

# Autodesk View and Data API with Visual Studio

---

## PART 1

### Introduction

The purpose of this article is to enable a technical professional to explore the Autodesk [View and Data API](#) using a development environment based on Visual Studio (VS Express or Premium), IIS Express and .NET. The article describes the visual studio setup of two web servers and sufficient browser and server coding to display a 3D model in a browser (Chrome, Firefox, IE 11), without a plug-in, using Autodesk's LMV (large model viewer). This description covers server to server authentication, cross-origin resource sharing (CORS), AngularJS in the browser and the large model viewer.

The intention of this article is to supplement the very good LMV developer material available on these Autodesk blogs:

- <http://the360view.typepad.com/blog/2015/02/autodesk-view-and-data-api-intro-overview.html>
- [http://adndevblog.typepad.com/cloud\\_and\\_mobile/2014/08/autodesk-view-data-api-javascript-wrapper.html](http://adndevblog.typepad.com/cloud_and_mobile/2014/08/autodesk-view-data-api-javascript-wrapper.html)

forum:

- <http://forums.autodesk.com/t5/view-and-data-api/bd-p/95>

and GitHub:

- [https://github.com/Developer-Autodesk/AuthTokenServer\\_Simple/blob/master/AuthTokenServer.js](https://github.com/Developer-Autodesk/AuthTokenServer_Simple/blob/master/AuthTokenServer.js)
- [https://github.com/Developer-Autodesk/AuthTokenServer\\_Simple/blob/master/MyAuthToken.js](https://github.com/Developer-Autodesk/AuthTokenServer_Simple/blob/master/MyAuthToken.js)

However much of that material is based on node.js or earlier .NET server client interaction technologies like Windows Forms.

### Solution

The structure of this solution is similar to the structure used by Autodesk in the two GitHub projects above and as further described in the two blog series also above.

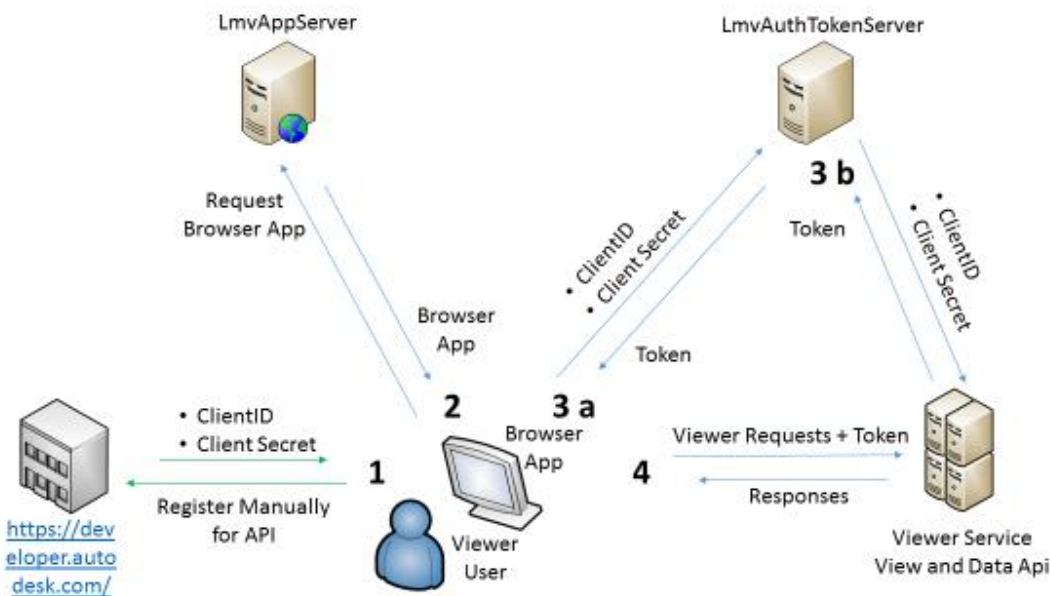
To explore the View and Data API:

- a) the user must be registered to use the API, <https://developer.autodesk.com/>
- b) the development environment must present a model in a browser.

The basic interactions are illustrated in the following diagram.

1. The viewer user is registered to use the API, and receives their ClientID and ClientSecret.
2. The viewer user accesses the browser application that hosts the large model viewer.
3. The browser application requests an authorization token.
4. The browser application instantiates the large model viewer and presents a model.

# Autodesk View and Data API with Visual Studio



This solution assumes: a) the developer has already registered to use the View and Data API and has a client\_id and client\_secret. These are obtained by registering with Autodesk to use the View and Data API. And, b) the developer has a model ready to be viewed, for 'how to' see <http://fast-shelf-9177.herokuapp.com/>.

The solution described here was developed on Windows 7 (and, separately, Windows 10 CTP) using Visual Studio Express 2013 for Web (free edition) Update 3, project target is .NET 4.5 (4.6). The application server and authentication server are hosted on IIS Express (included in 2013 Express). Both servers run ASP.NET Web API 2.2. Additionally, the authentication server has CORS (Cross-origin resource sharing) enabled (5.2.3). The browser uses the AngularJS (1.3.15) framework for user interaction. The Autodesk 3D viewer (1.2.14) interacts via the browser with Autodesk View and Data API (Viewer V1.2.14).

## Development Plan

The browser hosting the LMV uses the AngularJS framework to run a javascript application. That application is served from LmvAppServer the application server hosted on IIS Express (i.e. localhost:52938). To access the model to be presented the LMV requires an access token. The browser requests a token from LmvAuthTokenServer the authentication server (i.e. localhost:63577).

The token request causes LmvAuthTokenServer to request an access token from the Autodesk View and Data API. When the browser receives the response containing the token from LmvAuthTokenServer the browser's cross origin resource sharing policy is triggered. To allow the browser to use the token 'CORS' must be enabled on LmvAuthTokenServer.

LmvAuthTokenServer and LmvAppServer projects are in separate visual studio solutions. Each project is an ASP.NET Web Application that uses an 'Empty' template. LmvAuthTokenServer includes folders and core references for Web API.

# Autodesk View and Data API with Visual Studio

---

An example ASP.NET Web API 2 (C#) project is here:

- <http://www.asp.net/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api>

An example of using CORS is here:

- <http://www.asp.net/web-api/overview/security/enabling-cross-origin-requests-in-web-api>

In this article LmvAuthTokenServer is developed first followed by LmvAppServer. The LmvAppServer server development is sequenced. First basic connectivity to LmvAuthTokenServer is established. Then the AngularJS framework is set up and a small javascript application used to test the 'plumbing'. Finally the javascript functions to host LMV are built.

Install Visual Studio Express 2013 for Web



## PART 2

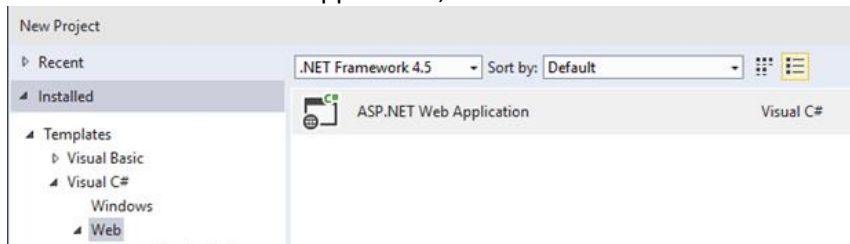
### Authentication Server Setup

Development of the authentication server, LmvAuthTokenServer, is described below.

