SD125671

Get Your Linked Data from Anywhere with the Revit API

Rahul Bhobe
Autodesk

Diane Christoforo
Ryan Duell

Learning Objectives

- Understand the options for providing Revit, DWG, and other CAD links from locations other than "a file on my hard drive"
- Understand how to create and provide custom keynote data without having to hand write a file
- Learn how to create Revit plug-ins using the external resource service in Revit software's API
- Learn how to use recent Revit API improvements to place and control your linked data

Description

Did you know you can get your keynote data from a database or your company wiki, or that you can share links across multiple users without complicated setups with mapped network drives? This class will teach Revit API users how to use the external resource service to manage linked models and other external data. You will learn how to easily create references to Revit software, DWG™ software, or other CAD formats that are provided by a Revit plug-in. This can give you more control over how to manage and control access to links. For example, in a work-shared environment, users with different machine setups can still access the same linked data. (Or not access it!) You will also learn how to customize your keynote data and avoid using a keynote file entirely if you wish. We'll also go over some tips and tricks for using links, including some enhancements made in Revit 2018 software.

Speaker(s)

Rahul Bhobe is a Principal Software Developer for Revit at Autodesk. 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 9 years working on several features of Revit. Currently he is working on providing cloud services/solutions for Revit. In 2008, he started working on Revit Conceptual Modeling design where he implemented Adaptive Components, Point Elements and Divided Surface. He has then worked on several Revit features like Family, Groups, Links and Worksharing. Prior to Autodesk, he had 8 years of CAD/CAM development experience.

Contents

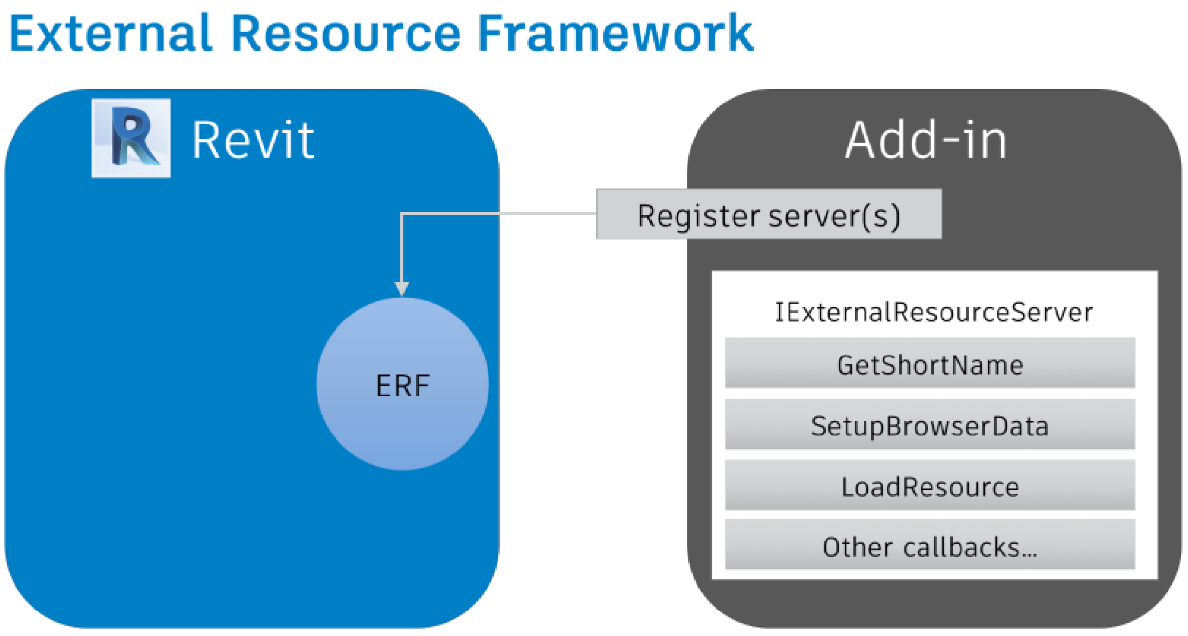
What is the External Resource framework?

The external resource framework is one of Revit's ExternalServices – the ExternalResourceService. It allows third parties to provide linked file data to Revit from places other than a file on a local hard drive. This allows you to provide data from a private company server, a website, a cloud service …

How do you use the framework?

To use the framework, you create a Revit addin with two parts – one implementing IExternalResourceServer and one implementing IExternalResourceUIServer.

1.) When the user browses for links, Revit will ask your IExternalResourceServer what resources it can provide.
2.) When the user chooses one of your resources, Revit asks your IExternalResourceServer for the data.
3.) When the load completes or fails, Revit asks your IExternalResourceUIServer if it wants to display any UI.



What resource types can use the framework?

The framework works with:

- Revit links
- DWG links
- Other CAD link types (except DWFs)
- The Keynote table
- The Assembly Code table

DWG and other CAD link types are new as of Revit 2018.

When should you use the framework?

At a high level, you should use the framework when you want to use Revit or DWG linked data from a source that isn't "a local file on my hard drive." Here are a few examples where it might be useful to you:

- When you want to get linked files from a cloud service which Revit doesn't have native support for.
- When you want to use keynotes without having to manage the keynote file.
- When you want to let different users keep their links in different places.
- When you're sick of the Unresolved References dialog and you want to display your own custom UI in response to link loading.

When should you avoid the framework?

Sometimes the External Resource Framework isn't what you want:

- If you want to stream Revit or CAD links directly to Revit, the framework cannot handle that. You will have to fully download the link before Revit can load it.
- If you want to use eTransmit on the links later - External Resource links can't be picked up by eTransmit for Revit.
- If you want to change the "permanent" data in the link on file open, you can't do that with this framework. You can only change the permanent data on a Reload From. (See the section on ExternalResourceReference for more detail on what this means.)

The code

(Quick) Summaries of the Classes Involved

Here's a quick overview of the classes involved in the framework. There will be more detailed explanations for the classes which you must implement or define in a later section.

ExternalService

The service implementation inside Revit. The External Resource Service has the id `ExternalServices.BuiltInExternalServices.ExternalResourceService`. You will register your server as part of the `that service` and it will execute your server when appropriate.

ExternalResourceType

An enum value indicating the type of an external resource. Internal types are listed at ExternalResourceTypes.BuiltInExternalResourceTypes. (For example, the keynote table is indicated with `ExternalResourceTypes.BuiltInExternalResourceTypes.KeynoteTable`.)

IExternalResourceServer

This is the primary interface you will implement, creating the actual resource provider. The main method you will implement is called LoadResource() and provides the linked file data to Revit. You will also implement methods to check whether resources are out of date, determine whether resources are in the right format for your server, and list the resources your server provides.

IExternalResourceUIServer

This is the secondary interface you will implement, for handling UI operations. The main method is HandleLoadResourceResults() which displays any UI or error messages to the user after Revit loads one of your server's resources.

ExternalResourceReference

This is the class which defines an external resource. It contains:

- A GUID indicating which server it belongs with
- A version string
- A user-visible display name for showing in Manage Links or other UI
- A string->string map called the reference information which your server uses to define resources

ExternalResourceLoadContent

When Revit asks your server for a resource, this is the class you will use to pass back the link data. Or more specifically, Revit will send your server a *subclass* of this class, appropriate to the type of data it needs. Two examples:

- For Revit links, Revit needs to receive a path to a local file on disk to read in the link. Your server will download or cache the link locally and return that local path to Revit via a LinkLoadContent object.
- For keynote tables, Revit can build the table from a set of strings. Your server will send Revit a set of name/parent/value keys via a KeyBasedTreeEntriesLoadContent.

ExternalResourceLoadContext

Revit sends an instance of this class to your server when it requests a resource. It contains the following information:

- A LoadOperationType value indicating whether the load request is Automatic or Explicit. An explicit request is based upon user action – adding a new link, reloading, or Reload from. An automatic reload occurs on, for example, file open.
- An ExternalResourceReference representing the currently-loaded resource, if one exists.
- A ModelPath indicating the path of the host model.

Knowing the LoadOperationType can be helpful if you wish to provide different functionality on file open. You may wish to skip certain error messages, or avoid doing time-consuming downloads.

Note: The function GetCallingDocumentModelPath(), which returns the host path, can throw an exception if there is no path. This can occur when the host document is "Project 1" and has not yet been saved. You will need to catch exceptions when calling this function.

ExternalResourceLoadStatus

An enum indicating the status of an IExternalResourceServer's attempt to load a resource.

Its values are:

0.) Success
1.) Failure
2.) Uninitialized (treated as failure)
3.) ResourceAlreadyCurrent
4.) CannotFindServer
5.) ServerThrewException

ExternalResourceLoadData

A wrapper class passed into IExternalResourceUIServer.HandleLoadResourceResults(). It contains both the inputs and outputs to HandleLoadResourceResults():

- An ExternalResourceLoadContent object containing the results of the call to IExternalResourceServer.LoadResource()
- An ExternalResourceReference indicating the resource that Revit asked your server to load
- The ExternalResourceType of the resource
- An ExternalResourceLoadStatus indicating whether the load request succeeded or failed
- A boolean property, ErrorsReported, indicating whether the UI server has handled its own errors or wants Revit to do default error messaging
- The GUID associated with the load request
- An ExternalResourceLoadContext object identifying the Revit context surrounding the load request

ExternalResourceServerExtensions

Some external resource types have type-specific functionality. (For example, the “Open (and Unload)” feature is only applicable to Revit links.) For each resource type that needs special handling, there is a <LinkType>Operations class which holds callbacks appropriate to the type. You as the server implementer set these callbacks with the behavior you want your server to have. The ExternalResourceServerExtensions object holds the “operations” classes. You can get the ExternalResourceServerExtensions object from an IExternalResourceServer by calling GetTypeSpecificServerOperations.

