of the product support organization. From there he transitioned into the development organization,

currently functioning as a QA Analyst and scrum team Product Owner for Revit. He holds a degree in design computing from Boston Architectural College. In his spare time, Ryan enjoys running and playing both current and classic Nintendo games.

In this handout, we wanted to help you understand how to create and use applications with Design Automation. What follows is documentation which we wrote for our private beta customers to help them get up and running. It's accurate as of October 2018 but the system is subject to change and so the described API may not be final.

Design Automation Terminology

Terminology	Description
appbundle	A package of binaries and supporting files which make your Revit Addin application.
activity	An action which can be executed in Design Automation. You create and post your own activities to run against particular `appbundles`.
workitem	A request to execute an `activity`. The relationship between an activity and workitem can be thought of as a "function definition" and "function call", respectively.
nickname	A way to map a Forge App `Client Id` to a customized string. The nickname lets you create a friendly, easy-to-read, string that can be used in place of the long Forge App `Client Id`.
alias	A label to a specific version of an `appbundle` or `activity`.

Using Design Automation: Step by Step

1. Convert your Revit Addin

   (If you don't have an existing addin, Appendix C has samples.)

2. Create a Forge App and get authenticated
3. Create a nickname for your Forge App
4. Publish your Design Automation appbundle
5. Publish your Design Automation activity
6. Post your Design Automation workitem
7. Appendix A: System Restrictions

8. Appendix B: Handling Failures in Revit Appbundles
9. Appendix C: Code sample

1. Convert your Revit Addin

Applications run on Design Automation are very similar to normal Revit addins. The primary difference is that there's no UI in Design Automation. Because of this, your addin will need to be written as an IExternalDBApplication rather than an IExternalApplication or IExternalCommand.

Here are our recommended steps for converting an existing addin.

Start with a small subset of your code

We recommend you start with a simple operation as a proof of concept. Converting an ExternalCommand can be a good starting point.

Referencing the DesignAutomationBridge DLL

We'll provide a small library (currently called DesignAutomationBridge) that you will use to interface with Design Automation. Add it as a dependency to your project. (Note: this is not available while Revit Design Automation is in private beta.)

Convert your IExternalApplication or IExternalCommand to IExternalDBApplication

You won't be adding any buttons or ribbon commands, since there won't be any UI interaction.

You will need to implement `OnStartup` and `OnShutdown`. These functions will get a `ControlledApplication` instead of a `UIControlledApplication`. The functions return an `ExternalDBApplicationResult` object.

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

[Autodesk.Revit.Attributes.Regeneration(Autodesk.Revit.Attributes.RegenerationOption.
Manual)]

[Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manu
al)]
   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;
      }
```

The .addin file can go in the normal place, but the addin type is `DBApplication`.

- Don't include references to RevitAPIUI! (Don't include WPF or Windows Forms or anything either, but we do not currently have a way to check this.) There's no UI interaction, so anything that pops up a dialog expecting user input will hang the system.

Add a reference to DesignAutomationBridge.dll and add an event handler for DesignAutomationReady

Note: DesignAutomationBridge is not available while Design Automation for Revit is in private beta. The "test your app on your local machine" instructions explain how to mimic its behavior with desktop Revit.

Add a reference `DesignAutomationBridge.dll`. This is the library which Design Automation will use to communicate with your addin.
For a C# project in Visual Studio, you can add a reference by opening the Solution Explorer, finding your C# project, expanding its contents, right-clicking on the References node and doing "Add Reference..."
In the Reference Manager dialog, use the "Browse..." button to browse to DesignAutomationBridge.dll. Click "Add" and then "OK" to add the reference to your project.
The `DesignAutomationBridge` defines an event `DesignAutomationReadyEvent`. Revit's engine will raise this event when it's ready for you to run your addin. You should execute your code inside the event handler.

```
   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);
      }
```

The event will give you a path `DesignAutomationData.MainModelPath` to the "main" model indicated in the WorkItem's arguments. There is also a success/failure argument `DesignAutomationReadyEventArgs.Succeeded` you should set; it will let the service know whether potential failures happened in your code or elsewhere.
Any files you load or create should be put into the working directory. On the cloud your write access is limited to the working directory and its children.

Handle failures encountered by Revit

A fundamental feature in Revit is how warnings and errors (collectively referred to as "failures") are handled. Understand your options for handling failures in Revit and implement a failure handling strategy in your application. We will cover failure handling options in more detail in Appendix B.

Test your app on your local machine

For local debugging, there isn't a DesignAutomationReady event to handle. However, you can get similar behavior by watching for the ApplicationInitialized event.

Do not use this event on the cloud, because Design Automation for Revit continues doing setup past the point at which `ApplicationInitialized` is raised. Locally it should mimic the "run automatically" behavior. For example, in our `DeleteWalls` example, we can do this:

```
   public class DeleteWallsApp : IExternalDBApplication
   {
      public ExternalDBApplicationResult
OnStartup(Autodesk.Revit.ApplicationServices.ControlledApplication app)
      {
         //Stop handling the event used by jobs on the cloud:
         //DesignAutomationBridge.DesignAutomationReadyEvent +=
HandleDesignAutomationReadyEvent;
         // And instead execute the code when desktop Revit is initialized.
         app.ApplicationInitialized += HandleApplicationInitializedEvent;
         return ExternalDBApplicationResult.Succeeded;
      }

      //public void HandleDesignAutomationReadyEvent(object sender,
DesignAutomationReadyEventArgs e)
      //{
      //   e.Succeeded = true;
```

```
      //   DeleteAllWalls(e.DesignAutomationData);
      //}

      public void HandleApplicationInitializedEvent(object sender,
Autodesk.Revit.DB.Events.ApplicationInitializedEventArgs e)
      {
         Autodesk.Revit.ApplicationServices.Application app = sender as
Autodesk.Revit.ApplicationServices.Application;
         // We don't need to provide the file
         DesignAutomationData data = new DesignAutomationData(app,
"/path/to/file.rvt");
         DeleteAllWalls(data);
      }
```

Additionally, we must provide the .addin file to Revit. We added it to `C:\ProgramData\Autodesk\Revit\Addins\2018\` (or `2019` if the target is Revit 2019) and changed `<Assembly>` to point to our DLL:

```
<Assembly>C:\test\DeleteWalls\DeleteWallsTest\bin\Debug\DeleteWalls.dll</Assembly>
```

This way, we can run locally without any UI intervention on Revit startup.

Note: Your application cannot use the network or write to any files outside of the current working directory.

2. Create a Forge App and get authenticated

Design Automation runs on the Autodesk Forge platform, so you will need a Forge App to use Design Automation.

Creating a Forge App

The first step to using Design Automation for Revit is to create a Forge app. Please create a Forge app using instructions in the following link: https://developer.autodesk.com/en/docs/oauth/v2/tutorials/create-app/

Authentication

Please refer to the following links for more details:

- https://developer.autodesk.com/en/docs/oauth/v2/reference/http/authenticate-POST/
- https://developer.autodesk.com/en/docs/oauth/v2/overview/field-guide/

You use the `Client ID` and `Client Secret` obtained above to authenticate your application and obtain a two-legged access token. The details of the HTTP response is given in the above link. Please learn about the access tokens and their validity. Please use `scope=code:all` instead of `scope=data:read` to obtain a token. All Design Automation for Revit APIs require this scope.

