Introduction to the BIM 360 Glue API
Mikako Harada – Autodesk
Manu Venugopal – Autodesk

SD6867 Come and discover the BIM 360 Glue software API. This API (application programming interface) is composed of 2 parts: the Web Services API and the display component. The Web Services API is REST-based, and it enables you to access your BIM 360 Glue software data, such as users, models, and actions. You can imbed the display component, or Viewer API, in your application. Using the BIM 360 Glue software API, you can, for example, automate the updating of projects and models from external applications. BIM 360 Glue software keeps track of a user’s actions as data in additional to traditional model-centric information. You can also use such information to make a report that will keep track of a team’s progress. Of course, BIM 360 Glue software is totally cloud based, so you can connect your own web services applications with the BIM 360 Glue software projects. We will also use examples of use cases to demonstrate what you can do with the BIM 360 Glue software API.

Learning Objectives
At the end of this class, you will be able to:

• Understand the basic structure of the BIM 360 Glue software REST API
• Learn how to use the BIM 360 Glue software web services API to access and modify your BIM 360 Glue software data
• Learn how to use the display components API to embed the BIM 360 Glue software view in your application
• Understand current capabilities and limitations of the BIM 360 Glue software API

About the Speaker
Mikako Harada works as an AEC Technical Lead and Americas Manager for the Developer Technical Services (DevTech) team at Autodesk, providing API technical support to Autodesk Developer Network members worldwide primarily for AEC products. Prior to joining Autodesk, Mikako was a researcher at universities and private research labs and worked in the areas of interactive techniques, optimization and layout synthesis. She is based in the Autodesk office in Manchester, New Hampshire, U.S.

Manu Venugopal is a Building Information Modeling (BIM) enthusiast who focuses on using technology to improve business processes in the architecture, engineering, and construction-facilities management industries. He leads the development and integration of BIM 360 cloud-collaboration platform with ERP (enterprise resource planning) systems, product lifecycle management, document management, project controls, or facilities management systems, and he plays a key role in spreading adoption of Autodesk cloud technologies. Manu plays many roles at Autodesk, including solution architect / customer implementation manager, Certified Scrum Master leading Autodesk teams through agile development processes, and API liaison for Autodesk Developer Network. Manu also designs and architects solutions for customers. Manu received his PhD from Georgia Institute of Technology under the tutelage of Professor
Chuck Eastman. His doctoral research introduced a new software engineering methodology aimed at improving the interoperability of BIM systems, and it also introduced semantic web technologies to BIM.

Prerequisite
This class assumes you have a basic understanding of web programming and familiar with terminologies related to web services. Working knowledge of the REST API is helpful for understanding the discussion. Actual web programming experience may not be necessary if your interests are for making decision, but you will hear a lot of buzzwords if you are not familiar with the web programming.

If you are interested in more on products usage, business discussion and use cases, following classes might be more suited:

- CO5528 – “Building Information Modeling for Construction Safety.”
  By Sijie Zhang - Chevron, Manu Venugopal. Tues, Dec 2, 5-6PM

  By Manu Venugopal. Wed, Dec 3, 1-2PM

- CO5007-R – “Return on Investment of Cloud-Based Collaboration and Mobility for Construction.”
  By Michael Moran, Ken Stowe, Manu Venugopal. Thurs, Dec 4 8-9:30AM. Round Table

- CO6084-L – “BIM 360 Glue: The Super Glue for Your Model Coordination Workflows.”
  By Manu Venugopal, Thurs, Dec 4, 10-11:30AM. Hands-on Lab

Agenda

- BIM 360 Glue API Overview
  - Web Services
  - Display Component
- Where to Begin
  - Resources
  - “My First Glue Application”
- Summary
- Q&A

BIM 360 Glue API Overview

The BIM 360 Software Development Kit (SDK) is a set of tools for developers to interface with the BIM 360 Glue Platform. The SDK allows building scalable integrations by focusing on Security, Data Integrations, Visualizations and Analytics. SDK consists of two components: Web Services API and Display Component or Viewer:
1. Web Services API:

BIM 360 REST APIs provides a RESTful interface to interact with the cloud database. This is basically a data access API that supports *Create*, *Retrieve*, *Update*, and *Delete* (CRUD) operations allowing developers to query and modify various data objects within the Glue platform. Through the BIM 360 REST APIs data integration between external applications such as project management systems, accounting systems or custom developed solutions and BIM 360 platform is realized.