Both Revit and DWG links have type-specific functionality. For DWG links, there is shared coordinates handling. For Revit links, there is shared coordinates handling, and opening a local copy of the file on “Open (and Unload)”.

ExternalResourceServerUtils

This is a static class which contains methods for checking whether a resource belongs to a server which can handle a specific resource type. For example, ExternalResourceServerUtils.ServerSupportsRevitLinks() takes an ExternalResourceReference as an input and returns true if that resource’s server can provide Revit Links. These functions are used for error checking in various parts of Revit.

ExternalResourceServiceUtils

This is a static class which contains the method GetServersByType(), which allows you to get all registered IExternalResourceServers which provide a given type of resource.

ExternalResourceUtils

This is a static class which allows you to get all elements corresponding to ExternalResourceReferences from a document. You can also get all ExternalResourceReference elements by resource type.

Element

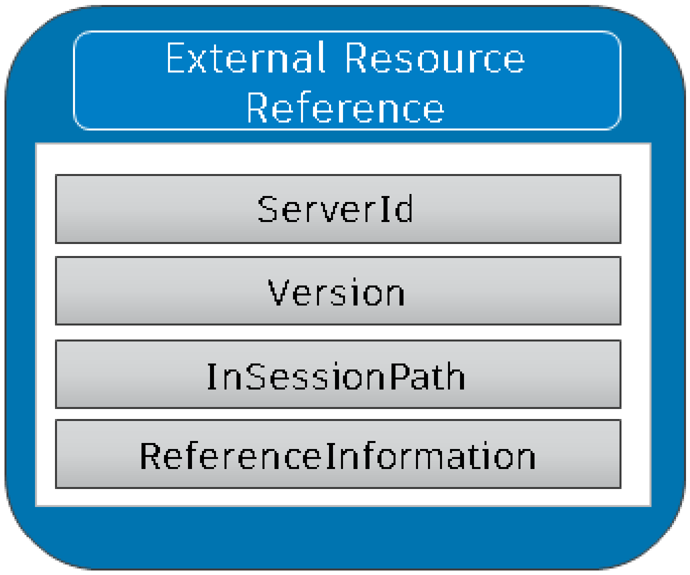
While the Element class is not new, it has several external resource methods:

- RefersToExternalResourceReferences() – Returns true if the element is an external resource reference element.
- GetExternalResourceReferences() – Gets any ExternalResourceReference objects associated with the element and returns them.
- RefersToExternalResourceReference(ExternalResourceType) – Returns true if the element is an external resource reference element of the given type.
- GetExternalResourceReference(ExternalResourceType) – Gets the ExternalResourceReference associated with the element (of the input type) and returns it.

The two “get” functions will throw exceptions if the element does not have a resource (of the appropriate type.)

Detailed class explanations - ExternalResourceReference

The ExternalResourceReference is what you will use to define a resource that your server can provide.



Important properties and methods

ExternalResourceReference.ServerId - Every ExternalResourceReference must contain the GUID of the server it is related to. This value should match the value of IExternalResourceServer.GetServerId(). (While it might be the case that multiple servers can provide the same data, every ExternalResourceReference is associated with one server.)

ExternalResourceReference.Version – A string indicating the version of the resource. It is up to your server to decide how to format the string. (Sequential numbers, timestamps, etc.) You will use this property to tell Revit whether a given resource is “up to date” or not. Revit can skip redundant resource loads in some cases. It is possible for your server to provide multiple versions of the same resource; this is your choice as the server creator.

ExternalResourceReference.InSessionPath – The user-visible path of the resource. In Manage Links or other UI, Revit displays external resources as <ServerName>://<FolderName(s)>/<ResourceName>.<extension>. (For example: MyWebServer://Hospital/EastWing/OuterShell.rvt.) This allows you to present something readable to the user without having to expose your internal structure. The server name will be the value returned by IExternalResourceServer.GetShortName(). When creating an ExternalResourceReference, do not include the server name at the beginning; Revit will automatically add it. (To continue the example, you would pass in “Hospital/EastWing/OuterShell.rvt” and Revit would take care of adding “MyWebServer://”.)

ExternalResourceReference.GetReferenceInformation() – This is a string->string map which contains whatever information your server needs to uniquely identify one of the resources it provides.

Structuring your reference information map

The reference information map should only contain *permanent* information about the resource. You will not be able to edit the resource information inside IExternalResourceServer.LoadResource(). You cannot, for example, add a local path for each new user who opens a central model. If you need different information for different users and that information cannot be calculated at load time, you will need to store it elsewhere – in a config file for your server or something similar. If underlying data about a resource is likely to change (because of renaming on the cloud, for example) it may be best to store a GUID as a permanent identifier.

Reference information can be as complicated or as simple as your server needs. A server which provides a very small number of resources might just number them. For a string-based resource type such as the keynote table, the map could theoretically contain the entire table. (We do not recommend this, but you could potentially do it.) For Revit’s built in server, which handles local-disk files, we store the path and the path type (relative or absolute.)

Creating resources

The ExternalResourceReference constructor takes in the serverId, referenceInformation, version, and inSessionPath. Note that you cannot set this data later. Revit will update the version and inSessionPath after a resource load, but you cannot set those values directly in the ExternalResourceReference.

There is also a separate constructor, ExternalResourceReference.CreateLocalResource(), which will create a reference to a local-disk file.

Comparing resources
One function you will have to implement on the server is IExternalResourceServer.AreSameResources(), which takes the reference maps from two resources. This function is used to see if a link already exists in a file.

This function is intended to return true if the underlying resource is the same. As such, you may or may not want to do a simple map comparison. For the local-disk file case, Revit considers the final absolute path of the resource to be what counts, so we must take care when comparing a relatively-pathed resource to an absolutely-pathed one.

Other notes
ExternalResourceReferences do not have types associated with them. When Revit asks your server for a resource or resources, it will tell you what type to provide, but the ExternalResourceReference object itself is not associated strongly with any link type.

Detailed class explanations – IExternalResourceServer
IExternalResourceServer tells Revit what resources it has available, and gives Revit those resources when asked.

Methods for you to implement
In addition to the normal IExternalServer methods, you will need to implement:

IExternalResourceServer.SupportsExternalResourceType() – This method takes an ExternalResourceType as input. Return true if your server can provide any resources of that type, and false otherwise. When the user is browsing for resources to add or load, Revit will only call your server if it supports the relevant type.

IExternalResourceServer.LoadResource() – Provide the requested resource to Revit. This function has its own section below.

IExternalResourceServer.IsResourceWellFormed() – Return true if the input ExternalResourceReference is formatted properly for your server. You should *not* check whether this resource is present – just whether the reference information is in a format that your server can understand. This method is used to validate input arguments to several other methods.

IExternalResourceServer.GetInSessionPath() – Return a display path for the input resource which is appropriate for this session of Revit. Revit will call this method when loading or displaying a resource. It allows you to update the user facing name of the resource after the ExternalResourceReference is initially created. This may be useful if, for example, the resource has been renamed on the cloud, or if you adjust the name to be specific to the locale of the local machine.

IExternalResourceServer.GetResourceVersionStatus() – Return whether the input resource represents the most recent version of the resource which your server can provide. Revit will skip some resource loads if the resource is already up-to-date. (Revit links, which much be loaded on file open, are an exception.)

IExternalResourceServer.AreSameResources() – Return whether the referenceInformation from two ExternalResourceReferences correspond to the same resource. This function is used by Revit to determine whether a given model already contains the resource.

IExternalResourceServer.SetupBrowserData() – Revit calls this method to determine what resources your server can provide. These resources will be displayed in the file navigation dialog under the “External Resources” button on the Places bar.

IExternalResourceServer.GetIconPath() – Return a path to the icon which Revit should display next to your server name in the file navigation dialog. The path must be an absolute path to an icon file containing 48x48, 32x32, and 16x16 pixel images. Returning a valid path is optional; Revit will use a default server icon if no path is provided or the image files are invalid.

IExternalResourceServer.GetShortName() – Return a name to be used as the prefix to resource display names in Manage Links and other UI. (For example, in “MyWebServer://Hospital/EastWing/OuterShell.rvt”, “MyWebServer” is the ShortName of a server.) This name cannot be empty, must be unique across IExternalResourceServers in the session, and cannot be a reserved name of another Autodesk service. (So no naming your server “RSN” even if you provide files from a Revit Server.)

IExternalResourceServer.GetInformationLink() – Provide a URL at which users can access support or more information about your servers. Revit will store this link in the model so that it is available if opened by another user without your server. If you are at all concerned about a user without your addin opening a model with links you provide, it’s important to provide a value here.

IExternalResourceServer.GetTypeSpecificServerOperations() – Revit will call this method when attempting to access some functionality which is not applicable to all link types. (For example, when saving shared coordinates changes back to a Revit or DWG link.) Revit will pass in a RevitServerExtensions object. You should set callbacks for the functionality your server supports and return. There is further discussion of this function in the “Special Cases” section later in this document.