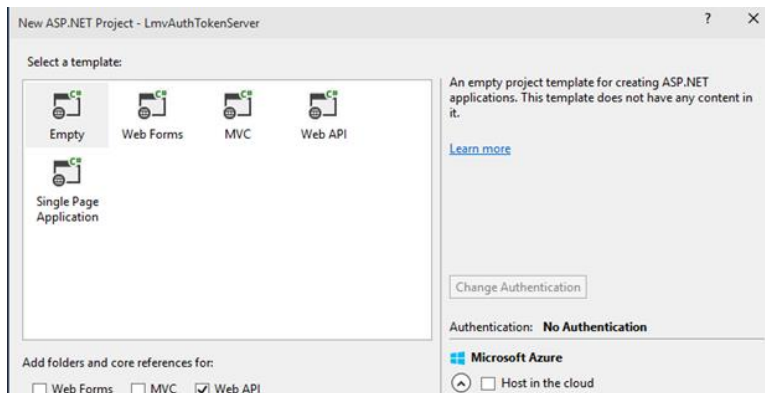
#### Create LmvAuthTokenServer project

Create the LmvAuthTokenServer project In Visual Studio Express 2013 for Web.

1. Start Page or File > New Project > Installed > Templates > Visual C# > Web
2. Select ASP.NET Web Application, Name: LmvAuthTokenServer



3. Select New ASP.NET Project > Select a template > Empty
4. In 'Add folders and core references for:' check Web API

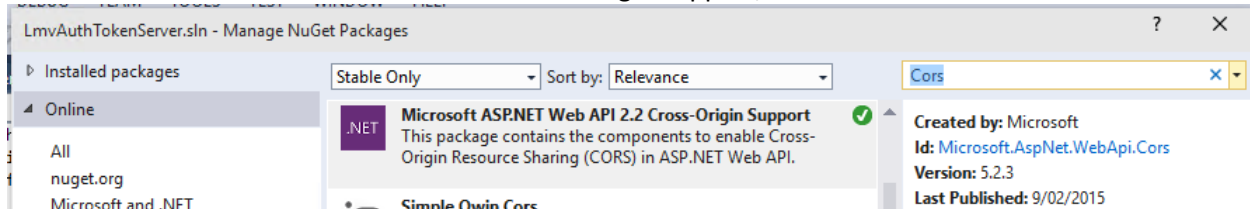


# Autodesk View and Data API with Visual Studio

## Add CORS Package

CORS is not required until the LmvAuthTokenServer build. This CORS package can be downloaded later, I did it now for convenience.

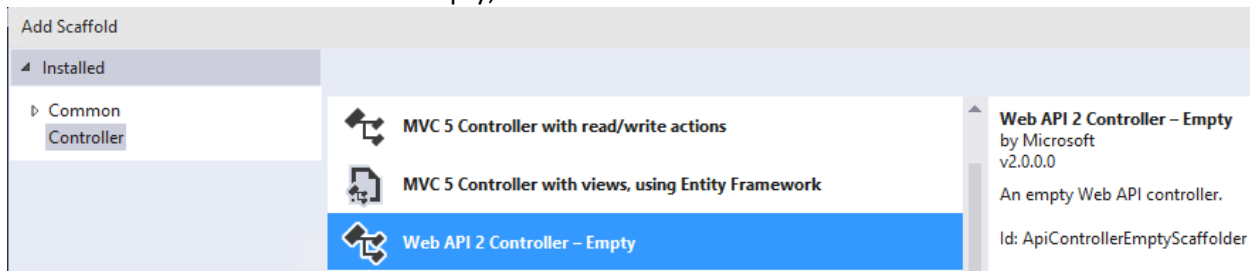
1. Tools > NuGet Package Manager > Manage NuGet Packages for Solution
2. Search Online "CORS"
3. Select Microsoft ASP.NET Web API 2.2 Cross-Origin Support, click Install



Note the dependencies, right hand panel, bottom. You do not need to initiate the downloads for these dependencies. Nuget downloads and installs package (e.g. CORS) dependencies automatically.

## Create AuthTokenController class

1. Right click Solution Explorer > LmvAuthTokenServer > Controllers
2. Select Add > Controller
3. Add Scaffold > Installed > Controller
4. Select Web API 2 Controller – Empty, file Name: AuthTokenController.cs



5. Delete all content in AuthTokenController.cs
6. Add this content to AuthTokenController.cs

```
//using System;
//using System.Collections.Generic;
//using System.Linq;
//using System.Net;
//using System.Net.Http;
using System.Web.Http;

using System.Web.Http.Cors;
using System.Threading.Tasks;
using LmvAuthTokenServer.Models;

namespace LmvAuthTokenServer.Controllers
{
    public class AuthTokenController : ApiController
    {
        // Cache token locally.
        static AuthToken ApiToken = new AuthToken();
    }
}
```

# Autodesk View and Data API with Visual Studio

---

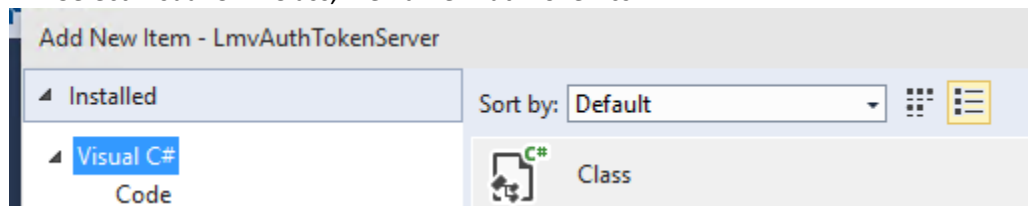
```
public async Task<string> GetAuthApiToken()
{
    await ApiToken.GetApiToken();
    return AuthToken.TheBrokerToken.AccessToken;
}
}
```

The GetAuthApiToken() method can be regarded as the 'entry point' on the LmvAuthTokenServer. The AuthTokenController class uses patterns declared in WebApiConfig.cs (in the App\_Start folder, installed as part of the 'scaffold') to match the uri it exposes e.g. api/AuthToken to the uri a browser requests e.g. <http://localhost:52938/api/authToken> .

The GetAuthApiToken() method is marked as async to let the C# compiler know it contains a statement that may not return a response immediately. The response from GetAuthApiToken() to the requesting browser ( or other type of end user) may be delayed because GetApiToken() initiates a chain of method invocations that result in a token request, across the web, to the Autodesk view and data API. The await key word preceding GetApiToken() identifies to the compiler the function that may not respond immediately. The GetApiToken() statement, in GetAuthApiToken(), invokes the GetApiToken method in class AuthToken which in turn invokes the GetAccessToken method that makes the PostAsync call to the View and Data "authentication/v1/authenticate" uri.