The BIM 360 Glue REST APIs are grouped into different services groups, as below:

- **Security Service**: This service is responsible for the security management of Glue user accounts
- **Project Service**: This service is used to manage the Projects within the Glue platform
- **Model Service**: This service is used to manage the 3D models within the Glue platform
- **User Service**: This service is used for the use and maintenance of Glue user information
- **Action Service**: This service is used to access / manage records (actions) within the Glue platform
- **System Service**: This service is available to get information about the Glue Platform and check availability

Anatomy of an HTTP request

2. Display Component / Viewer:

The Glue Viewer is an embeddable component used to visualize 3D models from the Glue platform within a browser or 3rd party application. Developers can query the BIM 360 platform for project data and pass them on to the viewer component as URL for streaming the BIM content live from the cloud. The figure below shows an example.
BIM 360 Glue SDK | Actions

The Glue Platform stores information about many user interactions or transactions within the system. Hence there is complete audit control as well as the ability to retrieve a “state” of information.

- These operations are referred to as “Actions/Activities”
- Typical Actions would be:
  - Uploading a Model
  - Creating a View on a Model
  - Adding Mark Ups to a Model
  - Creating a Clash report for a Model
- The Glue Web Services API returns many of these actions in API responses
- Actions can be loaded to the viewer to show the user the exact view/state/etc... of a model when the creator of the Action performed the operation

Where to Begin
Glue Developer Site
Let’s start with the documentation. There is a public ADN (Autodesk Developer Network) page for BIM 360 Glue API: www.autodesk.com/developglue

Note that the name of this site spells out as “develop glue”. This is the official Glue API page maintained by ADN. We intend to populate this page as additional resources become available and make it as “one stop” page for Glue API developers. For example, currently it has links to the Glue SDK home page (http://b4.autodesk.com/api)

References

Here is the list of references currently available: (These are linked from the Glue API Developer Page. You don’t need to remember them.)

- Glue API Developer Page mentioned above: www.autodesk.com/developglue
- API documentation main page http://b4.autodesk.com/api
- Samples http://adndevblog.typepad.com/files/glue-api-samples.zip
- Add-ins for AutoCAD (including Verticals), Revit and Navisworks https://b4.autodesk.com/addins/addins.html
- Product Page http://www.autodesk.com/bim360glue

Additional References

- BIM 360 Blog by Manu Venugopal http://the360view.typepad.com
- AEC DevBlog http://adndevblog.typepad.com/aec/

Development Environment

In order to use Glue API, you will need:

- BIM 360 Glue account
- API key and secret
- Programming environment of your choice (e.g., MSVS for C#, Editor for JavaScript)

Since Glue API is REST API, you can choose wider variety of programming choices.

At a moment, not everybody can have access to dev env.

- If you are already Glue customer ➔ You can obtain through the contract
- If you are an ADN member ➔ You can access ADN staging access through ADN. Please request through DevHelp Online
- Note that free trial (https://b4.autodesk.com/freetrial/) can give you an access to try out product itself. But it does not include API access
- What about today’s AU participants? We hope to give you trial access to ADN staging to AU
participant. The instruction will be given during the presentation.

**SDK Sample Demos**

SDK home page includes two very good samples as well as documentations:

- Documentations (and sample download)  
- Latest samples download  
  (once again, Glue SDK page may not be up to date.)
- Two samples:
  - Test Harness – Windows desktop client sample in C#.
  - Sample Web Application – ASP.NET. Web application in C#.
“My First Glue Application”

When we talk about REST API, typical approach is going through the detailed explanation about http protocol, POST, GET, JSON, etc. In this section, we are going to take a different approach of learning by an example and going through the minimum sample.

“Glue API Intro” Labs Exercises

“Glue API Intro” Labs exercises consists of four labs. They are incremental code projects to learn the basics of Glue API:

Lab1 - “Hello Glue World” is the minimum sign-in REST call.
Lab2 – retrieve a model id and display a model in an embedded viewer.
Lab3 – convert the client win forms application to web application version.
Lab4 – add thin JavaScript layer for accessing element properties and selection.