IExternalResourceServer.LoadResource()

This function is where the actual “work” of the server happens.

Inputs:

- loadRequestId – A GUID which uniquely identifies the load request. This is useful for matching error results to load requests later in IExternalResourceUIServer.HandleLoadResourceResults().
- resourceType – An ExternalResourceType indicating what type the resource is.
- desiredResource – An ExternalResourceReference representing the resource your server should provide.
- loadContext – An ExternalResourceLoadContext object containing info about the Revit context in which this load request was called. (Among other things, it contains the currently-loaded ExternalResourceReference, if this is a Reload or Reload From instead of a new link being created.)

Input/output:

- loadResults – A subclass of ExternalResourceLoadContent appropriate to the type of resource being requested. Your server should fill in this class with the data Revit needs to create or load the resource.

All subclasses of ExternalResourceLoadContent contain fields for version, indicating the version of the resource which your server is providing, and loadStatus, which is an ExternalResourceVersionStatus enum object indicating whether the server was able to provide the resource.

Loading Keynotes and Assembly Code Tables

The keynote and assembly code tables consist of sets of strings. You can build these tables server-side and pass them back to Revit via the KeyBasedTreeEntriesLoadContent. There are two mechanisms for creating a table:

1.) Read the table from a file on disk. You can call KeynoteEntries.LoadKeynoteEntriesFromFile() for keynotes, or ClassificationEntries.LoadClassificationEntriesFromFile() for an assembly code table. Both methods take a file path and a KeyBasedTreeEntriesLoadContent object as inputs.
2.) Create the table one line at a time via KeyBasedTreeEntriesLoadContent.AddEntry().

Either way, you will need to “finish” the table by calling KeyBasedTreeEntriesLoadContent.BuildEntries(). Revit will throw an exception if you try to set the loadStatus before building the table.

Loading Revit and CAD links

It is not possible to fully create a Revit or CAD link server-side. Instead, you will put a copy of the link somewhere locally, and return to Revit the local path to the file. Revit will pass in a LinkLoadContent object. You should call LinkLoadContent.SetLinkDataPath() to tell Revit where it can find the file. After LoadResource() returns, Revit will load the file from disk.

Revit will store the LinkLoadResults associated with the internal load from disk in the LinkLoadContent, and pass this data into IExternalResourceUIServer.HandleLoadResourceResults().

LoadResource() does not have a Document

Revit does *not* pass a Document into the call to LoadResource. All of the info you need from Revit should be passed in via the “desiredResource” or “loadContext” arguments. Revit does not expect the document to be changed during the call to LoadResource, so please do not try to search through the open documents to make changes. There may or may not be a transaction open. (You do not need to explicitly read a Revit or CAD link from disk; Revit will do that after LoadResource() returns.)

Using the LoadRequestId to store data for the UI Server

You may find that your UI server needs more information than Revit passes to IExternalResourceUIServer.HandleLoadResourceResults() by default. This is fairly common if you have any custom errors. In this case, you can store the information server-side. Revit will pass the same loadRequestId to HandleLoadResourceResults(), and you can use it to match to the data you have stored from LoadResource().

Detailed class explanations – IExternalResourceUIServer

IExternalResourceUIServer handles displaying error messages or other UI to the user once the IExternalResourceServer has tried to load a resource.

Methods for you to implement

In addition to the normal IExternalServer methods, you will need to implement:

IExternalResourceUIServer.GetDBServerId() – Return the id of the IExternalResourceServer associated with this UI server. Revit will use this function to determine which load results to give your UI server.

IExternalResourceUIServer.HandleBrowseResult() – Respond to errors that occur when the user is browsing for resources in Revit's UI. The documentation for this function states "It is recommended that the server only respond in the case of a critical error." In practice, it may be more annoying than informative to respond even to critical errors – if there is something wrong with the server in general, there may be a flood of error messages.

IExternalResourceUIServer.HandleLoadResourceResults() – Implement this function to handle the results of loading resources from your server.

Input:

- Document – the document into which the resources were loaded.

Input/output:

- A collection of ExternalResourceLoadData objects representing the resources which your server was asked to load. This class contains all the data passed into IExternalResourceServer.LoadResource(), plus some. It contains a flag which your UI server can set to indicate whether it handled any UI for the link or not. Also, for Revit or CAD links, the ExternalResourceLoadContent object within will contain a LinkLoadResults letting you know whether Revit's internals were able to successfully read the file from disk.

Within HandleLoadResourceResults(), you can do whatever post-processing is appropriate for your links. For example, you might tell a user that their trial is expired, let them know which links were already up to date, ask them to re-path missing links, or do nothing, depending on your particular server.

Getting more information from the DB-level server

If your server needs more information than is passed into HandleLoadResourceResults(), you can store it server-side at resource request time and retrieve it from within HandleLoadResourceResults() using the GUID via ExternalResourceLoadData.GetLoadRequestId(). This GUID will be identical to the one passed into the corresponding call to IExternalResourceServer.LoadResource() and can be used to match up requests.

Handling multiple links at once

On file open, Revit will collect the results of all resource load attempts and call HandleLoadResourceResults only once. This allows you to avoid repeated messaging if multiple links have the same problem – you can post a single "These links failed to load" message rather than several "Link X failed to load" messages.

Retrying failed load attempts

HandleLoadResourceResults() gives you a Document object, so that you can retry a resource load if that is appropriate. For example, if a DWG link fails to load because the user is not logged into your service, you might prompt them for a username/password and then call CADLinkType.LoadFrom(). Note that Revit will not call HandleLoadResourceResults() if the source of the request was an API function, so you should expect to display any UI from the second load attempt fully on your own.

Deferring error handling back to Revit
The ExternalResourceLoadData objects have a property called ErrorsReported. If set to true, Revit will not display any internal error messaging for (This value defaults to false, so don't forget to set it to true or you will get both external and internal messaging for your links.)

Automatic versus explicit loads
If your server displays success messages, you should consider skipping them on LoadOperationType.Automatic loads. On file open, users typically want to know only about serious errors. (You may even wish to skip some error conditions!)

Link loads which originate from the API
Revit does *not* call HandleLoadResourceResults() if the source of a resource load was a DB-layer API call (for example, RevitLinkType.LoadFrom().) If you expect third parties to load your resources from the API, you should provide a method they can call to display your UI.

Not implementing a UI server
The UI server is split out as a separate class to allow IExternalResourceServer to work with DBApplications. Technically, it is optional. However, there is no DB-level way to provide feedback from a call to LoadResource, so we recommend providing at least a simple implementation.

Behavior

Special cases

Nested Revit links
Nested Revit links require careful handling. If a user tries to upload a multi-level link tree to your service, you will have to fix the paths of the intermediate links. This can involve having to open and "Reload From" many links in order to fix their paths. Unfortunately, eTransmit cannot be used to automate this, because eTransmit cannot set an External Resource Reference.

Handling error messages for nested links can also be complicated. In IExternalResourceServer.HandleLoadResourceResults(), Revit will pass your server a collection of ExternalResourceLoadData objects which contain LinkLoadResults for any Revit links. The LinkLoadResults object will be for the top-level link in a Revit link chain, *even if* that link was not handled by your server. You may need to traverse the tree in the LinkLoadResults to find whether your specific link succeeded or failed. Please do not report results for links which your server did not handle.

Shared coordinates
You can use shared coordinates with DWG and Revit links. Revit will notify you that shared coordinates changes have been saved to one of your links via the `onLocalLinkSharedCoordinatesSaved` method in the `IOnLocalLinkSharedCoordinatesSavedCallback` callback.

To set a callback, implement your own callback class which implements IOnLocalLinkSharedCoordinatesSavedCallback. Set this callback in the RevitLinkOperations and/or CADLinkOperations class, depending on which link types you wish to handle.

Revit does not provide a mechanism to automatically re-upload the changed link to your service. You will have to implement this as appropriate for your service.

"Open and Unload" for Revit links

When an end user tries to use "Open (and Unload)" on an External Resource link, Revit will notify you via the IGetLocalPathForOpenCallback. You should provide Revit with a local path on disk at which Revit can open the file. (Typically, this is the same path from which Revit has loaded the link, but some implementations make a copy in a separate location, or provide a different user-facing filename.)

Revit will not specifically notify you when the document is saved or closed, but you can subscribe to the normal events for document save or close.

Missing servers

Revit models containing your resources may go to other users who don't have your addin. When this happens, the user will get the "Missing Addins" dialog. Revit will give the user as much of the following information as it has available:

- The names of the resources which couldn't be loaded
- The name of the missing server
- The vendor id associated with the missing server
- The vendor URL associated with the missing server

When Revit loads resources, it tries to store the info from IExternalResourceServer.GetServerName(), IExternalResourceServer.GetVendorId(), and IExternalResourceServer.GetInformationLink() within the model in case the server is missing at some later time.

(We have occasionally seen this dialog come up for local-disk resources when the user's install has not finished properly and the local resource server is not set up correctly. If this happens to you, please run the installer again!)

Limitations of the External Resource framework

eTransmit and External Resources

ETransmit cannot find links which are External Resource References, unless they are handled by the built-in file server. This also means that TransmissionData.ReadTransmissionData() and TransmissionData.WriteTransmissionData() will not find or write to external resource links.

IFC links

As of Revit 2018.2, there is a bug which causes all IFC links to be internally handled by the built-in server. They will still display as coming from the correct server, but the bug prevents the use case where different users have different local file paths.