## Create AuthToken class

1. Right click Solution Explorer > LmvAuthTokenServer > Models > Add > New Item
2. Select Visual C# > Class, file name: AuthToken.cs



3. Delete all content in AuthToken.cs
4. Add this content to AuthToken.cs

```
using System;
using System.Collections.Generic;
//using System.Linq;
//using System.Web;
using System.Net.Http;
using System.Threading.Tasks;
using System.Diagnostics;
using Newtonsoft.Json;

namespace LmvAuthTokenServer.Models
{
    public class AuthToken
    {
        private static List<KeyValuePair<string, string>> Credentials
            = new List<KeyValuePair<string, string>>
        {
            { new KeyValuePair<string, string>( "client_id", "<replace with
your_client_id>" )},
        }
```

# Autodesk View and Data API with Visual Studio

---

```
        { new KeyValuePair<string, string>( "client_secret", "<replace with  
your_client_secret>" )},  
        { new KeyValuePair<string, string>( "grant_type", "client_credentials" )}  
    };  
    public static BrokerToken TheBrokerToken { get; set; }  
    private HttpResponseMessage AResponseMessage { get; set; }  
    private DateTime TokenIssuedTime { get; set; }  
    private int AboutExpiredSeconds = 5;  
    // Check if token has been retrieved and store response.  
    // See Autodesk MyAuthToken.js  
    public async Task<BrokerToken> GetApiToken()  
    {  
        // Determine if token has expired.  
        if ((TheBrokerToken == null) ||  
            ((DateTime.Now - TokenIssuedTime).TotalSeconds  
             > (TheBrokerToken.ExpiresIn - AboutExpiredSeconds)))  
        {  
            Debug.WriteLine("AUTH TOKEN: First token or token has expired. Get new  
one ... ");  
            TheBrokerToken = await this.GetAccessToken();  
        }  
        else  
        {  
            Debug.WriteLine("AUTH TOKEN: Has not expired. Use existing token. ");  
        }  
        return TheBrokerToken;  
    }  
    // See Autodesk AuthTokenServer.js  
    // Request token from Autodesk API using credentials  
    private async Task<BrokerToken> GetAccessToken()  
    {  
        string baseUrl = "https://developer.api.autodesk.com/";  
        HttpContent reqData = new FormUrlEncodedContent(Credentials);  
  
        try  
        {  
            using (var client = new HttpClient())  
            {  
                string contentText = null;  
                client.BaseAddress = new Uri(baseUrl);  
                reqData.Headers.ContentType = new  
System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/x-www-form-  
urlencoded");  
  
                HttpResponseMessage response = await  
client.PostAsync("authentication/v1/authenticate", reqData);  
                if (!response.IsSuccessStatusCode) {  
                    Debug.WriteLine("AUTH TOKEN: Call to AD authenticate failed.");  
                }  
                contentText = await response.Content.ReadAsStringAsync();  
                TokenIssuedTime = DateTime.Now;  
                Debug.WriteLine(contentText, "AUTH TOKEN: Content text is ");  
                Debug.WriteLine(TokenIssuedTime.GetDateTimeFormats('T')[0], "AUTH  
TOKEN: Issued time is ");  
                return JsonConvert.DeserializeObject<BrokerToken>(contentText);  
            }  
        }  
        catch (Exception exception)
```

# Autodesk View and Data API with Visual Studio

---

```
        {
            System.Diagnostics.Debug.WriteLine("AUTH TOKEN: Caught authenticate call
to AD API.");
            System.Diagnostics.Debug.WriteLine(exception);
            throw;
        }
    }
}
// Helps to deserialize response
public class BrokerToken
{
    [JsonProperty(PropertyName = "token_type")]
    public string TokenType { get; set; }
    [JsonProperty(PropertyName = "expires_in")]
    public int ExpiresIn { get; set; }
    [JsonProperty(PropertyName = "access_token")]
    public string AccessToken { get; set; }
}
}
```

The AuthToken class does three things 1) holds the user credentials client\_id, client\_secret. Note the credentials are obtained external to this solution (see Soluton above), 2) makes the request for a token, across the web, to the Autodesk view and data API, and 3) caches the request response.

The AuthToken and AuthTokenController classes allocate static storage for ApiToken and BearerToken instances respectively. That enables the token in the request response from View and Data API to be cached so requests to the View and Data API are made only when AccessToken in TheBrokerToken has expired.

The BrokerToken class is in the same namespace, LmvAuthTokenServer.Models, as the AuthToken class. The BrokerToken class declares the type for storing token properties and provides the type for the JsonConvert.Deserialize<T>Object() method.

## PART 3

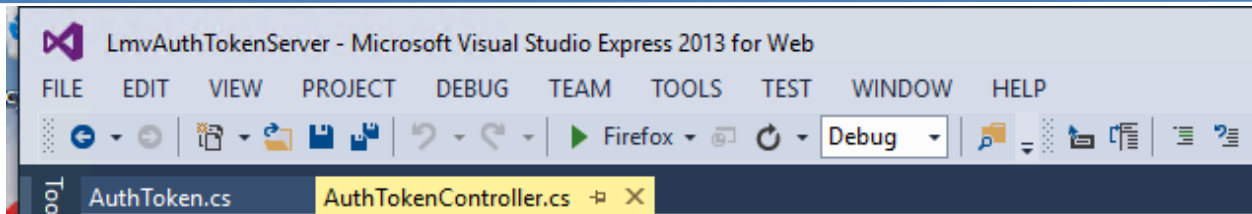
### Test Authentication Server

The LmvAuthTokenServer is tested by requesting a token from the View and Data API. The expected response is an access token. When the LmvAuthTokenServer project is started, IIS Express also starts and becomes 'Localhost' for the project. A browser on the local machine sends a request to the api/authtoken uri exposed by LmvAuthTokenServer, the server passes the request to the View and Data API which responds with a token, passed on by the server, and presented in the browser. There is no CORS.

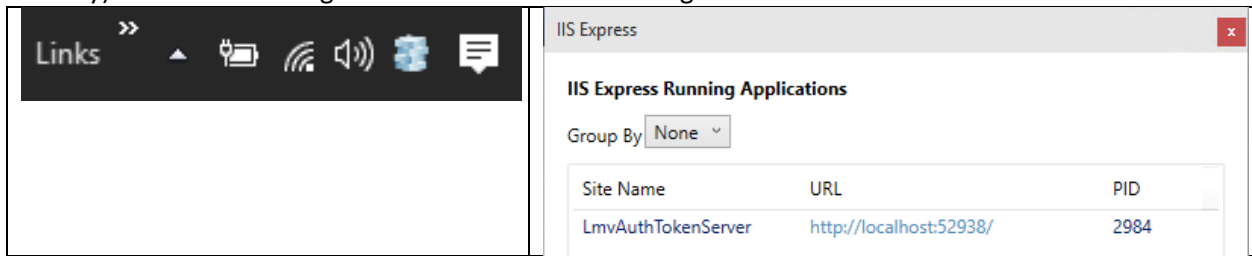
### Setup LmvAuthTokenServer test

1. Build LmvAuthTokenServer project, VS Express > F12 or Debug > Build.
2. Select browser, here I use Firefox on the server side (no particular reason).