All projects are minimum for clarity and for focusing on understanding the logic of Glue API for learning purposes. They are written C#, API.NET and JavaScript. After the Lab4, the final application looks like below:
Lab1 - “Hello Glue World”

Our THE minimum project is called “Hello Glue World”. In this lab, we learn the basic of REST API call. This is a simple Windows Form application written in C#. The Glue service we use is security/login. This is a sign-in Glue REST call. We use a library called RestSharp for simplicity.

To use RestSharp, in MS Visual Studio 2012, go to:
[Tools] >> [Library Package Manager] >> [Manage NuGet Packages for Solution…]
For more information, please refer to http://restsharp.org/

Login - Request
Login authenticates a user to the Glue platform and returns an authentication token (auth_token) that is used for subsequent service calls.
URL:  https://b4.autodesk.com/api/security/v1/login.{format}

Supported Request Methods: Post

Required Parameters
- Format
Among above required parameters, login_name and password are same as Autodesk ID and should be provided by the user. The company_id is the name of the host. api_key is provided by Autodesk. Normally, it is specific each company and should be shared only among people who needs it.

We'll talk about format, timestamp and signature shortly as we look at the code.

### Login - Response

After you send a request, you can receive response in one of two formats: JSON and XML. You can specify which format you would like to receive the information with when you request. This is one of the parameter “format” we saw earlier.

For example, when you are requesting Login, two types of format for response looks like this:

```plaintext
- JSON

```{json
    "auth_token":"76d47d99102b42e3abb30851e63d1a67",
    "user_id":"36477f1a-3366-44f5-bf0a-3a4a019c52bd"
}

- XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<security_login_response_v1
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <auth_token>76d47d99102b42e3abb30851e63d1a67</auth_token>
    <user_id>36477f1a-3366-44f5-bf0a-3a4a019c52bd</user_id>
</security_login_response_v1>
```

Note that for both cases, line break added for readability. As you can see above JSON is much simpler. In general it’s gaining popularity for its simplicity. For our little app, we are going to stick to JSON because of the same reason.

### Login - Sample Code

Here is a simple code to show the basic flow of login requests. Basically, what we are doing is (1) to fill out the information which are necessary to send a request, (2) make a web request, and (3) parse the response.

```csharp
public string Login(string login_name, string password)
```
{  
  // Calculate signature components.
  string timeStamp = Utils.GetUNIXEpochTimestamp().ToString();
  string signature = Utils.ComputeMD5Hash(apiKey + apiSecret + timeStamp);

  // (1) Build request
  var client = new RestClient();
  client.BaseUrl = baseApiUrl;

  // Set resource/end point
  var request = new RestRequest();
  request.Resource = "security/v1/login.json";
  request.Method = Method.POST;

  // Set required parameters
  request.AddParameter("login_name", login_name);
  request.AddParameter("password", password);
  request.AddParameter("company_id", companyId);
  request.AddParameter("api_key", apiKey);
  request.AddParameter("timestamp", timeStamp);
  request.AddParameter("sig", signature);

  // (2) Execute request and get response
  IRestResponse response = client.Execute(request);

  // (3) Get the auth token.
  string authToken = "Undefined";
  if (response.StatusCode == HttpStatusCode.OK)
  {
    JsonDeserializer deserial = new JsonDeserializer();
    LoginResponse loginResponse = deserial.Deserialize<LoginResponse>(response);
    authToken = loginResponse.auth_token;
  }

  return authToken;
}

### Timestamp and Signature

To define parameters, timestamp and signature, you can us generic helper functions:

- `timestamp = Util.GetUNIXEpochTimestamp()`
- `signature = Util.ComputeMD5Hash(apiKey + apiSecret + timeStamp)`

Time stamp is the number of seconds since January 1st, 1970 00:00:00 GMT (the UNIX epoch). The BIM 360 Glue Platform accepts timestamps up to a configurable number of minutes on either side of the server timestamp, to accommodate reasonable clock drift. Here is the code that you can copy and paste:

```csharp
public static int GetUNIXEpochTimestamp()
{
    TimeSpan tSpan = (DateTime.UtcNow - new DateTime(1970, 1, 1));
```
int timestamp = (int)tSpan.TotalSeconds;
return timestamp;
}

Another is a function that defines a signature. This calculates a hash based on MD5 message-digest algorithm. You can use .NET class System.Security.Cryptography.MD5. The Glue uses MD5 cryptographic hash of the concatenated string from API key + API secret + UNIX timestamp.

Here is an example of building a Glue API signature:

Concatenate the following information:

- API Key: ddbf3f51b3824ecbb824ae4e65d31be4
- API Secret: 12345678901234567890123456789012
- UNIX Timestamp: 1305568169 - (5/16/2011 5:50:36 PM)

Then, the following is the signature base string:

```
ddbf3f51b3824ecbb824ae4e65d31be4123456789012345678901234567890121305568169
```

Create the MD5 cryptographic hash which is sent as the "sig" parameter:

```
b3298cf0b4dc88450d00773b4449ba51
```

For actual calculation of MD5, you can use the function below. It shows a hash based on MD5 message-digest algorithm, which is 128-bit (16-byte) hash value, typically expressed as 32 digit hexadecimal number. Here we use System.Security.Cryptography.MD5. For further explanation of MD5, please refer to [http://en.wikipedia.org/wiki/MD5](http://en.wikipedia.org/wiki/MD5), for example:

```csharp
public static string ComputeMD5Hash(string aString)
{
    // step 1, calculate MD5 hash from aString
    byte[] inputBytes = System.Text.Encoding.UTF8.GetBytes(aString);
    byte[] hash = md5.ComputeHash(inputBytes);

    // step 2, convert byte array to hex string
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < hash.Length; i++)
    {
        sb.Append(hash[i].ToString("x2"));
    }
    return sb.ToString();
}
```

REST Call
To make REST call, we use RestSharp library. It’s easy to use and helps us focus on understanding the logic. For example, to make a client call:

```csharp
    var client = new RestClient();
    client.BaseUrl = baseApiUrl;
```

This sets a REST client with the given Url.

Next, set the request resource and method either Post or Get. e.g.,

```csharp
    var request = new RestRequest();
    request.Resource = "security/v1/login.json";
    request.Method = Method.POST;
```

Required parameters are set using AddParameter() method:

```csharp
    request.AddParameter("login_name", login_name);
```

Finally, call client.Execute(request) to get the response from the web service:

```csharp
    IRestResponse response = client.Execute(request);
```

Once you get a response, you can check the status of the response, and obtain header and response body info:

```csharp
    if (response.StatusCode == HttpStatusCode.OK)
```

**Parse Response**

To pause from response, you can use the following:

```csharp
    JsonDeserializer deserial = new JsonDeserializer();
    LoginResponse loginResponse = deserial.Deserialize<LoginResponse>(response);
    authToken = loginResponse.auth_token;
```

where, `LoginResponse` is defined as:

```csharp
    { 
        public string auth_token { get; set; }
        public string user_id { get; set; }
    }
```

You may use other utilities that may fit your need and that you may want to take advantage of.

Add UI to get a user name and password, and call above function. (A simple Win Form will serve our purpose for this.) In the image below, we added additional text fields to show http request and response. This is for learning purpose.
When you are able to login successfully, you should receive `auth_token` as a part of response. You will be using the `auth_token` for any subsequent call to the Glue. You are ready for the next lab to request a list of projects.

**Lab2 – “Glue API Intro”**

Let’s move on. Next, let’s try to retrieve a model and display it in a viewer. To achieve this, we use the following services:

- Security/login
- Project list
- Model list
- Display component (or viewer)

**Work Flow**

The work flow of our little app is as follows:

- Login – log in action obtains the `auth_token` from the Glue. We have learned this in the Lab1.

- Get a list of projects – we obtain the list of currently available projects. We keep one arbitrary project id (project_id) to further take action on model.

- Get a list of model from a given project >> we obtain the list of models. We keep one arbitrary model_id to further take action with a viewer.
model_id

- Display a model – use display component API to display a model.

**Project List - Request**

Project/List Returns a list of project from the Glue platform.

URL:  [https://b4.autodesk.com/api/project/v1/list.{format}](https://b4.autodesk.com/api/project/v1/list.{format})

Supported Request Methods: GET

Required Parameters

- format
- company_id
- api_key
- timestamp
- signature (sig)
- auth_token (obtained from login)

Doc:  [https://b4.autodesk.com/api/project/v1/list/doc](https://b4.autodesk.com/api/project/v1/list/doc)

We are using JSON as format. The next five parameters are always needed to for many subsequent requests. Note that auto_token is the one we received through the earlier Login request.

**Project List – Response**

Here is the sample response for Project/List:

```json
{
  "project_list":
  [
    {
      "folder_tree":null,
      "project_roster":null,
      "project_id":"The BIM 360 Glue Project ID",
      "project_name":"The name for the project (URL Encoded)",
      "company_id":"The Company ID for the Project",
      "created_date":"The date the Project was added",
      "modify_date":"The date the Project name was last modified",
      "start_date":"The date the Project was started",
      "end_date":"The date the Project ends",
      "last_activity_date":"The last date there was Project activity"
    },
    {
      "folder_tree":null,
      "project_roster":null,
      "project_id":"The BIM 360 Glue Project ID",
      ...
    }
  ],
  "page":1,
}
For our specific little app, we’ll be interested in **project_id** and **project_name**.

### Project List Sample Code

Here is the code sample to obtain a list of project:

```csharp
public List<Project> ProjectList(string authToken)
{
    string timeStamp = Utilities.GetUNIXEpochTimestamp().ToString();
    string signature = Utilities.ComputeMD5Hash(apiKey + apiSecret + timeStamp);

    // (1) Build request
    // set base url and authentication info.
    var client = new RestClient();
    client.BaseUrl = baseApiUrl;

    // Set resource or end point
    var request = new RestRequest();
    request.Resource = "project/v1/list.json";
    request.Method = Method.GET;

    // Add parameters
    request.AddParameter("company_id", companyId);
    request.AddParameter("api_key", apiKey);
    request.AddParameter("timestamp", timeStamp);
    request.AddParameter("sig", signature);
    request.AddParameter("auth_token", authToken);

    // (2) Execute request and get response
    IRestResponse response = client.Execute(request);

    if (response.StatusCode != HttpStatusCode.OK)
    {
        return null;
    }

    // Get a list of projects.
    JsonDeserializer deserial = new JsonDeserializer();
    ProjectListResponse projListResponse =
        deserial.Deserialize<ProjectListResponse>(response);
    List<Project> proj_list = projListResponse.project_list;

    return proj_list;
}
```

You may already see here that there are many part of the code that is more or less understandable from our previous experience with Login. In this example, we return a list of projects. You could easily modify that part to fit your needs, of course.
Let's go to the Model/List request next.

**Model List – Request**

Model/List returns a list of models from the Glue platform.

**URL:** [https://b4.autodesk.com/api/model/v1/list.{format}](https://b4.autodesk.com/api/model/v1/list.{format})

**Supported Request Methods:** GET

**Required Parameters**

- format
- company_id
- api_key
- timestamp
- signature (sig)
- auth_token (obtained from login)
- project_id (obtained from the previous call)

**Doc:** [https://b4.autodesk.com/api/model/v1/list/doc](https://b4.autodesk.com/api/model/v1/list/doc)

**Model List – Response**

Here is an example of Model/List response:

```json
{
  "model_list": [
    {
      "company_id": "The company identifier for this model",
      "project_id": "The Project identifier for this model",
      "model_id": "The model identifier for the model",
      "model_version": 1,
      "model_version_id": "The version identifier for this specific version of the model",
      "model_name": "The name for the model",
      "created_by": "The login_name of the creator of the model",
      "created_date": "The date of model was created",
      "modified_by": "The login_name of the last user to modify the model",
      "modified_date": "Date of last modification",
      ...
      "published": 0
    },
    {
      "company_id": "The company identifier for this model",
      ...
    }
  ]
}
```
Model List Sample Code

Here is the code sample to obtain a list of project:

```csharp
public List<ModelInfo> ModelList(string authToken, string projectId)
{
    string timeStamp = Utils.GetUNIXEpochTimestamp().ToString();
    string signature = Utils.ComputeMD5Hash(apiKey + apiSecret + timeStamp);

    // (1) Build request
    // set base url and authenticatopm info.
    var client = new RestClient();
    client.BaseUrl = baseApiUrl;

    // Set resource or end point
    var request = new RestRequest();
    request.Resource = "model/v1/list.json";
    request.Method = Method.GET;

    // Add parameters
    request.AddParameter("company_id", companyId);
    request.AddParameter("api_key", apiKey);
    request.AddParameter("timestamp", timeStamp);
    request.AddParameter("sig", signature);
    request.AddParameter("auth_token", authToken);
    request.AddParameter("project_id", projectId);

    // (2) Execute request and get response
    IRestResponse response = client.Execute(request);
    if (response.StatusCode != HttpStatusCode.OK)
    {
        return null;
    }

    // Get a list of models.
    JsonDeserializer deserial = new JsonDeserializer();
    ModellListResponse modellListResponse = deserial.Deserialize<ModellListResponse>(response);
    List<ModelInfo> model_list = modellListResponse.model_list;

    return model_list;
}
```
By now, I hope you see the pattern to make REST call and find it simple enough!