```
curl -v 'https://developer.api.autodesk.com/authentication/v1/authenticate'
  -X 'POST'
  -H 'Content-Type: application/x-www-form-urlencoded'
  -d 'client_id=YourForgeAppClientID'
  -d 'client_secret=YourForgeAppClientSecret'
  -d 'grant_type=client_credentials'
  -d 'scope=code:all'
```

The response body to this request contains the token that you shall use for all our APIs. The response also contains the expiration time of the token.

Forge API Errors

These are the most common errors which our private beta customers have encountered:

Expired token

```
Status: 401 Unauthorized
Body: The token has expired or is invalid
```

This error can happen when the token attained to authenticate the Forge Application has expired. Please obtain a fresh token and perform the original request once again.

Invalid scope

```
Status: 403 Forbidden
Body: Token does not have the privilege for this request.
```

This error can happen when wrong scope is provided while authenticating the Forge Application. Please use `scope=code:all` to obtain a fresh token and perform the original request once again.

Too Many Requests

```
Status: 429 Too Many Requests
Body: You reached Quota limit. Your total free Quota is 20 requests per minute.
Please try again soon.
```

This error can happen when more than allowed WorkItems are posted within a given minute. The current quota limit is 20 WorkItems per minute.

3. Create a nickname for your Forge App

In Design automation, your Forge App account will be the `owner` of your app and activity. A `nickname` is a way to map a Forge App ClientId to a customized string. A nickname lets you create a friendly, easy-to-read string that can be used in place of the long Forge App ClientId.

For example, your Forge App ClientId may be something hard to read such as `"YnhayiOjhgnsd&afh890ryehQW"`. If you create a new app "`DeleteWallsApp`" with an alias "`test`", you can reference this app

by `YnhayiOjhgnsd&afh890ryehQW.DeleteWallsApp+test`. This is easier to read but still not ideal.

However, by mapping this Forge App ClientId `"YnhayiOjhgnsd&afh890ryehQW"` to a nickname of `YourNickname`, you can reference the app by using `YourNickname.DeleteWallsApp+test`, more easily and nicely.

Creating a nickname

```
curl -X PATCH \
  https://developer.api.autodesk.com/da/us-east/v3/forgeapps/me \
  -H 'Authorization: Bearer LongStringAccessTokenObtainedDuringAuthentication' \
  -H 'Content-Type: application/json' \
  -d ' {"nickname": "YourNickName"}'
```

Note:

- `LongStringAccessTokenObtainedDuringAuthentication` is the access token returned by an authentication request with the Forge App, you want to map to a nickname.
- If your Forge App doesn't have any data, you can map this Forge App to another nickname, and the new nickname will overwrite the old one. Once your Forge app has data, you cannot `PATCH` a nickname with 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 for an app with data is by first calling `[DELETE] /forgeapps/me`. This will delete any Design Automation app data associated with that app, including the nickname.
- If the nickname is already used by another user, the PATCH request will return `409 Conflict`.

4. Publish your Design Automation appbundle

An `appbundle` is the package of binaries and supporting files which make your Revit Addin application.

Appbundle Structure

Design Automation API for Revit expects your `appbundle` to be a zip file with certain contents. Here is the zip file for a sample appbundle called DeleteWallsApp.zip.

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

The top-level folder needs be named `*.bundle`. In `*.bundle` put a `PackageContents.xml` file that contains the description of the appbundle and the relative path to its .addin file.

```
<?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>
```

Note: `SeriesMin` and `SeriesMax` both refer to Revit 2018 as `R2018`. As of October 2018, Design Automation for Revit supports appbundles which run on Revit `R2018` and `R2019`.

In the `*.bundle\Contents` folder put the `addin` file and the application `DLL` and its dependencies.

```
<?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>
```

Note: `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`.

Examples of the format for the `*.bundle` folder and `PackageContent.xml` file can been found in the presentation on Autodesk Exchange Revit Apps here.

While PackageContents.xml from existing Autodesk Exchange Revit apps can be used as-is, Design Automation for Revit only reads the RuntimeRequirements and ComponentEntry blocks.

Publish an Appbundle

To publish your appbundle to Design Automation, you need to POST your appbundle's identity and upload its package.

This example creates a new appbundle `DeleteWallsApp` by posting its identity. The target engine of Revit running in Design Automation for this example appbundle is Revit 2018.

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

JSON	Description
id	The name given to the new appbundle.
engine	The engine running in Design Automation used by the appbundle.

Response

```
{
    "uploadParameters": {
        "endpointURL": "https://[myURL].com",
        "formData": {
            "key": "apps/Revit/DeleteWallsApp/1",
            "content-type": "application/octet-stream",
            "policy": "eyJleHBpcmF0aW9uIjoiMjAxOC... (truncated)",
            "success_action_status": "200",
            "success_action_redirect": null,
            "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": "YourNickname.DeleteWallsApp"
```

```
}
```

JSON	Description
`endpointURL`	This is the URL to which you must upload your appbundle's ZIP file.
`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 after `3600`seconds.

Upload appbundle zip file

Now you can upload your appbundle's ZIP to the signed URL returned by `endpointURL`:

```
curl -X POST \
  https://[myURL].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'
```

This is a `curl` example. You can use other tools, e.g. Postman, to do the uploading. Just remember to include all the form-data from the create appbundle response in your request.

Create an Alias for the Appbundle

The new version of your appbundle will be referenced via an alias.

This example creates an alias with id `test`. This alias labels version `1` of appbundle `DeleteWallsApp`.

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

Notes:

- `https://developer.api.autodesk.com/da/us-east/v3/appbundles/{appId}/aliases` - The {appId}(`DeleteWallsApp`) can be changed to use this example with other appbundles.

Update an Existing Appbundle

Create a New Version Number

To update an existing appbundle, you need to create a new version for the appbundle and then upload the updated zip package.

If you still do the POST request for creating a new appbundle above, you will get a `409 Conflict` error.

This POST creates a new version for the appbundle `DeleteWallsApp`.

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

Notes:

- You can **omit** `id` in the request body. If you have `id` in the request body, you **must** assign `null` for "id", otherwise you will get errors.
- `https://developer.api.autodesk.com/da/us-east/v3/appbundles/{appId}/versions` - The {appId}(`DeleteWallsApp`) can be changed to use this example with other appbundles.

Response