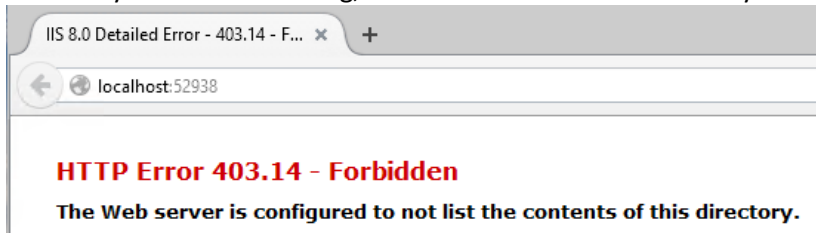
# Autodesk View and Data API with Visual Studio



3. Start LmvAuthTokenServer project in Visual Studio Express, either click the green triangle, or F5, or Debug > Debug. IIS Express starts up and the browser appears.
4. Verify IIS Express is running. Confirm the presence of the its icon in the notification area (system tray) – the blue rectangle below. Form more detail right click the icon.

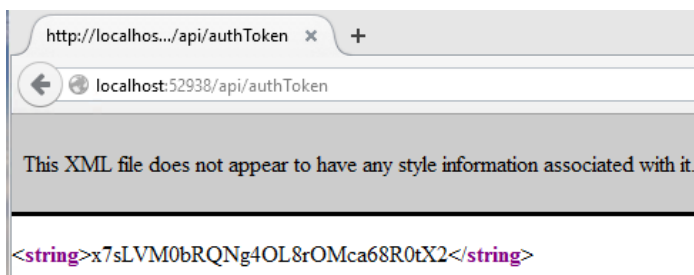


5. Verify browser is running, before the test the browser may look something like this:



## Perform LmvAuthTokenServer test

1. Change the url in the browser to <http://localhost:52938/api/authToken> . Note use your unique port number i.e. not '52938'.



2. Verify the token is visible in the browser, a string something like the above should be present.

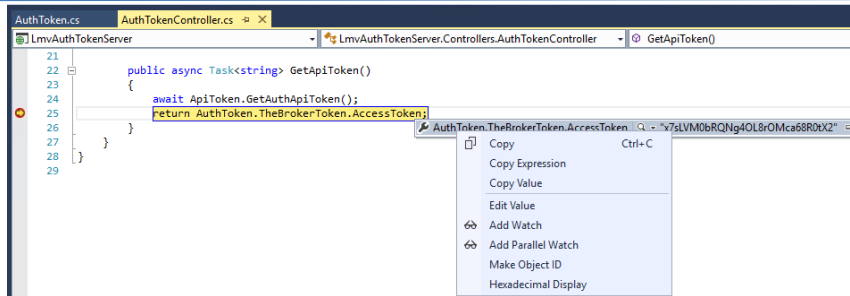
## Debug LmvAuthTokenServer test

If the expected response is not received, debug:

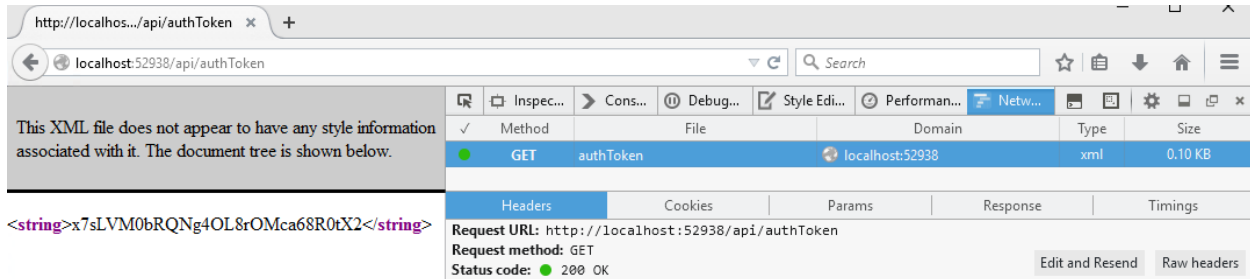
- VS Express Debug



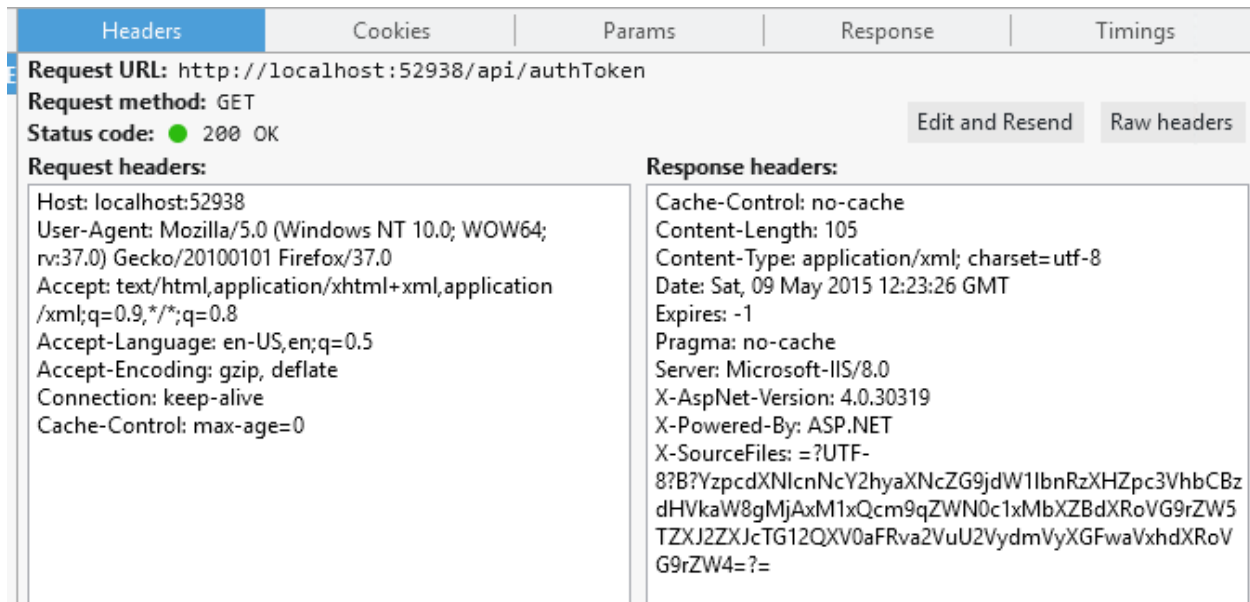
# Autodesk View and Data API with Visual Studio



- **Browser Debug** – to look between the browser and LmvAuthTokenServer. F12 on the browser.



Example: Call header information between the browser and LmvAuthTokenServer, using Firefox.



- **Fiddler** (or equivalent) – Install Fiddler to look between LmvAuthTokenServer and the View and Data API.

# Autodesk View and Data API with Visual Studio

## PART 4

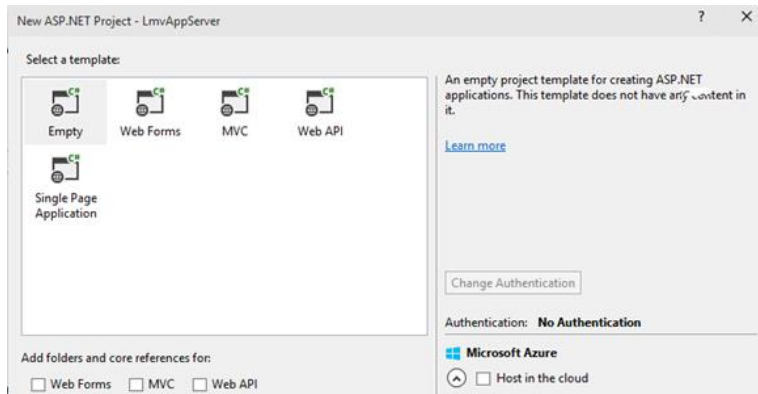
### Application Server Setup

Development of the application server, LmvAppServer, is sequenced. First basic connectivity to LmvAuthTokenServer is established. Then the AngularJS framework is set up and a small javascript application used to test the 'plumbing'. Finally the javascript functions necessary to host LMV are added.