Misc

The built-in server

The server which handles local files on disk uses ExternalResourceReferences like any other server. It has the server GUID BD4F0F53-394A-4468-B37E-1E7949013382. The reference information has two fields:

- "Path" – the relative or absolute path to the file on disk
- "PathType" – Either "Absolute", "Relative to Central Model", or "Relative to Library Locations", corresponding to PathType.Absolute, PathType.Relative, and PathType.Content respectively.

You can create these resources by hand but ExternalResourceReference.CreateLocalResource() is likely more convenient. Most link API functions which take a file path have a corresponding version which takes an ExternalResourceReference.

Example source code

DynamoDBKeynotesServer

An IExternalResourceServer server that creates and loads key notes entries stored in Amazon DynamoDB. These entries can we used by a Revit user by loading it from the KeyNotes settings dialog. Once loaded, an element can be annotated with any of the key note entries created by the server.

Supporting code is also uploaded in the additional course material for the talk. The following code sample allows a user to load a keynotes from a DynamoDB table by browsing in Keynotes settings dialog inside Revit. Snapshots of DynamoDB table, and file browser in Revit is also included below after the code snippets.

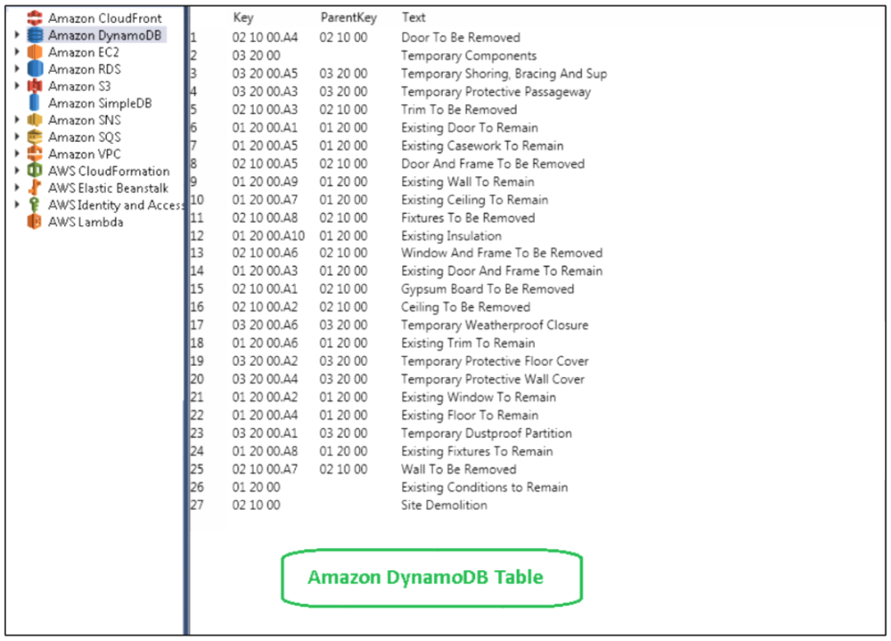
Code snippets

```
using System;
using System.Collections.Generic;
using System.Linq;
using Autodesk.Revit.DB;
using Autodesk.Revit.DB.ExternalService;

namespace KeynotesDemo
{
    public class DynamoDBKeynotesServer : IExternalResourceServer
    {
        public DynamoDBKeynotesServer() { }
        public System.Guid GetServerId() => Application.DBServerId;
        public ExternalServiceId GetServiceId() => ExternalServices.BuiltInExternalServices.ExternalResourceService;
        public string GetShortName() => ServerName;
        public System.String GetName() => ServerName;
        public static string ServerName => "DynamoDBKeynotes";
        public System.String GetVendorId() => "AU Demo";
        public System.String GetDescription() => "Provides Keynotes from DynamoDB";

        public bool SupportsExternalResourceType(ExternalResourceType resourceType)
        {
            return (resourceType == ExternalResourceTypes.BuiltInExternalResourceTypes.KeynoteTable);
        }

        public String GetInSessionPath(ExternalResourceReference resourceReference, String originalPath)
        {
            return ServerName + "://" + resourceReference.GetReferenceInformation()["TableName"] + ".txt";
        }

        public void LoadResource(Guid loadRequestId, ExternalResourceType resourceType,
           ExternalResourceReference resourceReference, ExternalResourceLoadContext loadContext,
           ExternalResourceLoadContent content)
        {
            KeyBasedTreeEntriesLoadContent kdrlc = (KeyBasedTreeEntriesLoadContent)content;
            string tableName = resourceReference.GetReferenceInformation()["TableName"];

            DBUtils.GetAllDBItems(tableName).ForEach(item =>

            {
                kdrlc.AddEntry(new KeynoteEntry(item.Key, item.ParentKey, item.Text));
            });

            kdrlc.BuildEntries();
            kdrlc.LoadStatus = ExternalResourceLoadStatus.Success;
            return;
        }

        public void SetupBrowserData(ExternalResourceBrowserData browserData)
        {
            Application.DynamoDBTables.ForEach(table =>
            {
                IDictionary<String, String> referenceInformation = new Dictionary<String, String>();
                referenceInformation["TableName"] = table;
                browserData.AddResource(table + ".txt", "1", referenceInformation);
            });
        }

        public bool AreSameResources(IDictionary<string, string> refInfo1, IDictionary<string, string>
refInfo2)
        {
            if ((refInfo1 == null) != (refInfo2 == null))
                return false;
            if ((refInfo1 == null) == (refInfo2 == null))
                return true;

            if (refInfo1.Count != refInfo2.Count)
                return false;
            return refInfo1.OrderBy(pair => pair.Key)
               .SequenceEqual(refInfo2.OrderBy(pair => pair.Key));
        }

        public bool IsResourceWellFormed(ExternalResourceReference extRef) => true;
        public string GetIconPath() => string.Empty;
        public ResourceVersionStatus GetResourceVersionStatus(ExternalResourceReference err) =>
ResourceVersionStatus.OutOfDate;
        public String GetInformationLink() => "http://www.autodesk.com";
        public void GetTypeSpecificServerOperations(ExternalResourceServerExtensions extensions) { }
    }
}
```

Supporting code is also uploaded to course material.

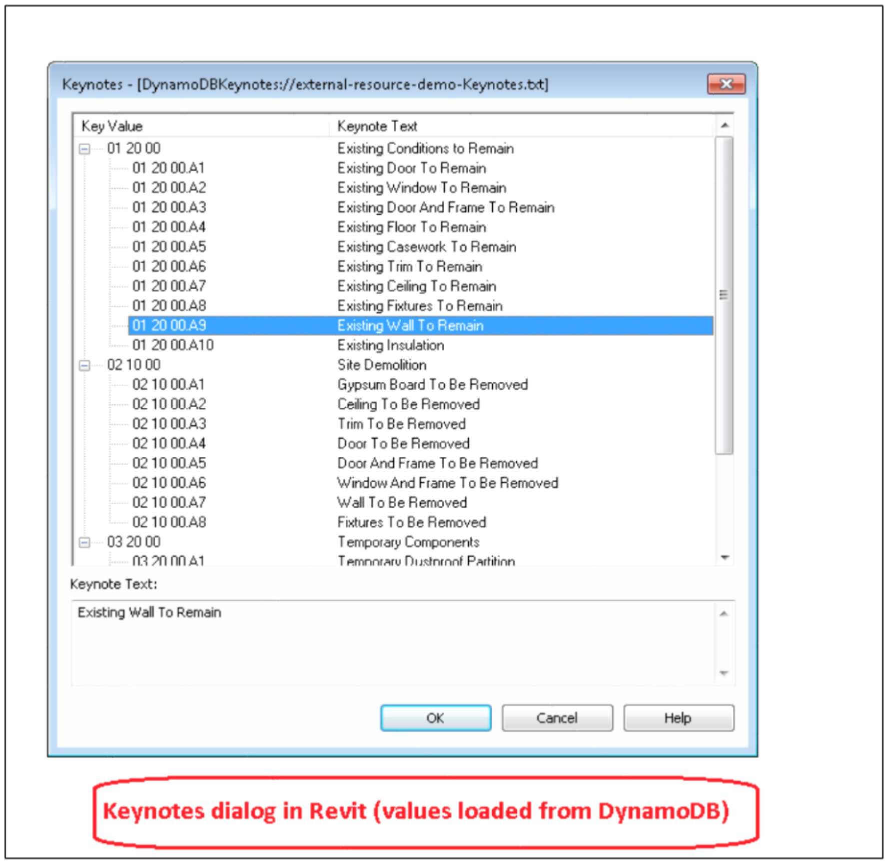
Amazon DynamoDB and Revit Keynotes dialog snapshot.

A snap shot of DynamoDB table used in the current example. Revit loads the values from cloud and to be used by Keynotes.