```
{
```

```
    "package": "https://[myURL].com/appbundles/xxxxxxxx",
    "engine": "Autodesk.Revit+2018",
    "description": "Delete Walls appbundle based on Revit 2018",
    "version": 2,
    "id": "YourNickname.DeleteWallsApp"
}
```

The response to the appbundle version post includes:

JSON	Description
`package`	This is the signed URL to which you must upload your updated appbundle package ZIP file.
`version`	The new version number for the appbundle created by the above POST request.

Now you can upload the updated appbundle's zip file to the new signed URL returned by `package` same as above.

Assign an existing alias to another version of an appbundle

You can update an existing alias to point to another version of an appbundle.

For example, after you post a new version of an appbundle, you may wish to assign an existing alias to point to that new appbundle's version.

Here is an example where alias `test` labels version `1` of an appbundle `DeleteWallsApp`. A new version `2` has been posted for this appbundle, but no alias labels version `2`:

id	alias	version
DeleteWallsApp	test	1
DeleteWallsApp		2

You can reassign alias `test` to label appbundle version `2`:

id	alias	version
DeleteWallsApp		1
DeleteWallsApp	test	2

To update the alias, you can either

- Delete the existing alias and recreate it with the version which you want to label.

- Do a PATCH 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 LongStringAccessTokenObtainedDuringAuthentication' \
-d '{
    "version": 2
      }'
```

Notes:

  - `https://developer.api.autodesk.com/da/us-east/v3/appbundles/{appId}/aliases/{aliasId}` - The {appId}(`DeleteWallsApp`) and {aliasId}(`test`) can be changed to use this example with other appbundle ids and alias ids.
  - `version` - The version of the appbundle the alias will label.

Engine Version Aliases

Each appbundle POST request specifies the `engine` on which the application will run. Different Design Automation engine version aliases correspond to different releases of Revit. The specified `engine` needs to be compatible with your appbundle's PackageContent.xml `SeriesMin` and `SeriesMax`.
The active engine version aliases are:

Engine	Description	JSON in appbundle post
`Autodesk.Revit+2018`	Revit 2018.3	"engine": "Autodesk.Revit+2018"
`Autodesk.Revit+2019`	Revit 2019.1	"engine": "Autodesk.Revit+2019"

5. Publish your Design Automation activity

An `activity` is an action which can be executed in Design Automation. You create and post your own activities to run against target appbundles.

Create a New Activity

To create a new `activity` with the id `DeleteWallsActivity`, post this request:

```
curl -X POST \
  https://developer.api.autodesk.com/da/us-east/v3/activities \
```

```
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer LongStringAccessTokenObtainedDuringAuthentication' \
  -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": [ "YourNickname.DeleteWallsApp+test" ],
            "description": "Delete walls from Revit file."
    }'
```

JSON	Description
`id`	The name given to your new activity.
`commandLine`	The command run by this activity. • `$(engine.path)\\\\revitcoreconsole.exe` - The full path to the folder from which the engine for Revit executes. The `engine` is defined in the request body as `"engine": "Autodesk.Revit+2018"`. More information about `engines` is here. **Do not edit or alter this "commandLine" in the request body of activity posts.** • `$(args[rvtFile].path)` - The full path to the folder which contains the input Revit model. `rvtFile`is the parameter name that the activity (`DeleteWallsActivity`) defines for the input Revit model. • `$(appbundles[DeleteWallsApp].path)` - The full path to the folder from which the `appbundle`executes. `DeleteWallsApp` refers to the

JSON	Description
	appbundle's id in "appbundles": ["YourNickname.DeleteWallsApp+test"]. • `YourNickname` - The owner of the appbundle `DeleteWallsApp`. More information about `nicknames` can be found here.
`engine`	The engine on which your activity runs. The available `engine` versions are described here.

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": [
        "YourNickname.DeleteWallsApp+test"
    ],
    "description": "Delete walls from Revit file.",
    "version": 1,
    "id": "YourNickname.DeleteWallsActivity"
}
```

The response to the new `activity` post includes:

JSON	Description
`version`	The version number for the activity created by the POST request. For the Post request creating a new activity, version always returns `1`.

Create an Alias to the New Activity Version

The activity will be referenced using an `alias`. You cannot reference an activity by its `id`. An `alias` targets a specific version of an activity.

Create an alias with the name `test` that refers to version `1` of the `DeleteWallsActivity`.

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

Note:

- `https://developer.api.autodesk.com/da/us-east/v3/activites/{activity_id}/aliases` - The {activity_id}(`DeleteWallsActivity`) in this endpoint URL can be changed to use this example with other activity's id.

Response

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

Update an Existing Activity

It is possible to update the definitions of existing activities.

Create a New Version Number

To update the definition of an existing `activity`, you will have to creates a new version of the activity.

If you still do the Post request for creating a new activity above, you will get a `409 Conflict` error.

This POST command creates a new version for the activity `DeleteWallsActivity`.

```
 curl -X POST \
  https://developer.api.autodesk.com/da/us-
east/v3/activities/DeleteWallsActivity/versions \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer LongStringAccessTokenObtainedDuringAuthentication' \
  -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": [ "YourNickname.DeleteWallsApp+test" ],
        "description": "Delete walls from Revit file Updated."
 }'
```

Note:

- You can **omit** `id` in the request body. If you have `id` in the request body, you **must** assign `null` for "id", otherwise you will get errors!
- `https://developer.api.autodesk.com/da/us-east/v3/activities/{activity_id}/versions` - The activity {activity_id}(`DeleteWallsActivity`) in the endpoint URL can be changed to use this example with other activity's id.

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": [
        "YourNickname.DeleteWallsApp+test"
    ],
    "description": "Delete walls from Revit file Updated.",
    "version": 2,
    "id": "YourNickname.DeleteWallsActivity"
}
```

Assign an existing alias to another version of an activity

You can update an existing alias to point to another version of an activity.

This is very similar to updating aliases for appbundles, although you'll use the activity endpoint instead of the appbundle one.

Using a different language

You can request Revit to run in a different language using the `/l` specifier in the command line, for example, when creating an activity:

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

- This will tell Revit to launch in German, and all Revit output will be in German. This would be useful if you want in-built elements and types to have German names.