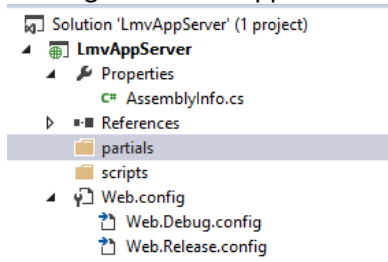
### Create LmvAppServer project

Create the LmvAppServer project In Visual Studio Express 2013 for Web.

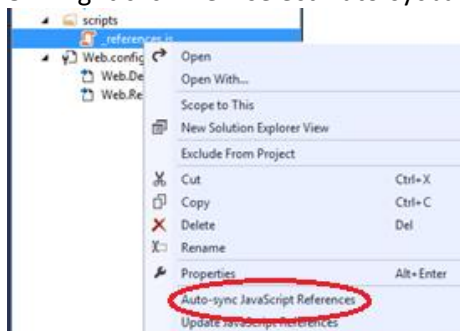
1. Start Page or File > New Project > Installed > Templates > Visual C# > Web
2. Select ASP.NET Web Application, Name: LmvAppServer
3. Select New ASP.NET Project > Select a template > Empty
4. In 'Add folders and core references for:' do not check Web API



5. Right click LmvAppServer > Add > New Folder for 'partials'
6. Right click LmvAppServer > Add > New Folder for 'scripts'



7. Right click folder scripts > Add > JavaScript File, filename \_references.js to folder scripts. This file is used by IntelliSense but not included in an Empty template.
8. Right click file > select Auto-sync JavaScript References



# Autodesk View and Data API with Visual Studio

---

## Create get-access-token.js

1. Right click folder scripts > Add > JavaScript File, file Name: get-access-token.js
2. Add this content to get-access-token.js

```
function getAccessToken() {  
    var theUrl = 'http://localhost:52938/api/authToken';  
    var xmlHttp = null;  
    xmlHttp = new XMLHttpRequest();  
    xmlHttp.open("GET", theUrl, false);  
    xmlHttp.send(null);  
    if (xmlHttp.status === 200 || xmlHttp.status === 304) {  
        var rt = xmlHttp.responseText;  
        if (rt == null) {  
            alert("xmlHttp.responseText is null")  
        } else {  
            return JSON.parse(rt);  
        }  
    }  
}
```

Function getAccessToken() makes a synchronous call to the value of theUrl. Modify theUrl value to use the port number (e.g. modify 52938) used by your LmvAuthTokenServer localhost instance.

## Test LmvAppServer request to LmvAuthTokenServer service

Verify a token request, initiated from a browser on the LmvAppServer server, to the api/authToken uri on LmvAuthTokenServer returns an access token.

## Setup service test

The getAccessToken() function is called from a browser using a javascript function in the Index.html file.

1. Right click LmvAppServer (project) > Add > HTML Page, file Name: Index.html
2. Delete all content in Index.html
3. Add this content to Index.html

```
<!DOCTYPE html>  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head>  
    <title>LmvAppSvr</title>  
</head>  
<body>  
    <div>  
        Test API call to LmvAuthTokenServer  
        <br/>  
    </div>  
    <script src="scripts/get-access-token.js"></script>  
    <script type="text/javascript">  
        document.write((getAccessToken()));  
    </script>  
</body>  
</html>
```

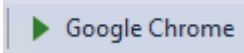
The getAccessToken() function in the document.write statement returns the response to the xmlHttp.send request sent to LmvAuthTokenServer api/authToken uri. The document.write statement displays that response in the browser.

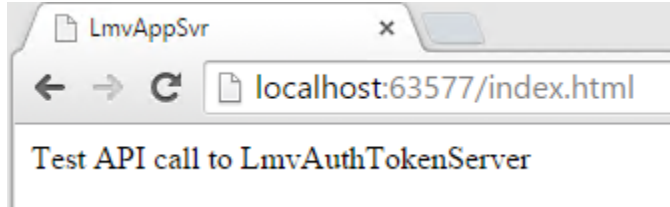
# Autodesk View and Data API with Visual Studio

## Setup service test


1. Build LmvAppServer project, VS Express > F5 or Debug > Build.
2. Select browser, here I use Chrome on the 'client' side (no particular reason).

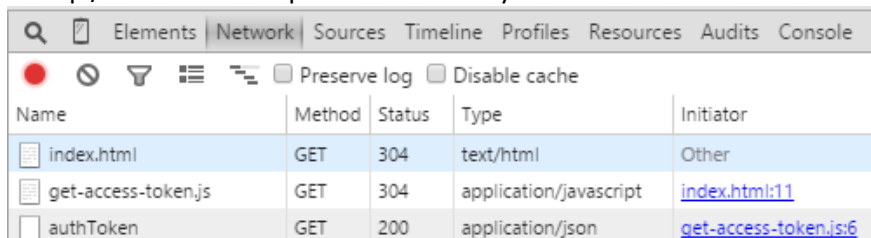
## Perform service test

3. Start LmvAppServer project in Visual Studio Express, either click  the green triangle, or F5, or Debug > Debug. IIS Express starts up and the browser appears.
4. Verify IIS Express is running. Confirm as described above in 'Setup LmvAuthTokenServer test'.
5. View browser, there is no token string displayed, so the test outcome is unexpected:

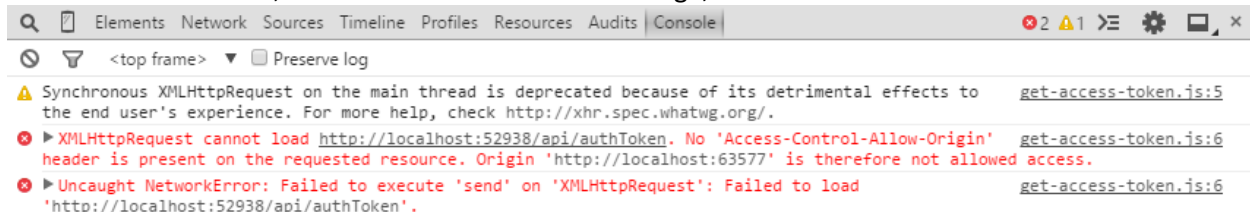


## Debug service test

1. Open Chrome developer tools (F12)
2. Select Network tab
3. Reload browser page 
4. Confirm, Index.html, get-token.js, and authToken are present in the 'Name' column of the Network tab. Note: Status 304 (Not Modified) since last loaded, 200 (OK, i.e. successful). So the call to api/authToken completed successfully.



5. Select Console tab, however there is an error message,



The 'No Access-Control-Allow-Origin header present' console message indicates:

- a) Browser made a cross origin request to share (access) a resource.
- b) Response returned from LmvAuthTokenServer did not contain an Access-Control-Allow-Origin header, and
- c) Browser's CORS policy prevented the browser from emitting the response value to getAccessToken(). Note the token was transmitted to the browser, hence don't rely on CORS for security.