	Key	ParentKey	Text
1	02 10 00.A4	02 10 00	Door To Be Removed
2	03 20 00		Temporary Components
3	03 20 00.A5	03 20 00	Temporary Shoring, Bracing And Sup
4	03 20 00.A3	03 20 00	Temporary Protective Passageway
5	02 10 00.A3	02 10 00	Trim To Be Removed
6	01 20 00.A1	01 20 00	Existing Door To Remain
7	01 20 00.A5	01 20 00	Existing Casework To Remain
8	02 10 00.A5	02 10 00	Door And Frame To Be Removed
9	01 20 00.A9	01 20 00	Existing Wall To Remain
10	01 20 00.A7	01 20 00	Existing Ceiling To Remain
11	02 10 00.A8	02 10 00	Fixtures To Be Removed
12	01 20 00.A10	01 20 00	Existing Insulation
13	02 10 00.A6	02 10 00	Window And Frame To Be Removed
14	01 20 00.A3	01 20 00	Existing Door And Frame To Remain
15	02 10 00.A1	02 10 00	Gypsum Board To Be Removed
16	02 10 00.A2	02 10 00	Ceiling To Be Removed
17	03 20 00.A6	03 20 00	Temporary Weatherproof Closure
18	01 20 00.A6	01 20 00	Existing Trim To Remain
19	03 20 00.A2	03 20 00	Temporary Protective Floor Cover
20	03 20 00.A4	03 20 00	Temporary Protective Wall Cover
21	01 20 00.A2	01 20 00	Existing Window To Remain
22	01 20 00.A4	01 20 00	Existing Floor To Remain
23	03 20 00.A1	03 20 00	Temporary Dustproof Partition
24	01 20 00.A8	01 20 00	Existing Fixtures To Remain
25	02 10 00.A7	02 10 00	Wall To Be Removed
26	01 20 00		Existing Conditions to Remain
27	02 10 00		Site Demolition

Amazon DynamoDB Table



Keynotes - [DynamoDBKeynotes://external-resource-demo-Keynotes.txt]

Key Value	Keynote Text
01 20 00	Existing Conditions to Remain
01 20 00.A1	Existing Door To Remain
01 20 00.A2	Existing Window To Remain
01 20 00.A3	Existing Door And Frame To Remain
01 20 00.A4	Existing Floor To Remain
01 20 00.A5	Existing Casework To Remain
01 20 00.A6	Existing Trim To Remain
01 20 00.A7	Existing Ceiling To Remain
01 20 00.A8	Existing Fixtures To Remain
01 20 00.A9	Existing Wall To Remain
01 20 00.A10	Existing Insulation
02 10 00	Site Demolition
02 10 00.A1	Gypsum Board To Be Removed
02 10 00.A2	Ceiling To Be Removed
02 10 00.A3	Trim To Be Removed
02 10 00.A4	Door To Be Removed
02 10 00.A5	Door And Frame To Be Removed
02 10 00.A6	Window And Frame To Be Removed
02 10 00.A7	Wall To Be Removed
02 10 00.A8	Fixtures To Be Removed
03 20 00	Temporary Components
03 20 00.A1	Temporary Dustproof Partition

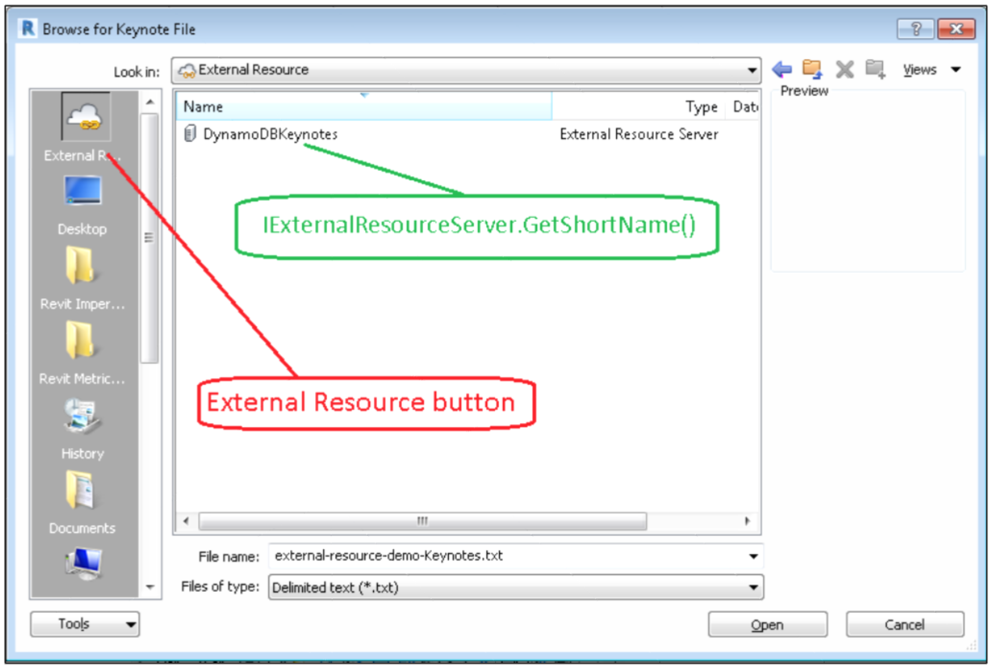
Keynote Text:

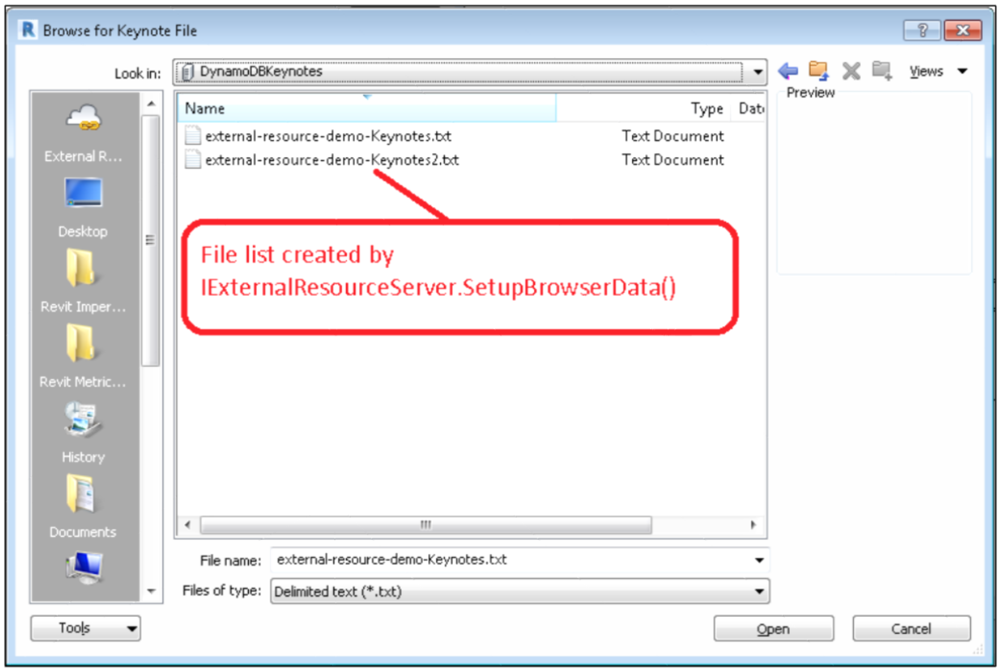
Existing Wall To Remain

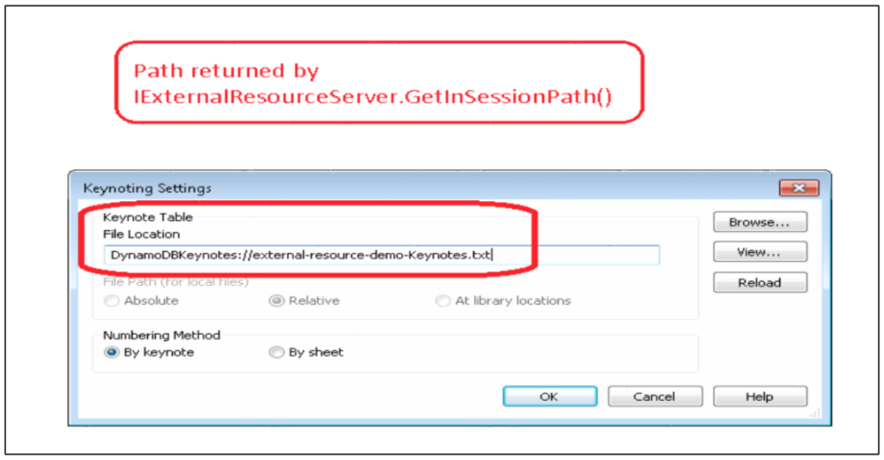
Keynotes dialog in Revit (values loaded from DynamoDB)

File browser

The user is presented this file browser dialog while attempting to change keynotes settings. The External Resource button on the top left is the entry point for the UI supported by the addin. As marked in the pictures below IExternalResourceServer.GetShortName() and IExternalResourceServer.SetupBrowserData() are responsible for population this dialog.




Browse for Keynote File
Look in: DynamoDBKeynotes
Name
Type
Date
external-resource-demo-Keynotes.txt
Text Document
external-resource-demo-Keynotes2.txt
Text Document
File list created by
IExternalResourceServer.SetupBrowserData()
External R...
Desktop
Revit Imper...
Revit Metric...
History
Documents
File name: external-resource-demo-Keynotes.txt
Files of type: Delimited text (*.txt)
Tools
Open
Cancel
Views
Preview



Keynote Table
File Location
DynamoDBKeynotes://external-resource-demo-Keynotes.txt
File Path (for local files)
Absolute [✓ Relative] At library locations
Numbering Method
[✓ By keynote] By sheet

The function IExternalResourceServer.GetInSessionPath() is responsible for the path shown here in the dialog. This indicates the user's selection for keynotes settings. One thing to note here is that there is no physical file at this location or anywhere else. The values are being read from cloud.