See this article for the full list of language codes.

Note that Design Automation for Revit output is only in English; however Revit API output is localized and can be customized to use different languages.

6. Post your Design Automation workitem

When you post a workitem to Design Automation, you are requesting a job to be run on Design Automation.

A workitem is used 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.

POST a Workitem

Here is an example of a workitem that executes the `DeleteWallsActivity`.

```
curl -X POST \
  https://developer.api.autodesk.com/da/us-east/v3/workitems \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer LongStringAccessTokenObtainedDuringAuthentication' \
  -d '{
        "activityId": "YourNickname.DeleteWallsActivity+test",
        "arguments": {
          "rvtFile": {
            "url": "https://[myURL]/DeleteWalls.rvt"
          },
          "result": {
            "verb": "put",
            "url": "https://myWebsite/signed/url/to/result"
          }
        }
      }'
```

`LongStringAccessTokenObtainedDuringAuthentication` needs to be replaced with your authentication token string.

JSON	Description
`activityId`	The target activity defined by "owner.activity+alias"(`YourNickname.DeleteWallsActivity+test`) this workitem will execute.
`arguments`	The argument list that is required by the activity (`DeleteWallsActivity`): • `rvtFile` - It is the URL to get the input file that will be processed by the workitem.

JSON	Description
	• `result` - It is the URL to which the output will be "put" (uploaded). **You must provide this URL; there are no default Design Automation URLs for output data.**

Reponse

The HTTP response will contain the `id` of the posted workitem.

```
{
    "status": "pending",
    "stats": {
        "timeQueued": "2018-04-16T21:45:08.1357163Z"
    },
    "id": "e8a3ee53770a4eaeb86f267aab76af47"
}
```

Workitem Status

Design Automation workitems are queued before they are processed. Processed workitems may have run successfully or may have failed execution.

You can check the status of a workitem by calling `[GET] /workitems/{id}`:

```
curl -X GET \
  https://developer.api.autodesk.com/da/us-
east/v3/workitems/e8a3ee53770a4eaeb86f267aab76af47 \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer LongStringAccessTokenObtainedDuringAuthentication'
```

`LongStringAccessTokenObtainedDuringAuthentication` needs to be replaced with your authentication token string.

Notes: `https://developer.api.autodesk.com/da/us-east/v3/workitems/{workitemId}` - The {workitemId}(`e8a3ee53770a4eaeb86f267aab76af47`) must be changed to use your workitem id.

Response

The Response is like below:

```
{
    "status": "success",
    "reportUrl":
"https://[myURL].com/workItem/Revit/e8a3ee53770a4eaeb86f267aab76af47/report.txt?XXXXX
XXXX",
    "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": "e8a3ee53770a4eaeb86f267aab76af47"
}
```

JSON	Description
`status`	Indicates if execution is pending, successful, failed or cancelled.
`reportUrl`	The URL to get the report log for this workitem's execution.
`progress`	*A place holder for future use. You can ignore this now.*

Arguments: More Support

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`, which indicates which Revit elements to take a count of. 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 below.
By prefixing those values with `data:application/json`,, it instructs the Design Automation framework to treat them as JSON stream and save them as a JSON file.

```
curl -X POST \
  https://developer.api.autodesk.com/da/us-east/v3/workitems \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer LongStringAccessTokenObtainedDuringAuthentication' \
  -d '{
          "activityId": "YourNickname.CountItActivity+test",
          "arguments": {
            "rvtFile": {
              "url": "https://[myURL].com/CountIt.rvt"
            },
            "countItParams": {
              "url": "data:application/json,{'walls': false, 'floors': true, 'doors':
true, 'windows': true}"
            },
            "result": {
              "verb": "put",
              "url": "https://myWebsite/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.

```
curl POST \
  https://developer.api.autodesk.com/da/us-east/v3/workitems \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer LongStringAccessTokenObtainedDuringAuthentication' \
  -d '{
        "activityId" : "YourNickname.CountItActivity+test",
        "arguments": {
          "rvtFile": {
            "url": "https://[myURL].com/TopHost.zip"
          },
          "countItParams": {
            "url": "data:application/json,{'walls': true, 'floors': true, 'doors':
true, 'windows': true}"
          },
          "result": {
            "verb": "put",
            "url": "https://myWebsite/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

```
curl POST \
  https://developer.api.autodesk.com/da/us-east/v3/workitems \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer LongStringAccessTokenObtainedDuringAuthentication' \
  -d '{
        "activityId": "YourNickname.CountItActivity+test",
        "arguments": {
          "rvtFile": {
            "url": "https://[myURL].com/TopHost.rvt",
            "references": [
              {
                "url": "https://[myURL].com/LinkA.rvt",
                "references": [
                  {
                    "url": "https://[myURL].com/LinkA1.rvt"
                  },
                  {
                    "url": "https://[myURL].com/LinkA2.rvt"
```

```
                }
              ]
            },
            {
              "url": "https://[myURL].com/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 of these files are 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://[myURL].com/TestForSubFolders/Host.rvt",
  "references": [
    {
      "url": "https://[myURL].com/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 like 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
```

Output Arguments: onComplete callback

Each `workitem` is furnished with a special output argument named `onComplete`. When provided, the callback URL will be called on completion of the workitem.

Here is an example of how to call `[POST] /workitems`, which adds `onComplete` argument to the earlier example.

```
curl -X POST \
  https://developer.api.autodesk.com/da/us-east/v3/workitems \
  -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer LongStringAccessTokenObtainedDuringAuthentication' \
  -d '{
        "activityId": "YourNickname.DeleteWallsActivity+test",
        "arguments": {
          "rvtFile": {
            "url": "https://[myURL].com/DeleteWalls.rvt"
          },
          "result": {
            "verb": "put",
            "url": "https://myWebsite/signed/url/to/result"
          },
          "onComplete": {
            "verb": "post",
            "url": "https://myWebsite/callback"
          }
        }
      }'
```

This argument is not `required` to be provided on every `[POST] /workitems` call.

On completion of the workitem, the sepecified url is called with a payload identical to the response received on `[GET] /workitems/{id}` call.

Since the nature of the implementation of the callback url is similar to how one may implement a callback url for Webhooks API, you may refer the Webhooks API documentation. You may also find their documentation for configuring local server helpful.

7. Appendix A: Design Automation Quotas and Restrictions

There are quotas and restrictions on `appbundles`, `activities` and `workItems`.

Naming restrictions

There are some restrictions on `nickname`, `alias` and `id` of `appbundle/activity`: you can only use alphanumeric characters [a-zA-Z_0-9].