# Autodesk View and Data API with Visual Studio

authToken	GET	200
3 requests 1.1 KB transferred Finish: 349 ms		
× Headers Preview Response Timing		
1	"S7p6wVymMd41QRsmfC3HF1UJ0851"	

The request was 'cross origin' because the browser (see Request Header) served from LmvAppServer at Origin <http://localhost:63577/> invokes api/authToken service on LmvAuthTokenServer at Host <http://localhost:52938/>. The Origin and Host are different because the urls are different. In this case the url difference is determined by the port identifier. Note: Internet Explorer does not consider the port when comparing origins see <http://www.asp.net/web-api/overview/security/enabling-cross-origin-requests-in-web-api>

Note there is no Access-Control-Allow-Origin header and value in the 'Response Headers' for the failed request. For the headers associated with a successful request see below.

×	Headers	Preview	Response	Timing
▼ General				
Request URL: http://localhost:52938/api/authToken				
Request Method: GET				
Status Code: 200 OK				
▼ Response Headers view parsed				
HTTP/1.1 200 OK				
Cache-Control: no-cache				
Pragma: no-cache				
Content-Type: application/json; charset=utf-8				
Expires: -1				
Server: Microsoft-IIS/8.0				
X-AspNet-Version: 4.0.30319				
X-SourceFiles: =?UTF-8?B?YzpcdXN1cnNcY2hyaXNcZG				
XV0aFRva2VuU2VydMvyXGfwaVxhdXR0VG9rZW4=?=				
X-Powered-By: ASP.NET				
Date: Thu, 14 May 2015 07:53:56 GMT				
Content-Length: 30				
▼ Request Headers view parsed				
GET /api/authToken HTTP/1.1				
Host: localhost:52938				
Connection: keep-alive				
Cache-Control: max-age=0				
Origin: http://localhost:63577				
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)				
Accept: */*				
Referer: http://localhost:63577/index.html				
Accept-Encoding: gzip, deflate, sdch				
Accept-Language: en-US,en;q=0.8				

A description of how CORS works that I found helpful is here:

- <https://msdn.microsoft.com/en-us/magazine/dn532203.aspx> CORS Support in ASP.NET Web API 2.

# Autodesk View and Data API with Visual Studio

## Part 5

### Enable CORS

CORS is enabled on LmvAuthTokenServer, the server responsible for handing the resource (Token) to a browser with a different Origin.

1. Stop LmvAuthTokenServer.
2. Open LmvAuthTokenServer > App\_Start > WebApiConfig.cs file.
3. Add `config.EnableCors()` statement to Register method in WebApiConfig.cs, and save.

```
11 public static void Register(HttpConfiguration config)
12 {
13     // Enable CORS
14     config.EnableCors();
15
16     // Web API configuration and services
```

4. Open LmvAuthTokenServer > Controllers > AuthTokenController.cs file.
5. Add attribute `[EnableCors(origins: "http://localhost:63577", headers: "*", methods: "*")]`

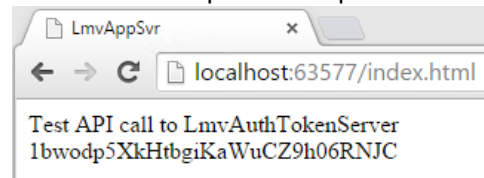
```
13 namespace LmvAuthTokenServer.Controllers
14 {
15     [EnableCors(origins: "http://localhost:63577", headers: "*", methods: "*")]
16
17     public class AuthTokenController : ApiController
18     {
19         // Cache token locally.
20         static AuthToken ApiToken = new AuthToken();
21
22         public async Task<string> GetAuthApiToken()
23         {
24             await ApiToken.GetApiToken();
25             return AuthToken.TheBrokerToken.AccessToken;
26         }
27     }
28 }
```

6. Modify the value of the 'origins' property to use the port number of your localhost.
7. Save file AuthTokenController.cs
8. Build LmvAuthTokenServer project
9. Restart project LmvAuthTokenServer

### Re-run service test

CORS is now enabled on LmvAuthTokenServer




1. Reload page in Chrome with url <http://localhost:63577/Index.html>
2. Browser output is as expected – token string is displayed




3. The token is different to the token above because LmvAuthTokenServer was restarted following the recompile after adding the EnableCors statement.





Chrome > More Tools > Developer Tools > Network Tab

# Autodesk View and Data API with Visual Studio

Name	Method	Status	Type	Initiator
 Index.html	GET	304	text/html	Other
 get-access-token.js	GET	304	application/javascript	<a href="#">Index.html:11</a>
 authToken	GET	200	application/json	<a href="#">get-access-token.js:6</a>

Note the Access-Control-Allow-Origin header and value in the 'Response Headers' for the successful request. For the headers associated with an unsuccessful request see above.

 authToken	GET	200	application/json	<a href="#">get-access-token.js:6</a>	510 B
---	-----	-----	------------------	---------------------------------------	-------

3 requests  1.1 KB transferred  Finish: 324 ms  DOMContentLoaded: 325 ms  Load: 326 ms

×

Headers

Preview

Response


Timing

▼

General

Request URL: http://localhost:52938/api/authToken

Request Method: GET

Status Code:  200 OK

▼

Response Headers

view parsed

HTTP/1.1 200 OK

Cache-Control: no-cache

Pragma: no-cache

Content-Type: application/json; charset=utf-8

Expires: -1

Server: Microsoft-IIS/8.0

Access-Control-Allow-Origin: http://localhost:63577

X-AspNet-Version: 4.0.30319

X-SourceFiles: =?UTF-8?B?YzpcdXN1cnNcY2hyaXNcZG9jdW11bnRzXHZcTG12QXV0aFRva2VuU2VydMvyXGFwaVxhdXRoVG9rZW4=?=

X-Powered-By: ASP.NET

Date: Thu, 14 May 2015 07:43:57 GMT

Content-Length: 30

▼

Request Headers

view parsed

GET /api/authToken HTTP/1.1

Host: localhost:52938

Connection: keep-alive

Cache-Control: max-age=0

Origin: http://localhost:63577

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36

Accept: \*/\*

Referer: http://localhost:63577/Index.html

Accept-Encoding: gzip, deflate, sdch

Accept-Language: en-US,en;q=0.8

×

Headers

Preview

Response

Timing

1	"1bwodp5XkHtbgiKaWu CZ9h06RNJC"
---	---------------------------------

# Autodesk View and Data API with Visual Studio

---

## Javascript Application Overview

This solution is a javascript application that runs in the browser and uses the AngularJS framework. The application, served from LmvAppServer;

- a) Retrieves an authentication token from the Autodesk View and Data service,
- b) Causes the large model viewer to be populated with a 3D model, previously uploaded to the View and Data service.
- c) Loads the large model viewer from the view and service into the browser.
- d) Allows the user to manipulate the model through the browser.

AngularJS has many features, these are made available through a collection of modules, visible here:

<https://ajax.googleapis.com/ajax/libs/AngularJS/1.3.15/MANIFEST>

This solution references only those modules necessary, angular.js and angular-route.min.js, for the features used. AngularJS modules are referenced in the LmvAppServer project, index.html file. Note the solution does not download a copy of these files to the Visual Studio project folders, but you can.