LinksMultiLanguageServer

A simple IExternalResourceServer that controls the display names of the Links as per the current locale of the user. A firm translates and maintains a list of links in multiple languages. Each linked file (CAD or Revit Link) has a unique GUID associated with it. The database maintains the “display names” of each file in all supported languages (English and German in the current sample).

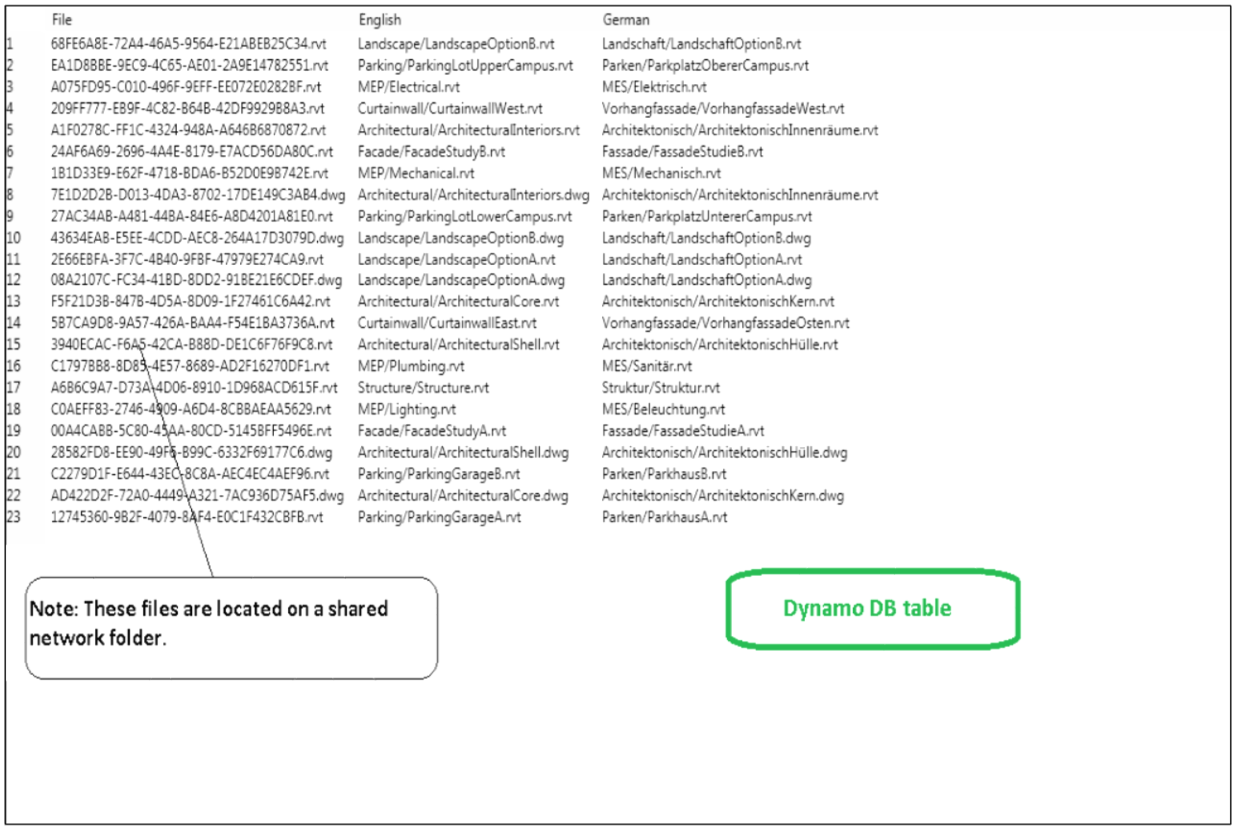
While the real physical files themselves have unreadable GUID based names, the users are presented names from database in a language they read. The files are stored in a common network drive and is accessible to each user. Another side feature of this implementation is that the files can be renamed in their respective languages without making a change in Revit and without having to reload these files in Revit.

The code for LinksMultiLanguageServer and its supporting is also uploaded to the additional course material section. Some pictures of managed links dialog and dynamo db table are given below after the code snippets.

Code snippets

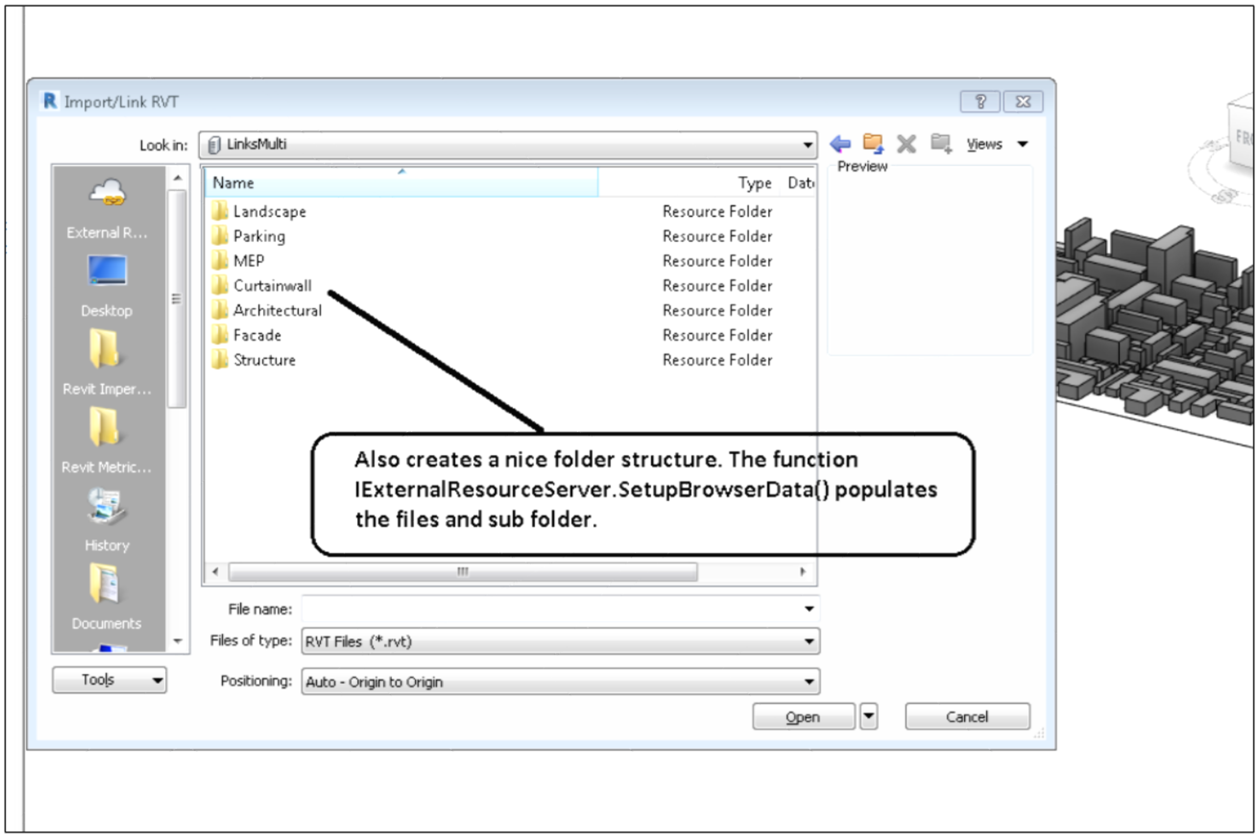
```
using System;
using System.Collections.Generic;
using System.Linq;
using Autodesk.Revit.DB;
using Autodesk.Revit.DB.ExternalService;

namespace LinksDemo
{
    public class LinksMultiLanguageServer : IExternalResourceServer
    {
        public LinksMultiLanguageServer() { }
        public System.Guid GetServerId() => new Guid("be5e293b-26e5-428f-9606-ca16ed046311");
        public ExternalServiceId GetServiceId() => ExternalServices.BuiltInExternalServices.ExternalResourceService;
        public string GetShortName() => ServerName;
        public System.String GetName() => ServerName;
        public static string ServerName => "LinksDemo";
        public System.String GetVendorId() => "AU Demo";
        public System.String GetDescription() => "Provides CAD and Revit Links";

        public bool SupportsExternalResourceType(ExternalResourceType resourceType)
        {
            return ((resourceType == ExternalResourceTypes.BuiltInExternalResourceTypes.CADLink) ||
(resourceType ==ExternalResourceTypes.BuiltInExternalResourceTypes.RevitLink));
        }

        public String GetInSessionPath(ExternalResourceReference resourceReference, String originalPath)
        {
            string file = resourceReference.GetReferenceInformation()["File"];
            DBUtils.DBItem dbItem = DBUtils.FindDBItem(file);
            return GetPath(dbItem);
        }

        public static String GetPath(DBUtils.DBItem item)
        {
            return ServerName + "://" + DBUtils.GetDBItemName(item, Application.CurrentLanguage);
        }

        public void LoadResource(Guid loadRequestId, ExternalResourceType resourceType,
           ExternalResourceReference resourceReference, ExternalResourceLoadContext loadContext,
           ExternalResourceLoadContent content)
        {
            LinkLoadContent linkLoadContent = (LinkLoadContent)content;

            string file = resourceReference.GetReferenceInformation()["File"];
            DBUtils.DBItem dbItem = DBUtils.FindDBItem(file);
            if ((dbItem == null) || (dbItem.Private))
            {
                content.LoadStatus = ExternalResourceLoadStatus.Failure;
                return;
            }

            ModelPath linksPath = ModelPathUtils.ConvertUserVisiblePathToModelPath(Application.BaseFolder + file);
            linkLoadContent.SetLinkDataPath(linksPath);
            content.LoadStatus = ExternalResourceLoadStatus.Success;
            return;
        }

        public void SetupBrowserData(ExternalResourceBrowserData browserData)
        {
            string curPath = browserData.FolderPath;

            DBUtils.GetAllDBItems().ForEach(item =>
            {
                Dictionary<string, string> referenceInformation = new Dictionary<string, string>();
                referenceInformation["File"] = item.File;

                string toDisplay = DBUtils.GetDBItemName(item, Application.CurrentLanguage);
                if (!FileUtils.FileBelongsToFolder(curPath, toDisplay))
                    return;

                string []dirInfo = FileUtils.SplitRelativePath(curPath, toDisplay);
                if (dirInfo.Length == 1)
                    browserData.AddResource(dirInfo[0], "1", referenceInformation);
                else if (dirInfo.Length > 1)
                    if (!browserData.GetSubFolders().Contains(dirInfo[0]))
                        browserData.AddSubFolder(dirInfo[0]);
            });
        }

       public bool AreSameResources(IDictionary<string, string> refInfo1, IDictionary<string, string> refInfo2)
        {
            if (refInfo1.Count != refInfo2.Count)
                return false;
            return refInfo1.OrderBy(pair => pair.Key)
               .SequenceEqual(refInfo2.OrderBy(pair => pair.Key));
        }

        public bool IsResourceWellFormed(ExternalResourceReference extRef) => true;
        public string GetIconPath() => string.Empty;
        public ResourceVersionStatus GetResourceVersionStatus(ExternalResourceReference err) =>
ResourceVersionStatus.OutOfDate;
        public String GetInformationLink() => "http://www.autodesk.com";
        public void GetTypeSpecificServerOperations(ExternalResourceServerExtensions extensions) { }
    }
}
```