Appbundle Limits

Limit	Value	Description
Appbundle upload size	100 MB	Max permitted size of an appbundle upload in megabytes.
Payload size	8 KB	Max permitted size for appbundle/activity json payload in kilobytes.
Versions	100	Max permitted number of appbundle/activity versions a client can have at any one time.

Additional Appbundle Restrictions

There are additional restrictions placed on appbundles that run on Design Automation API for Revit. These include:

- No access to Revit's UI interfaces. Your application must be a RevitDB application only.

- Applications must be able to process jobs to completion without waiting for user input or interaction.
- No network access is allowed.
- Writing to the disk is restricted to Revit's current working directory.
- Your application is run with low privileges, and will not be able to freely interact with Windows OS.

Alias Limits

Limit	Value	Description
Aliases	100	Max permitted number of aliases that a client can have at any one time.

Activity Limits

Limit	Value	Description
Payload size	8 KB	Max permitted size for appbundle/activity json payload in kilobytes.
Total uncompressed appbundles size	2,000 MB	Max permitted size of all appbundles referenced by an activity. It is enforced when you post a workitem.
Versions	100	Max permitted number of appbundle/activity versions a client can have at any one time.

Workitem Limits

Limit	Value	Description
Downloads	200	Max number of downloads per workitem.
Download size	2,000 MB	Max total size of all downloads in MB per workitem.
Processing time	3,600 seconds (1 hour)	Max duration of processing in seconds per workitem (includes download and upload time).
Uploads	200	Max number of uploads per workitem.
Upload size	2,000 MB	Max total size of all uploads in MB per workitem.
Workitems per minute	20 per/min	Max number of workitems a client can submit in one minute.

8. Appendix B: Handling Failures in Revit Appbundles

Normally posted failures are processed by Revit's standard failure resolution UI at the end of a transaction when Transaction.Commit() or Transaction.Rollback() are invoked. The user is presented information and options to deal with the failures.

If an operation (or set of operations) on the document requires some special treatment from a Revit addin for certain errors, failure handling can be customized to carry out this resolution. Custom failure handling can be supplied:

- For a given transaction using the interface IFailuresPreprocessor.
- For all possible errors using the FailuresProcessing event.

Finally, the API offers the ability to completely replace the standard failure processing user interface using the interface IFailuresProcessor. Although the first two methods for handling failures should be sufficient in most cases, this last option can be used in special cases, such as to provide a better failure processing UI (UI is not available in Design Automation for Revit) or when an application is used as a front-end on top of Revit.

Overview of Failure Processing

It is important to remember there are many things happening between the call to Transaction.Commit() and the actual processing of failures. Auto-join, overlap checks, group checks and workset editability checks are just to name a few. These checks and changes may make some failures disappear or, more likely, can post new failures. Therefore, conclusions cannot be drawn about the state of failures to be processed when Transaction.Commit() is called. To process failure correctly, it is necessary to hook up the actual failure processing mechanism.

When failures processing begins, all changes to a document that are supposed to be made in the transaction are made, and all failures are posted. Therefore, no uncontrolled changes to a document are allowed during failures processing. There is a limited ability to resolve failures via the restricted interface provided by FailureAccessor. If this has happened all end of transaction checks and failure processing must be repeated. So, there may be a few failure resolution cycles at the end of one transaction.

Each cycle of failures processing includes 3 steps:

1. Preprocessing of failures (FailuresPreprocessor)
2. Broadcasting of failures processing events (FailuresProcessing event)
3. Final processing (FailuresProcessor)

Each of these 3 steps can control what happens next by returning different FailureProcessingResults. The options are:

- **Continue** - has no impact on execution flow. If FailuresProcessor returns "Continue" with unresolved failures, Revit will instead act as if "ProceedWithRollBack" was returned.
- **ProceedWithCommit** - interrupts failures processing and immediately triggers another loop of end-of-transaction checks followed by another failures processing. Should be returned after an attempt to resolve failures. Can easily lead to an infinite loop if returned without any successful failure resolution. Cannot be returned if transaction is already being rolled back and will be treated as "ProceedWithRollBack" in this case.
- **ProceedWithRollback** - continues execution of failure processing, but forces transaction to be rolled back, even if it was originally requested to commit. If before ProceedWithRollBack is returned FailureHandlingOptions are set to clear errors after rollback, no further error processing will take place, all failures will be deleted and the transaction is rolled back silently. Otherwise default failure processing will continue but the transaction is guaranteed to be rolled back.

Depending on the severity of failures posted in the transaction and whether the transaction is being committed or rolled back, each of these 3 steps may have certain options to resolve errors. All information about failures posted in a document, information about ability to perform certain operations to resolve failures and API to perform such operations are provided via the FailuresAccesor class. The Document can be used to obtain additional information, but it cannot be changed other than via FailuresAccessor.

Failure Handling in Design Automation for Revit

Default Failure Handling

The DesignAutomationBridge comes with a default failure handler. This failure handler will suppress all the warnings it encounters. In case of errors it will try to resolve them. If resolved successfully, the transaction will proceed with commit. If the default resolution is to delete elements the failure handler will not perform the delete action and will proceed with roll back.

Custom Failure Handling

You can implement your own custom failure handler to override the default failure handler. The failure posting API is easy to use. A custom failure definition can be registered in the HandleDesignAutomationReadyEvent() method of an external application, and then the failure severity and resolution type can be set. To register a failures processor, implement the interface IFailuresProcessor and register it using the Application.RegisterFailuresProcessor() method. Here is a code fragment to register a custom failure handler.

```
public void HandleDesignAutomationReadyEvent(object sender,
DesignAutomationReadyEventArgs e)
{
        // Hook up the CustomFailureHandling failure processor.
        Application.RegisterFailuresProcessor(new CustomFailureHandlingProcessor());

        // Run the application logic.
        SketchItFunc(e.DesignAutomationData);

        e.Succeeded = true;
}
```

FailuresProcessor

The IFailuresProcessor interface gets control last, after the FailuresProcessing event is processed. There is only one active IFailuresProcessor in a Revit session. If there is a previously registered failures processor, it is discarded. If the addin does not provide a failure processor of its own, then the default failure processor from Design Automation for Revit will be active. You can deactivate the default failure processor from Design Automation for Revit by passing null to RegisterFailuresProcessor(). In the absence of all failure processors, Revit resolves failures via the default failure resolution type.

Warning: The IFailuresProcessor.ProcessFailures() method is allowed to return WaitForUserInput, which leaves the transaction pending so that the FailuresProcessor can display UI and get user input before resolving the failure. Don't use WaitForUserInput in Design Automation for Revit; there is no UI or user interaction in Design Automation for Revit.