Add UI’s to get a project and a model id. For example, add a button to call ProjectList(). Choose an arbitrary project. Add another button to call ModelList() and choose an arbitrary model for this exercise. We should have a project id and model id.

Next, we would like to add a viewer to our little app and display a model.

**Viewer**

The display component, or viewer API allows users to place Glue 3D models on a web page.

**URL:** [https://b2.autodesk.com](https://b2.autodesk.com)

Required Parameters: 5 usual required parameters, plus two ways:

- `&runner=embedded/#` + company_id + `"/"` + project_id + `"/"` + model_id
- `&runner=embedded/#` + company_id + `"/action"` + `"?"` + action_id


Note: as of this writing, the page is not up to date. But it still includes useful information about the idea of viewer.

Response for the viewer is, of course, is graphics, which looks something like this:
Here is a sample code:

```csharp
public string View(string authToken, string projectId, string modelId)
{
    string timeStamp = Utils.GetUNIXEpochTimestamp().ToString();
    string signature = Utils.ComputeMD5Hash(apiKey + apiSecret + timeStamp);

    string callArgs = "";
    // We need these 5 arguments for every subsequent requests.
    // Auth token is returned when you login.
    callArgs += "&company_id=" + HttpUtility.UrlEncode(companyId);
    callArgs += "&api_key=" + HttpUtility.UrlEncode(apiKey);
    callArgs += "&timestamp=" + HttpUtility.UrlEncode(timeStamp);
    callArgs += "&sig=" + HttpUtility.UrlEncode(signature);
    callArgs += "&auth_token=" + HttpUtility.UrlEncode(authToken);
    //
    // Two ways to pass parameters.
    // (1) 
    // (2) 
    // We use #1 here with the saved project and model ids.
    callArgs += "&runner=embedded/#" + company_id + "/" + projectId + "/" + modelId;
    callArgs += "&runner=embedded/#" + projectId + "/" + modelId;
```
Add UI’s to call View() shown above with the project and the model id obtained in previous steps. You should be able to view a model.

Note: viewer components provides JavaScript layer, which may throw script error messages in the environment like win form. To avoid the message, set ScriptErrorsSuppressed as True in the properties of WebBrowser tool.

Lab3 – Glue API Web Intro

So far, we have written a program which is desktop client application. Next, we will write a simple ASP.NET web application. For now, we keep the basic functionality same as Lab2. Later in Lab4, we will add JavaScript layer to have UI interaction. You should be able to reuse most of Glue web services REST call you wrote in the previous labs. In the sample project accompany this class, we put the common Glue call under Glue folder.

Instead of Win Form, we use Web Form when we choose a project in Visual Studio. For this exercise, a simple single web page will be enough. MSVS ASP.NET development environment gives you a set of tools that is analogous to win forms tools. If you have been programming a windows application using visual studio, it will look familiar to you.

Difference is, of course, the UI is a web page. ASP.NET tools starts with “asp:”. We can use <iframe/> to embed a display component, which looks like this:

<iframe id="iframeGlue" title="BIM 360 Glue Display" src="" runat="server"></iframe>

Then in the corresponding code, we set src with the url that we constructed in Glue.View().

protected void ButtonView_Click(object sender, EventArgs e)
{
    string authToken = HttpContext.Current.Session["authToken"] as string;
    string project_id = HttpContext.Current.Session["projectId"] as string;
    string model_id = HttpContext.Current.Session["modelId"] as string;

    string url = Glue.View(authToken, project_id, model_id);
    // embed a viewer in iframe
    iframeGlue.Src = url;
Notice that we are using HttpContent.Current.Session to keep temporary, session specific data. You can apply the same in other tools’ event handling. Image below shows an example of how an application might look.