Dynamo DB Table



	File	English	German
1	68FE6A8E-72A4-46A5-9564-E21ABEB25C34.rvt	Landscape/LandscapeOptionB.rvt	Landschaft/LandschaftOptionB.rvt
2	EA1D8BBE-9EC9-4C65-AE01-2A9E14782551.rvt	Parking/ParkingLotUpperCampus.rvt	Parken/ParkplatzObererCampus.rvt
3	A075FD95-C010-496F-9EFF-EE072E0282BF.rvt	MEP/Electrical.rvt	MES/Elektrisch.rvt
4	209FF777-EB9F-4C82-B64B-42DF9929B8A3.rvt	Curtainwall/CurtainwallWest.rvt	Vorhangfassade/VorhangfassadeWest.rvt
5	A1F0278C-FF1C-4324-948A-A646B6870872.rvt	Architectural/ArchitecturalInteriors.rvt	Architektonisch/ArchitektonischInnenräume.rvt
6	24AF6A69-2696-4A4E-8179-E7ACD56DA80C.rvt	Facade/FacadeStudyB.rvt	Fassade/FassadeStudieB.rvt
7	1B1D33E9-E62F-4718-BDA6-B52D0E9B742E.rvt	MEP/Mechanical.rvt	MES/Mechanisch.rvt
8	7E1D2D2B-D013-4DA3-8702-17DE149C3AB4.dwg	Architectural/ArchitecturalInteriors.dwg	Architektonisch/ArchitektonischInnenräume.rvt
9	27AC34AB-A481-44BA-84E6-A8D4201A81E0.rvt	Parking/ParkingLotLowerCampus.rvt	Parken/ParkplatzUntererCampus.rvt
10	43634EAB-E5EE-4CDD-AEC8-264A17D3079D.dwg	Landscape/LandscapeOptionB.dwg	Landschaft/LandschaftOptionB.dwg
11	2E66EBFA-3F7C-4B40-9FBF-47979E274CA9.rvt	Landscape/LandscapeOptionA.rvt	Landschaft/LandschaftOptionA.rvt
12	08A2107C-FC34-41BD-8DD2-91BE21E6CDEF.dwg	Landscape/LandscapeOptionA.dwg	Landschaft/LandschaftOptionA.dwg
13	F5F21D3B-847B-4D5A-8D09-1F27461C6A42.rvt	Architectural/ArchitecturalCore.rvt	Architektonisch/ArchitektonischKern.rvt
14	5B7CA9D8-9A57-426A-BAA4-F54E1BA3736A.rvt	Curtainwall/CurtainwallEast.rvt	Vorhangfassade/VorhangfassadeOsten.rvt
15	3940ECAC-F6A5-42CA-B88D-DE1C6F76F9C8.rvt	Architectural/ArchitecturalShell.rvt	Architektonisch/ArchitektonischHülle.rvt
16	C1797BB8-8D85-4E57-8689-AD2F16270DF1.rvt	MEP/Plumbing.rvt	MES/Sanitär.rvt
17	A6B6C9A7-D73A-4D06-8910-1D968ACD615F.rvt	Structure/Structure.rvt	Struktur/Struktur.rvt
18	C0AEFF83-2746-4909-A6D4-8CBBAEAA5629.rvt	MEP/Lighting.rvt	MES/Beleuchtung.rvt
19	00A4CABB-5C80-45AA-80CD-5145BFF5496E.rvt	Facade/FacadeStudyA.rvt	Fassade/FassadeStudieA.rvt
20	28582FD8-EE90-49F6-B99C-6332F69177C6.dwg	Architectural/ArchitecturalShell.dwg	Architektonisch/ArchitektonischHülle.dwg
21	C2279D1F-E644-43EC-8C8A-AEC4EC4AEF96.rvt	Parking/ParkingGarageB.rvt	Parken/ParkhausB.rvt
22	AD422D2F-72A0-4449-A321-7AC936D75AF5.dwg	Architectural/ArchitecturalCore.dwg	Architektonisch/ArchitektonischKern.dwg
23	12745360-9B2F-4079-8AF4-E0C1F432CBFB.rvt	Parking/ParkingGarageA.rvt	Parken/ParkhausA.rvt

Filer Browser



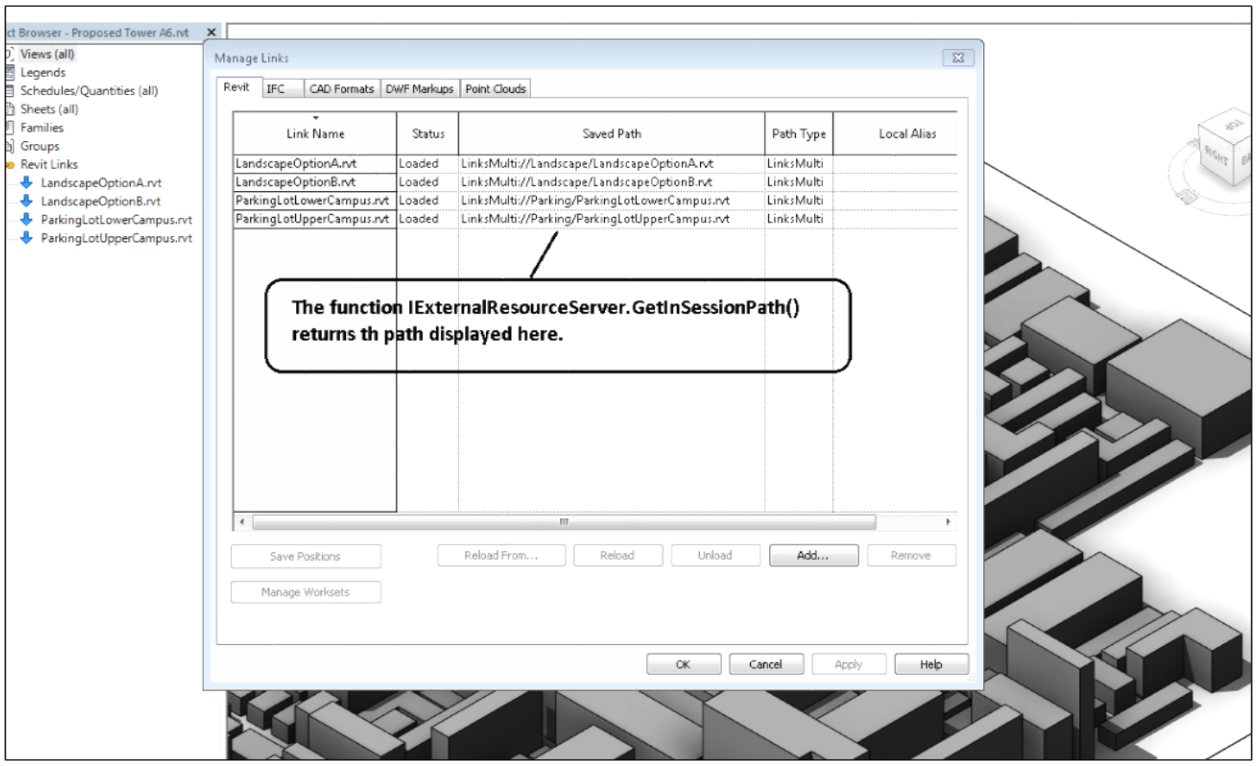
Name	Type
Landscape	Resource Folder
Parking	Resource Folder
MEP	Resource Folder
Curtainwall	Resource Folder
Architectural	Resource Folder
Facade	Resource Folder
Structure	Resource Folder

File name:
Files of type: RVT Files (*.rvt)
Positioning: Auto - Origin to Origin

Also creates a nice folder structure. The function IExternalResourceServer.SetupBrowserData() populates the files and sub folder.