The default failure handler from Design Automation for Revit implements IFailuresProcessor and suppresses any warnings. If there are errors, the code rolls back the transaction. If you need a different behavior, you can modify the code for the default FailureProcessor to fit your specifications.

9. Appendix C: Sample Appbundle

We have code samples showing simple examples of the creation, modification, and extraction workflows for Design Automation.

Creation Workflow - SketchIt

SketchIt is an application that creates walls and floors in a rvt file. It takes a JSON file that specifies the walls and floors to be created, and outputs a new rvt file.

Note: You will not be able to run this sample today. The DesignAutomationBridge dependency is not available, and Design Automation is not accepting Revit queries yet. However, we wanted to provide some sample code to show you what an appbundle will look like. As before, the interfaces are subject to change before release, but this is accurate as of October 2018.

Dependencies

This project was built in Visual Studio 2017.

This sample references Revit 2018's `RevitAPI.dll`, DesignAutomationBridge.dll for Revit 2018 and Newtonsoft JSON framework.
In order to POST appbundles, activities, and workitems you must have credentials for Forge.

Creating and Publishing the Appbundle

Create an `appbundle` zip package from the build outputs and publish the `appbundle` to Design Automation.
The `JSON` in your appbundle POST should look like this:

```
{
  "id": "SketchItApp",
  "engine": "Autodesk.Revit+2018",
  "description": "SketchIt appbundle based on Revit 2018"
}
```

Notes:

- `engine = Autodesk.Revit+2018`

After you upload the `appbundle` zip package, you should create an alias for this appbundle. The `JSON` in the POST should look like this:

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

Creating the Activity

Define an `activity` to run against the `appbundle`.

The `JSON` that accompanies the `activity` POST will look like this:

```
{
   "id": "SketchItActivity",
   "commandLine": [ "$(engine.path)\\\\revitcoreconsole.exe /al
$(appbundles[SketchItApp].path)" ],
   "parameters": {
      "sketchItInput": {
         "zip": false,
         "ondemand": false,
         "verb": "get",
         "description": "SketchIt input parameters",
         "required": true,
         "localName": "SketchItInput.json"
      },
      "result": {
         "zip": false,
         "ondemand": false,
         "verb": "put",
         "description": "Results",
         "required": true,
         "localName": "sketchIt.rvt"
      }
   },
   "engine": "Autodesk.Revit+2018",
   "appbundles": [ "YourNickname.SketchItApp+test" ],
   "description": "Creates walls and floors from an input JSON file."
}
```

Notes:

- `engine` = `Autodesk.Revit+2018`
- `YourNickname` - The owner of appbundle `SketchItApp`.

Then you should create an alias for this activity. The `JSON` in the POST should look like this:

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

POST a WorkItem

Now POST a `workitem` against the `activity` to run a job on your `appbundle`.

The `JSON` that accompanies the `workitem` POST will look like this:

```
{
  "activityId": "YourNickname.SketchItActivity+test",
  "arguments": {
    "sketchItInput": {
```

```
      "url": "data:application/json,{ 'walls': [ {'start': { 'x': -100, 'y': 100,
'z': 0.0}, 'end': { 'x': 100, 'y': 100, 'z': 0.0}}, {'start': { 'x': -100, 'y': 100,
'z': 0.0}, 'end': { 'x': 100, 'y': 100, 'z': 0.0}}, {'start': { 'x': 100, 'y': 100,
'z': 0.0}, 'end': { 'x': 100, 'y': -100, 'z': 0.0}}, {'start': { 'x': 100, 'y': -100,
'z': 0.0}, 'end': { 'x': -100, 'y': -100, 'z': 0.0}}, {'start': { 'x': -100, 'y': -
100, 'z': 0.0}, 'end': { 'x': -100, 'y': 100, 'z': 0.0}}, {'start': { 'x': -500, 'y':
-300, 'z': 0.0}, 'end': { 'x': -300, 'y': -300, 'z': 0.0}}, {'start': { 'x': -300,
'y': -300, 'z': 0.0}, 'end': { 'x': -300, 'y': -500, 'z': 0.0}}, {'start': { 'x': -
300, 'y': -500, 'z': 0.0}, 'end': { 'x': -500, 'y': -500, 'z': 0.0}}, {'start': {
'x': -500, 'y': -500, 'z': 0.0}, 'end': { 'x': -500, 'y': -300, 'z': 0.0}}],'floors'
: [ [{'x': -100, 'y': 100, 'z':0.0}, {'x': 100, 'y': 100, 'z': 0.0}, {'x': 100, 'y':
-100, 'z': 0.0}, {'x': -100, 'y': -100, 'z': 0.0}], [{'x': -500, 'y': -300, 'z':0.0},
{'x': -300, 'y': -300, 'z': 0.0}, {'x': -300, 'y': -500, 'z': 0.0}, {'x': -500, 'y':
-500, 'z': 0.0}] ]}"
    },
    "result": {
      "verb": "put",
      "url": "https://myWebsite/signed/url/to/sketchIt.rvt"
    }
  }
}
```

Notes:

- `YourNickname` - The owner of activity `SketchItActivity`.

`SketchItActivity` expects an input file `SketchItInput.json`. The contents of the embedded JSON are stored in a file named `SketchItInput.json`, as specified by the `parameters` of `sketchItInput` in the activity `SketchItActivity`. The SketchIt application reads this file from current working folder, parses the JSON and creates walls and floors from the extracted specifications in a new created Revit file `sketchIt.rvt`, which will be uploaded to `url` you provide in the workitem.
The function `SketchItFunc` in SketchIt.cs performs these operations.