The application steps are sequenced:

1. Create the javascript application
2. Verify plumbing is working
3. Add the functions specific to the large model viewer
4. Verify large model viewer is working

## Step 1 Create the javascript application

This application is composed of these files. Index.html, lmviewer.html, get-auth-token.js, get-access-token.js, app.js. This file, viewer-embed.js, is added in step 3 to provide the large model viewer (LMV) feature.

### Create Index.html

1. Rename Index.html on LmvAppServer to Index-test.html.
2. Create another Index.html. Project LmvAppServer > Add > HTML Page.
3. Delete all content in Index.html
4. Add this content to Index.html

```
<!DOCTYPE html>
<html ng-app="LmvAppSvr">
<head>
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>LmvApp</title>

  <!-- Angular CDN -->
  <script
src="https://ajax.googleapis.com/ajax/libs/AngularJS/1.3.15/angular.js"></script>
  <script src="https://ajax.googleapis.com/ajax/libs/AngularJS/1.3.15/angular-
route.min.js "></script>

  <!-- Autodesk -->
  <!--<link rel="stylesheet"
href="https://developer.api.autodesk.com/viewingservice/v1/viewers/style.css"
type="text/css">
```



# Autodesk View and Data API with Visual Studio

---

```
<script
src="https://developer.api.autodesk.com/viewingservice/v1/viewers/viewer3D.js"></script>-
->

<!--LmvAppSvr Application -->
<script src="scripts/get-access-token.js"></script>
<script src="scripts/get-auth-token.js"></script>
<!--<script src="scripts/viewer-embed.js"></script>-->
<script src="scripts/app.js"></script>
</head>
<body>
  <div>
    <div data-ng-view></div>
  </div>
</body>
</html>
```

## Create lmvviewer.html

1. Right click LmvAppServer > partials
2. Select Add > HTML Page, file name: lmvviewer.html
3. Delete all content in lmvviewer.html
4. Add this content to lmvviewer.html

```
<div>
  <p>Large Model Viewer</p>
  <p><{{checkPlumbing}}</p>
  <!--<p>Initial Access Token:  {{authToken}}</p>-->
  <br />
</div>
```

## Create get-auth-token.js

1. Right click LmvAppServer > scripts
2. Select Add > JavaScript File, file name: get-auth-token.js
3. Add this content to get-auth-token.js

```
var getAuthToken = angular.module("getAuthToken", []);

// Sync call to LmvAuthTokenServer
getAuthToken.factory('authTokenFactory', function () {

  var getToken = function checkPlumbing() {
    return "The plumbing is running";
  }
  return getToken;
});
```

## Create app.js

1. Right click LmvAppServer > scripts
2. Select Add > JavaScript File, file name: app.js
3. Add this content to app.js

```
var app = angular.module("LmvAppSvr", ["ngRoute", "getAuthToken"])

app.config(function ($routeProvider) {
  $routeProvider
```


# Autodesk View and Data API with Visual Studio

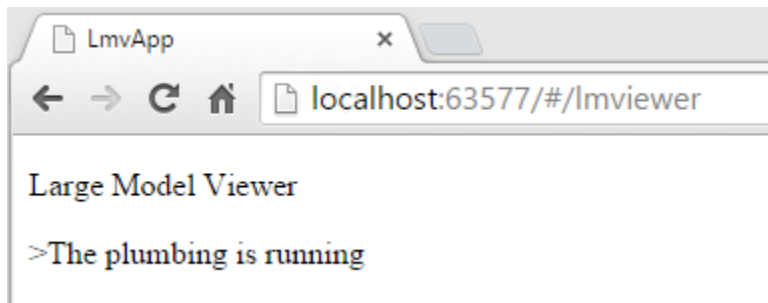
---

```
.when ('/lmviewer', {
  controller: 'lmvCtrl',
  templateUrl: 'partials/lmviewer.html'
})
.otherwise({redirectTo: '/lmviewer'})
});

app.controller('lmvCtrl', ['$scope', 'authTokenFactory',
  function ($scope, authTokenFactory) {
    $scope.checkPlumbing = authTokenFactory();
    console.log("checkPlumbing . . .", $scope.checkPlumbing);
  }
]);
```

## Step 2 verify plumbing is working

1. Start LmvAppServer, click  . Browser opens.



## Part 6

### Step 3 Add large model viewer specific content

#### Create Index.html

1. Delete Index.html, or rename it if you want to keep the test code around.
2. Create new Project LmvAppServer > Add > HTML Page, file name: Index.html
3. Delete all content in Index.html
4. Add this content to Index.html

```
<!DOCTYPE html>
<html data-ng-app="LmvWebApi">
<head>
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>LmvApp</title>

  <!-- Angular CDN -->
  <script
src="https://ajax.googleapis.com/ajax/libs/angularJS/1.3.15/angular.js"></script>
  <script src="https://ajax.googleapis.com/ajax/libs/angularJS/1.3.15/angular-
route.min.js "></script>

  <!-- Autodesk -->
```

# Autodesk View and Data API with Visual Studio

```
<link rel="stylesheet"
href="https://developer.api.autodesk.com/viewingservice/v1/viewers/style.css"
type="text/css">
<script
src="https://developer.api.autodesk.com/viewingservice/v1/viewers/viewer3D.js"></script>

<!--LmvApiSvr Application -->
<script src="scripts/get-access-token.js"></script>
<script src="scripts/get-auth-token.js"></script>
<script src="scripts/viewer-embed.js"></script>
<script src="scripts/app.js"></script>

</head>
<body>
  <div>
    <div data-ng-view></div>
  </div>
</body>
</html>
```

The LmvAppSvr application uses the AngularJS factory feature. A factory is used to inject the `getAccessToken()` function, in file `get-access-token.js`, into the 'LmvCtrl' controller of the angular.module 'LmvAppSvr', in file `app.js`. The factory function 'authTokenFactory' is in file `gt-auth-token.js`

## Modify Lmviewer.html

1. Double click to open LmvAppServer > partials > Lmviewer.html
2. Replace content with:

```
<div>
  <p>Large Model Viewer</p>
  <!--<p>{{checkPlumbing}}</p>-->
  <p>Initial Access Token:  {{authToken}}</p>
  <br />
</div>
<div>
  <input id="ButtonView" type="button" value="View" onclick="initialize()" /><br />
</div>
<div id="viewer" style="position:absolute; width:97%; height:40%;"></div>
```

## Modify get-auth-token.js

1. Double click to open LmvAppServer > scripts > get-auth-token.js
2. Replace content with:

```
var getAuthToken = angular.module("getAuthToken", []);
// Sync call to LmvAuthTokenServer
getAuthToken.factory('authTokenFactory', function () {
  var getAToken = getAccessToken;
  return getAToken;
});
```

Note the `getAccessToken` function defined `get-access-token.js` calls in `api/authToken` service in `LmvAuthTokenServer` using `XMLHttpRequest()`. This is a synchronous call from the main UI thread and cause javascript to throw the warning in the browser's console:

⚠ Synchronous XMLHttpRequest on the main thread is deprecated because of its detrimental effects to the end user's experience. For more help, check <http://xhr.spec.whatwg.org/>. [get-access-token.js:5](#)

# Autodesk View and Data API with Visual Studio

---

Changing to an async call, for example using Angularjs \$http or \$q, does not work with the viewer code. \$http, and \$q return a promise (for more details see angular documentation, and the 'q' literature in general) which allows 'mainline' code to continue while awaiting the async response.