The folder and file structure is created by function IExternalResourceServer.SetupBrowserData().

Manage links dialog

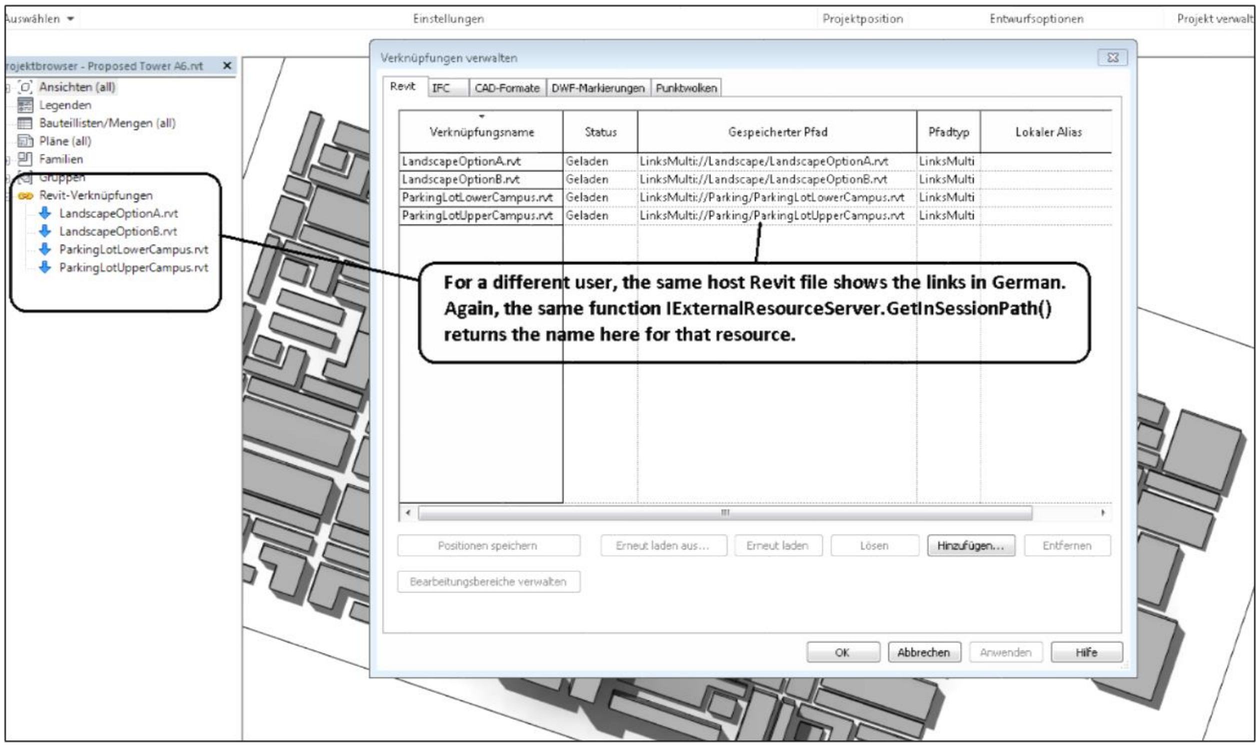


Link Name	Status	Saved Path	Path Type	Local Alias
LandscapeOptionA.rvt	Loaded	LinksMulti://Landscape/LandscapeOptionA.rvt	LinksMulti	
LandscapeOptionB.rvt	Loaded	LinksMulti://Landscape/LandscapeOptionB.rvt	LinksMulti	
ParkingLotLowerCampus.rvt	Loaded	LinksMulti://Parking/ParkingLotLowerCampus.rvt	LinksMulti	
ParkingLotUpperCampus.rvt	Loaded	LinksMulti://Parking/ParkingLotUpperCampus.rvt	LinksMulti	

The function IExternalResourceServer.GetInSessionPath() returns th path displayed here.

The path shown in Mange Links dialog is returned by the function IExternalResourceServer.GetInSessionPath().

Mange Links dialog in German



Verknüpfungsname	Status	Gespeicherter Pfad	Pfadtyp	Lokaler Alias
LandscapeOptionA.rvt	Geladen	LinksMulti://Landscape/LandscapeOptionA.rvt	LinksMulti	
LandscapeOptionB.rvt	Geladen	LinksMulti://Landscape/LandscapeOptionB.rvt	LinksMulti	
ParkingLotLowerCampus.rvt	Geladen	LinksMulti://Parking/ParkingLotLowerCampus.rvt	LinksMulti	
ParkingLotUpperCampus.rvt	Geladen	LinksMulti://Parking/ParkingLotUpperCampus.rvt	LinksMulti	

For a different user, the same host Revit file shows the links in German. Again, the same function IExternalResourceServer.GetInSessionPath() returns the name here for that resource.

The function IExternalResourceServer.GetInSessionPath() returns a name corresponding to the current users locale. In this case we see German in the picture above.

Bonus section! Revit 2018 link API changes

New in 2018.2

RevitLinkType.GetLinkedFileStatus() allows you to determine the load status of a Revit link. Previously, the only way to get the loaded/unloaded state was to call GetExternalFileReference() and query that object. However, C4R links do not have an external file reference object associated with them, and so there was no way to determine whether a C4R link was “unloaded” or “not found”.

New in 2018

Link API additions

External Resource framework additions

IExternalResourceServer can now provide CAD format links. The following new values have been added to ExternalResourceType.BuiltInExternalResourceTypes:

- CADLink

CADLinkType additions

Several new methods have been added to CADLinkType as part of external resource framework enhancements:

- CADLinkType.Reload() - Reloads a CADLinkType from its current location.
- CADLinkType.Reload(CADLinkOptions options) - Reloads a CADLinkType from its current location, including options to control graphic overrides.
- CADLinkType.LoadFrom(String fileName) - Reloads a CADLinkType from a file on disk.
- CADLinkType.LoadFrom(ExternalResourceReference reference) - Reloads a CADLinkType from an external resource server.

ImportInstance additions

Several new methods have been added to create ImportInstance elements, either from an existing link type or a new link type:

- ImportInstance.Create(DWGImportOptions options, String fileName) - Creates a new DWG or DXF type and instance from a file on disk.
- ImportInstance.Create(DWGImportOptions options, ExternalResourceReference reference) - Creates a new DWG or DXF type and instance from an external resource location.
- ImportInstance.Create(ElementId typeId) - Creates a new DWG or DXF instance from an existing CADLinkType.

Methods have also been added to create DGN, SAT, and SKP links from external resource locations.

Shared Coordinates API additions

Acquire and Publish coordinates API additions

Two new methods allow users to acquire and publish shared coordinates:

- Document.AcquireCoordinates() - Acquires project coordinates from the specified link instance. This method accepts both Revit links (RevitLinkInstance) and DWG links (ImportInstance).
- Document.PublishCoordinates() - Publishes shared coordinates to the specified ProjectLocation. This method works only on Revit links.

SiteLocation API additions

Two new read-only properties have been added to provide information on the geographic coordinate system of a SiteLocation. The geographic coordinate system is imported from a DWG file from AutoCAD or Civil 3D. If the SiteLocation has geographic coordinate system information, the latitude and longitude of the SiteLocation will be updated automatically
when the model's Survey Point is moved.

- SiteLocation.GeoCoordinateSystemId - Gets a string corresponding to geographic coordinate system ID, such as "AMG-50" or "Beijing1954/a.GK3d-40" for the SiteLocation. The value will be the empty string if there is no coordinate system specified for the SiteLocation. This property is read-only.
- SiteLocation.GeoCoordinateSystemDefinition - Gets an XML string describing the geographic coordinate system. The value will be the empty string if there is no coordinate system specified for the SiteLocation. This property is read-only.

The new method:

- SiteLocation.IsCompatibleWith() - Checks whether the geographic coordinate system of this site is compatible with the given site.

ProjectLocation API additions

The new method:

- ProjectLocation.Create()

  creates a new ProjectLocation in the document from the given SiteLocation and with the given name.

Revit Link API additions

The new method:

- static RevitLinkInstance.Create(ImportPlacement placement)

  creates a new instance of a linked Revit project (RevitLinkType). Instances will be placed origin-to-origin or by shared coordinates according to the input placement type.

RevitLinkLoadResult rename

The class RevitLinkLoadResult has been renamed to LinkLoadResult. Note that applications referring to RevitLinkLoadResult will need to be recompiled using the new name. The class is otherwise unchanged. The new CADLinkType reload and creation functions will return a LinkLoadResult object.

Shared Coordinates API changes

The following properties have been deprecated and replaced:

Deprecated member	New/replacement member

ProjectLocation.ProjectPosition	ProjectLocation.GetProjectPosition()
	ProjectLocation.SetProjectPosition()

ProjectLocation.SiteLocation	ProjectLocation.GetSiteLocation()