Source code

SketchIt.cs

```
using System;
using System.Collections.Generic;
using System.IO;

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

namespace SketchIt
{
```

```
[Autodesk.Revit.Attributes.Regeneration(Autodesk.Revit.Attributes.RegenerationOption.Manu
al)]

[Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)]
   class SketchItApp : IExternalDBApplication
   {
      public ExternalDBApplicationResult
OnStartup(Autodesk.Revit.ApplicationServices.ControlledApplication app)
      {
         DesignAutomationBridge.DesignAutomationReadyEvent +=
HandleDesignAutomationReadyEvent;
         return ExternalDBApplicationResult.Succeeded;
      }

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

      public void HandleDesignAutomationReadyEvent(object sender,
DesignAutomationReadyEventArgs e)
      {
         // Run the application logic.
         SketchItFunc(e.DesignAutomationData);

         e.Succeeded = true;
      }

      private static void SketchItFunc(DesignAutomationData data)
      {
         if (data == null)
            throw new InvalidDataException(nameof(data));

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

         Document newDoc = rvtApp.NewProjectDocument(UnitSystem.Imperial);
         if (newDoc == null)
            throw new InvalidOperationException("Could not create new document.");
         string filePath = "sketchIt.rvt";

         string filepathJson = "SketchItInput.json";
         SketchItParams jsonDeserialized = SketchItParams.Parse(filepathJson);

         CreateWalls(jsonDeserialized, newDoc);

         CreateFloors(jsonDeserialized, newDoc);

         newDoc.SaveAs(filePath);
      }
```

```
      private static void CreateWalls(SketchItParams jsonDeserialized, Document newDoc)
      {
         FilteredElementCollector levelCollector = new FilteredElementCollector(newDoc);
         levelCollector.OfClass(typeof(Level));
         ElementId someLevelId = levelCollector.FirstElementId();
         if (someLevelId == null || someLevelId.IntegerValue < 0) throw new
InvalidDataException("ElementID is invalid.");

         List<Curve> curves = new List<Curve>();
         foreach (WallLine lines in jsonDeserialized.Walls)
         {
            XYZ start = new XYZ(lines.Start.X, lines.Start.Y, lines.Start.Z);
            XYZ end = new XYZ(lines.End.X, lines.End.Y, lines.End.Z);
            curves.Add(Line.CreateBound(start, end));
         }

         using (Transaction wallTrans = new Transaction(newDoc, "Create some walls"))
         {
            wallTrans.Start();

            foreach (Curve oneCurve in curves)
            {
               Wall.Create(newDoc, oneCurve, someLevelId, false);
            }

            wallTrans.Commit();
         }
      }

      private static void CreateFloors(SketchItParams jsonDeserialized, Document newDoc)
      {
         foreach (List<Point> floorPoints in jsonDeserialized.Floors)
         {
            CurveArray floor = new CurveArray();
            int lastPointOnFloor = floorPoints.Count - 1;

            for (int pointNum = 0; pointNum <= lastPointOnFloor; pointNum++)
            {
               XYZ startPoint = new XYZ(floorPoints[pointNum].X, floorPoints[pointNum].Y,
floorPoints[pointNum].Z);
               XYZ endPoint;

               if (pointNum == lastPointOnFloor)
               {
                  endPoint = new XYZ(floorPoints[0].X, floorPoints[0].Y,
floorPoints[0].Z);
               }
               else
               {
                  endPoint = new XYZ(floorPoints[pointNum + 1].X, floorPoints[pointNum +
1].Y, floorPoints[pointNum + 1].Z);
               }

               Curve partOfFloor = Line.CreateBound(startPoint, endPoint);
```

```
                floor.Append(partOfFloor);
            }

            using (Transaction floorTrans = new Transaction(newDoc, "Create a floor"))
            {
                floorTrans.Start();
                newDoc.Create.NewFloor(floor, false);
                floorTrans.Commit();
            }
        }
    }
}
```

SketchItParams.cs

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SketchIt
{

    internal class Point
    {
        [JsonProperty(PropertyName = "x")]
        public double X { get; set; } = 0.0;
        [JsonProperty(PropertyName = "y")]
        public double Y { get; set; } = 0.0;
        [JsonProperty(PropertyName = "z")]
        public double Z { get; set; } = 0.0;
    }
    internal class WallLine
    {
        [JsonProperty(PropertyName = "start")]
        public Point Start { get; set; }
        [JsonProperty(PropertyName = "end")]
        public Point End { get; set; }
    }



    internal class SketchItParams
    {
        [JsonProperty(PropertyName = "walls")]
        public IList<WallLine> Walls { get; set; }
        [JsonProperty(PropertyName = "floors")]
        public IList<IList<Point>> Floors { get; set; }
```

```
        static public SketchItParams Parse(string jsonPath)
        {
            try
            {
                if (!File.Exists(jsonPath))
                    return new SketchItParams();

                string jsonContents = File.ReadAllText(jsonPath);
                return JsonConvert.DeserializeObject<SketchItParams>(jsonContents);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Exception happens when parsing the json file: " + ex);
                return null;
            }
        }
    }
}
```

Modification Workflow – DeleteWalls

DeleteWalls is an application that takes in a rvt file and outputs another rvt file with all of the walls removed.

Notes on POSTing the DeleteWalls WorkItem

The `JSON` that accompanies the `WorkItem` POST will look like this:

```
{
  "activityId": "YourNickname.DeleteWallsActivity+test",
  "arguments": {
    "rvtFile": {
      "url": "https://myWebsite/DeleteWalls.rvt"
    },
    "result": {
      "verb": "put",
      "url": "https://myWebsite/signed/url/to/result.rvt"
    }
  }
}
```

The value of the `rvtFile` parameter is the URL to the input file `DeleteWalls.rvt`. DeleteWalls application opens `DeleteWalls.rvt`, deletes the walls in it and saves it as `result.rvt`. The output file `result.rvt` will be uploaded to `url` you provide in the workitem.

The function `DeleteAllWalls` in DeleteWalls.cs performs these operations.

Source code

DeleteWalls.cs

```
using System;
using System.IO;

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

namespace DeleteWalls
{

[Autodesk.Revit.Attributes.Regeneration(Autodesk.Revit.Attributes.RegenerationOption.Manu
al)]

[Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)]
   public class DeleteWallsApp : IExternalDBApplication
   {
      public ExternalDBApplicationResult
OnStartup(Autodesk.Revit.ApplicationServices.ControlledApplication app)
      {
         DesignAutomationBridge.DesignAutomationReadyEvent +=
HandleDesignAutomationReadyEvent;
         return ExternalDBApplicationResult.Succeeded;
      }

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

      public void HandleDesignAutomationReadyEvent(object sender,
DesignAutomationReadyEventArgs e)
      {
         e.Succeeded = true;
         DeleteAllWalls(e.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());
     }
  }
}
```

Extraction Workflow – CountIt

CountIt is an application that counts walls, floors, doors and windows in a rvt file and its rvt links. It takes a JSON file that specifies which categories of elements will be counted. The output of this application is a text file which contains the element counts.

Notes on creating the Activity

Define an `activity` to run against the `appbundle`.

The `JSON` that accompanies the `activity` POST will look like this:

```
{
   "id": "CountItActivity",
   "commandLine": [ "$(engine.path)\\\\revitcoreconsole.exe /i $(args[rvtFile].path)
/al $(appbundles[CountItApp].path)" ],
   "parameters": {
      "rvtFile": {
         "zip": false,
         "ondemand": false,
         "verb": "get",
         "description": "Input Revit model",
         "required": true,
         "localName": "$(rvtFile)"
      },
      "countItParams": {
         "zip": false,
         "ondemand": false,
         "verb": "get",
         "description": "CountIt parameters",
         "required": false,
         "localName": "CountItParams.json"
      },
```