In general async is a good thing for UI threads as the UI remains responsive. However the model viewer doesn't know what to do with the 'promise' and the viewer's subsequent call using the token fails because the value of token, whatever it is, is not a token value.

## Add viewer-embed.js

1. Right click LmvAppServer > scripts
2. Select Add > JavaScript File, file name: viewer-embed.js
3. Add content to viewer-embed.js from 'viewer-embed.js' file box at:  
<http://the360view.typepad.com/blog/2015/03/lab4-view-and-data-api-web-intro-js.html>

## Modify app.js

1. Double click LmvAppServer > scripts > app.js
2. Comment out these two statements

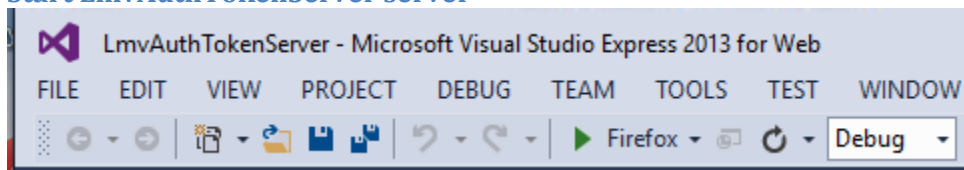
```
//$scope.checkPlumbing = authTokenFactory();  
//console.log("checkPlumbing . . .", $scope.checkPlumbing);
```

3. Add this content to app.js after the console.log statement. Make sure you use your encoded document urn. Note the 'urn:' prefix is required and in addition to the urn string included in the encoded string.

```
initialize();  
function initialize() {  
    // Change docUrn value to your Encoded URN see Step 10 from http://fast-shelf-  
9177.herokuapp.com/  
    var docUrn = '<replace with your document URN generated with same client  
credentials as above>';  
    var viewerElement = document.getElementById('viewer');  
    ViewerEmbed.initialize(getToken, docUrn, viewerElement);  
}  
function getToken() {  
    return authTokenFactory();  
}
```

## Step 4 verify large model viewer is working

### Start LmvAuthTokenServer server

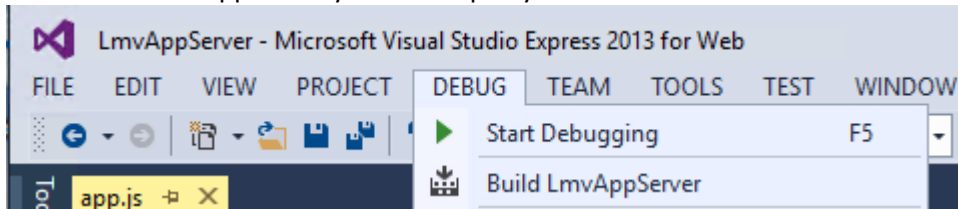


Click green triangle 'Firefox'.

# Autodesk View and Data API with Visual Studio

## Build LmvAppServer

Generally it is not necessary to build the project after small changes because the app is all javascript and javascript is compiled at run time. However given the magnitude of the changes I build to give the environment an opportunity to clean up any unwanted artifacts.

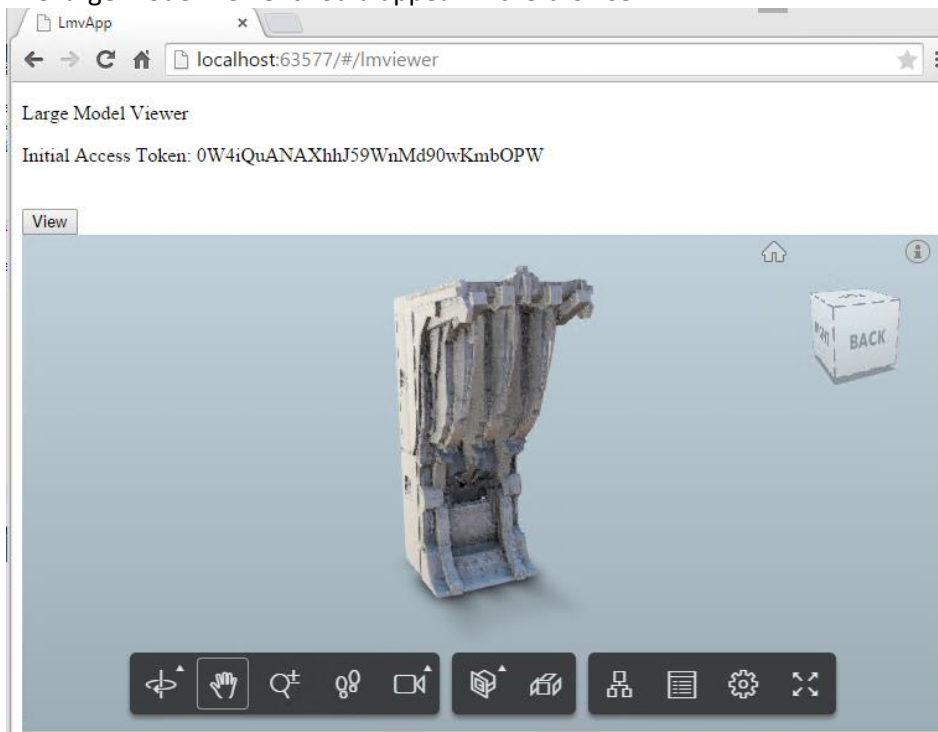


## Start LmvAppServer



Click green triangle 'Google Chrome'.

The large model viewer should appear in the browser.



## If not

### • Verify model is viewable

1. Go through the steps at <http://fast-shelf-9177.herokuapp.com/> to manually verify the bucket contains a viewable model. Note the bucket can be present but the model is not.
2. If so upload the model again and verify it is viewable.

### • Verify the CORS configuration.

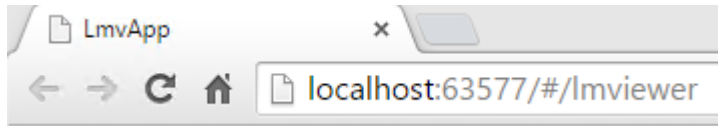
1. Go to LmvAuthTokenServer > AuthTokenController.cs file and note the value of the origins: attribute of the EnableCors statement e.g.  

```
[EnableCors(origins: "http://localhost:63577", headers: "*", methods: "*")]
```

# Autodesk View and Data API with Visual Studio

---

2. Compare that value with the with Chrome browser URL. For non IE browsers the two values need to be same. See above for IE comment. Otherwise, the browser's CORS policy executes, no token is passed to app.js and the large model viewer fails to initialize. An error message will also be present in the browser's dev tools console.



- **Otherwise**
  1. Open the browser's developer tools (F12 in Chrome)
  2. Check the console log
  3. Check Network activity
  4. Set break points e.g. in app.js via the browser's development tools source tab. Note Index.html references the non minimized download for viewer3D.js which means its source is readable. Search for getaccesstoken you can find where the gettoken value is passed into the viewer code if necessary.