Lab4 – Glue API Web Intro JS

The last piece to add to our Intro app is an ability to select an element in the viewer and obtain its property.

JavaScript API Layer

To do this, Glue API provide a thin layer of JavaScript API. It is written on top of jQuery library, which is very popular JavaScript library and makes query documents parts or the access of web document object
very easy. The code is included in the accompanying sample. It is called “glue-embedded.js” under Scripts folder.

It uses event handlers and post messages to interface between web page and embedded viewer. GlueEmbedded defines the following functions, which are used to post messages to the viewer:

- Init(iframe)
- setSelection(objectPath)
- getProperties(objectPath)
- getSelectedProperties()
- zoomSelection()

From the Viewer side, it send message event, which you can catch, for example, using $(document).on( message, function(e) {}).

- ‘selectionchanged’
- ‘gotproperties’

The following diagram illustrates flow of the messages.

In our example, we use the following four methods to select an element, get:

- $(document).on(‘selectionchanges’, function(e))
- $(document).on(‘gotproperties’, function(e))
- GlueEmbedded.getSelectedProperties()
- GlueEmbedded.zoomSelection()

Here is the JavaScript code for user interaction portion. Here we set id for a text area to display messages as “messages” in the web page, and “Get Properties”, and “Zoom Selection” buttons as “get_properties” and “zoom_selection” respectively.

```javascript
// view.js
//===----------------------------------------------------------------------
// Welcome to the Glue API intro.
// This is a minimum sample that demonstrates Glue viewer API.
//===----------------------------------------------------------------------
```
$(document).ready(function () {
    // initialize the embedded viewer passing in the target iframe window
    GlueEmbedded.init(window.frames[0]);

    // Setting event handlers to get a messages from the viewer.
    // specify an event handler on the document for the "selectionchanged" from the
    // viewer.
    // The event's eventData property will be set to the array of selected objects
    $(document).on('selectionchanged', function (e) {
        $('#messages').val("object selected: " + JSON.stringify(e.eventData));
    });

    // specify an event handler on the document for the "gotproperties" event from the
    // viewer.
    // The event's eventData property will be set to the collection of object properties
    $(document).on('gotproperties', function (e) {
        $('#messages').val("got properties: " + JSON.stringify(e.eventData));
    });

    // Posting a message to the viewer.
    // When the user click get_properties button, this is called.
    $('#get_properties').on('click', function () {
        GlueEmbedded.getSelectedProperties();
    });

    // This is for zooming in a currently selected element.
    $('#zoom_selection').on('click', function () {
        GlueEmbedded.zoomSelection();
    });
});

The above code is executed when a web page is all loaded. It first initializes the GlueEmbedded, passing
the iframe. We then set two event handlers to receive message from the embedded viewer:
'selectionchanged' and 'gotproperties'. Two buttons whose ids we named as 'get Properties' and
'zoom selection' are set to post messages to the viewer on the user's click event.

In the head of your Web Form should include lines like the following to reflect this additional features:

    <script src="Scripts/jquery-1.11.0.min.js"></script>
    <script src="Scripts/glue-embedded.js"></script>
    <script src="Scripts/viewer.js"></script>
An example of final web page with looks like this:

After a model is displayed, try selecting an element, press [Get Properties] button. You will see a set of properties of a given element in the text area.

Conclusion
In this class, we talked about Glue API’s, its capabilities, and uses cases. We have also showed how to get started with Glue API through a minimum viewer sample.

You have now learned how to:

- Describe the basic structure of the Autodesk BIM 360 Glue REST API
- Use the Glue web services API to access and modify Glue data
- Use the display components API to embed the Glue viewer in their application
- Describe the current capabilities and limitations of the Glue API
Appendix

Q. What do I need to do to try Glue API after this class?

As mentioned earlier, Glue API is not openly available at a moment. You will need to receive API key and secret in order to use API.

- If you are already a Glue customer → request through the contract
- If you are already an ADN member → request through DevHelp Online. ADN members can use ADN sandbox as development environment to experiment with the API.
- For today’s AU Class participants, a sandbox is made available till January 31, 2015:
  - Send an e-mail to one of SD6867 class presenters
  - Will send you keys and an instruction