```
      "result": {
         "zip": false,
         "ondemand": false,
         "verb": "put",
         "description": "Results",
         "required": true,
         "localName": "result.txt"
      }
   },
   "engine": "Autodesk.Revit+2018",
   "appbundles": [ "YourNickname.CountItApp+test" ],
   "description": "Count and output elements from Revit file."
}
```

Then you should create an alias for this activity. The `JSON` in the POST should look like this:

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

Notes on POSTing the WorkItem

Now POST a `workitem` against the `activity` to run a job on your `appbundle`.
The `JSON` that accompanies the `workitem` POST will look like this:

```
{
  "activityId": "YourNickname.CountItActivity+test",
  "arguments": {
    "rvtFile": {
      "url": "https://myWebsite/CountIt.rvt"
    },
    "countItParams": {
      "url": "data:application/json,{'walls': false, 'floors': true, 'doors': true,
'windows': true}"
    },
    "result": {
      "verb": "put",
      "url": "https://myWebsite/signed/url/to/result.txt"
    }
  }
}
```

`CountItActivity` expects an input file `CountItParams.json`. The contents of the embedded JSON will be stored in a file named `CountItParams.json`, as specified by the `parameters` of `countItParams` in the activity `CountItActivity`. The CountIt application

reads this file from current working folder and parses the JSON to determine which element categories should be counted. The counting result is saved to `result.txt` which will be uploaded to `url` you provide in the workitem.
The function `CountElementsInModel` in Main.cs performs these operations.

Source code

Main.cs

```
using System.Collections.Generic;
using System.IO;

using Autodesk.Revit.ApplicationServices;
using Autodesk.Revit.DB;
using DesignAutomationFramework;
using Newtonsoft.Json;


namespace CountIt
{

[Autodesk.Revit.Attributes.Regeneration(Autodesk.Revit.Attributes.RegenerationOption.Manu
al)]

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

         public ExternalDBApplicationResult OnShutdown(ControlledApplication app)
         {
            return ExternalDBApplicationResult.Succeeded;
         }

         public void HandleDesignAutomationReadyEvent(object sender,
DesignAutomationReadyEventArgs e)
         {
            e.Succeeded = CountElementsInModel(e.DesignAutomationData.RevitApp,
e.DesignAutomationData.FilePath, e.DesignAutomationData.RevitDoc);
      }

        internal static List<Document> GetHostAndLinkDocuments(Document revitDoc)
        {
            List<Document> docList = new List<Document>();
            docList.Add(revitDoc);
```

```
            // Find RevitLinkInstance documents
            FilteredElementCollector elemCollector = new
FilteredElementCollector(revitDoc);
            elemCollector.OfClass(typeof(RevitLinkInstance));
            foreach (Element curElem in elemCollector)
            {
                RevitLinkInstance revitLinkInstance = curElem as RevitLinkInstance;
                if (null == revitLinkInstance)
                   continue;

                Document curDoc = revitLinkInstance.GetLinkDocument();
                if (null == curDoc) // Link is unloaded.
                   continue;

                // When one linked document has more than one RevitLinkInstance in the
                // host document, then 'docList' will contain the linked document
multiple times.

                docList.Add(curDoc);
            }

            return docList;
        }

        internal static void CountElements(Document revitDoc, CountItParams
countItParams, ref CountItResults results)
        {
            if (countItParams.walls)
            {
                FilteredElementCollector elemCollector = new
FilteredElementCollector(revitDoc);
                elemCollector.OfClass(typeof(Wall));
                int count = elemCollector.ToElementIds().Count;
                results.walls += count;
                results.total += count;
            }

            if (countItParams.floors)
            {
                FilteredElementCollector elemCollector = new
FilteredElementCollector(revitDoc);
                elemCollector.OfClass(typeof(Floor));
                int count = elemCollector.ToElementIds().Count;
                results.floors += count;
                results.total += count;
            }

            if (countItParams.doors)
            {
                FilteredElementCollector collector = new
FilteredElementCollector(revitDoc);
                ICollection<Element> collection =
collector.OfClass(typeof(FamilyInstance))
                                                    .OfCategory(BuiltInCategory.OST_Doors)
```

```
                                                    .ToElements();

                int count = collection.Count;
                results.doors += count;
                results.total += count;
            }

            if (countItParams.windows)
            {
                FilteredElementCollector collector = new
FilteredElementCollector(revitDoc);
                ICollection<Element> collection =
collector.OfClass(typeof(FamilyInstance))

.OfCategory(BuiltInCategory.OST_Windows)
                                                    .ToElements();

                int count = collection.Count;
                results.windows += count;
                results.total += count;
            }
        }

        public static bool CountElementsInModel(Application rvtApp, string
inputModelPath, Document doc)
        {
            if (rvtApp == null)
               return false;

            if (!File.Exists(inputModelPath))
               return false;

            if (doc == null)
               return false;

            // For CountIt workItem: If RvtParameters is null, count all types
            CountItParams countItParams = CountItParams.Parse("CountItParams.json");
            CountItResults results = new CountItResults();

            List<Document> allDocs = GetHostAndLinkDocuments(doc);
            foreach(Document curDoc in allDocs)
            {
               CountElements(curDoc, countItParams, ref results);
            }

            using (StreamWriter sw = File.CreateText("result.txt"))
            {
               sw.WriteLine(JsonConvert.SerializeObject(results));
               sw.Close();
            }

            return true;
        }
    }
```

```
}
```

CountItParams.cs

```
using System;
using System.IO;
using Newtonsoft.Json;

namespace CountIt
{
   internal class CountItParams
   {
      public bool walls { get; set; } = false;
      public bool floors { get; set; } = false;
      public bool doors { get; set; } = false;
      public bool windows { get; set; } = false;

      static public CountItParams Parse(string jsonPath)
      {
         try
         {
            if (!File.Exists(jsonPath))
               return new CountItParams { walls = true, floors = true, doors = true,
windows = true };

            string jsonContents = File.ReadAllText(jsonPath);
            return JsonConvert.DeserializeObject<CountItParams>(jsonContents);
         }
         catch (Exception ex)
         {
            Console.WriteLine("Exception when parsing json file: " + ex);
            return null;
         }
      }
   }
}
```

CountItResults.cs

```
namespace CountIt
{
   internal class CountItResults
   {
      public int walls { get; set; } = 0;
      public int floors { get; set; } = 0;
      public int doors { get; set; } = 0;
      public int windows { get; set; } = 0;
      public int total { get; set; } = 0;
   }
}
```