Table of Contents

FME Server REST API 1.1
About This Document 1.2
Course Overview 1.2.1
Course Resources 1.2.2
Chapter 1. Overview of the REST API 1.3
  1.1 What is a REST API? 1.3.1
  1.2 What components are in a Request? 1.3.2
  1.3 Authorization And Authentication 1.3.3
  1.4 Tokens and the REST API 1.3.4
  1.5 Exercise 1- Create a User 1.3.5
Chapter 2. Getting Started with the REST API 1.4
  2.1 Exploring the REST API Homepage 1.4.1
  2.2 REST API Response 1.4.2
Chapter 3. Using an HTTP Caller to Make Calls to the Server 1.5
  3.1 Exercise 2- Creating a call with a REST Client Tool 1.5.1
  3.2 Exercise 3- Authorization and the REST API 1.5.2
Chapter 4. Interacting with Workspaces 1.6
  4.1 Exercise 4- Running a Synchronous Job With Standard Parameter 1.6.1
  4.2 Exercise 5- Running a Job Asynchronously 1.6.2
  4.3 Exercise 6- Retrieve Job Results 1.6.3
  4.4 Exercise 7- Running a Job Synchronously and Upload Data: Transact Data 1.6.4
  4.5 Using REST API Commands in a Workbench 1.6.5
  4.6 Exercise 8- Chaining Jobs Together using the REST API in FME Workbench 1.6.6
Chapter 5. Web Services and the REST API 1.7
  5.1 Exercise 9- Introduction to Using Direct URLs 1.7.1
Chapter 6. Server Admin Tasks 1.8
  6.1 Exercise 10- Working with Resources 1.8.1
  6.2 Exercise 11- Using a Shared Resource as a Workspace Output Location 1.8.2
Chapter 7. Job Management 1.9
  7.1 Exercise 12- Creating a Job Queue 1.9.1
  7.2 Retrieve Job History 1.9.2
  7.3 Exercise 13- Creating a Workspace to Retrieve Job History 1.9.3
Chapter 8. Developing with the FME Server REST API 1.10
  8.1 Best Practices 1.10.1
  8.2 Hosting Web Applications 1.10.2
  8.3 Exercise 14- Setting up Your Web Application 1.10.3
Chapter 9. Building Custom Applications 1.11
  9.1 Introduction to Creating the Dynamically Populated Form Web Page 1.11.1
  9.2 Exercise 15- Creating the Dynamically Populated Form Web Page 1.11.2
  9.3 Introduction to Data Distribution 1.11.3
  9.4 Exercise 16- Data Distribution 1.11.4
  9.5 Introduction to Data Visualization 1.11.5
  9.6 Exercise 17- Data Visualization 1.11.6
  9.7 Introduction to Data Uploads 1.11.7
  9.8 Exercise 18- Data Uploads 1.11.8
  9.9 Exercise 19- Data Validation Application 1.11.9
Chapter 10. Course Wrap Up 1.12
  10.1 Product Information and Resources 1.12.1
10.2 Community Information and Resources 1.12.2
10.3 Feedback and Certificates 1.12.3
10.4 Troubleshooting 1.12.4
10.5 Thank You! 1.12.5
FME Server REST API Training 2018

This is the manual for the training course for Safe Software’s FME Server REST API.

SAFE SOFTWARE®

The FME Server REST API Training is a training course that teaches FME users how to master the FME Server REST API and build their own web applications to solve data challenges. The REST API allows a user to interact with the FME Server without the use of the FME Server Graphical User Interface. This enables the user to create web applications that allow a client access to the FME Server without ever having to log in or directly access the server. This course teaches users what the REST API is, how to use it, and how to build custom web applications that leverage the power of FME.

Course Structure

The full course is made up of two sections. These sections are:

- **Interacting with the FME Server REST API**
  - Getting Started with the REST API
  - Using a REST Client Tool to Make Calls to the FME Server
  - Interacting with Workspaces
  - Web Services, Direct URLs, and the REST API
  - Server Administration Tasks
  - Job Management

- **Developing with the FME Server REST API**
  - Best Practices when Building a Web Application
  - Creating a Dynamically Populated Form
  - Data Distribution Using ArcGIS
  - Data Visualization Using ArcGIS
  - Data Uploads and Downloads
  - Creating a Data Validation Application
  - Course Wrap Up

Current Status

The current status of this manual is: **Complete**: this manual can be used for training, subject to minor, last-minute fixes.

This manual applies to **FME2018.0**

The status of each chapter is:

- Chapter 0: Complete content. No exercises
- Chapter 1: Complete content and exercises
- Chapter 2: Complete content. No exercises
- Chapter 3: Complete content and exercises
- Chapter 4: Complete content and exercises
- Chapter 5: Complete content and exercises
- Chapter 6: Complete content and exercises
- Chapter 7: Complete content and exercises
- Chapter 8: Complete content and exercises
- Chapter 9: Complete content and exercises
- Chapter 10: Complete content no exercises
- Slides: Complete
- FMEData: Complete
• Course Outline: Complete

**NB:** Even for completed content, Safe Software Inc. assumes no responsibility for any errors in this document or their consequences, and reserves the right to make improvements and changes to this document without notice. See the full licensing agreement for further details.
Course Prerequisites

- Basic working knowledge of FME Desktop applications
- Completion of FME Server Authoring Training

Requirements

- FME Desktop– for publishing
- FME Server
- Access to a REST Client
  - Postman is used in course

Licensing and Warranty

Permission is hereby granted to use, modify and distribute the FME Tutorials and related data and documentation (collectively, the “Tutorials”), subject to the following restrictions:

1. The origin of the Tutorials and any associated FME® software must not be misrepresented.
2. Redistributions in original or modified form must include Safe Software’s copyright notice and any applicable Data Source(s) notices.
3. You may not suggest that any modified version of the Tutorials is endorsed or approved by Safe Software Inc.
4. Redistributions in original or modified form must include a disclaimer similar to that below which: (a) states that the Tutorials are provided “as-is”; (b) disclaims any warranties; and (c) waives any liability claims.

Safe Software Inc. makes no warranty either expressed or implied, including, but not limited to, any implied warranties of merchantability, non-infringement, or fitness for a particular purpose regarding these Tutorials, and makes such Tutorials available solely on an “as-is” basis. In no event shall Safe Software Inc. be liable to anyone for direct, indirect, special, collateral, incidental, or consequential damages in connection with or arising out of the use, modification or distribution of these Tutorials.

This manual describes the functionality and use of the software at the time of publication. The software described herein, and the descriptions themselves, are subject to change without notice.

Data Sources

City of Vancouver

Unless otherwise stated, the data used here originates from open data made available by the City of Vancouver, British Columbia. It contains information licensed under the Open Government License - Vancouver.

Others

Forward Sortation Areas: Statistics Canada, 2011 Census Digital Boundary Files, 2013. Reproduced and distributed on an "as is" basis with the permission of Statistics Canada. © This data includes information copied with permission from Canada Post Corporation.

Digital Elevation Model: GeoBase®

Fire Hall Data: Some attribute data adapted from content © 2013 by Wikipedia, used under a Creative Commons Attribution-ShareAlike license

Stanley Park GPS Trail: Used with kind permission of VancouverTrails.com.

OpenStreetMap Datasets: © OpenStreetMap contributors.

Contains information licensed under the Open Government Licence – British Columbia

Copyright

© 2005–2018 Safe Software Inc. All rights are reserved.
Revisions

Every effort has been made to ensure the accuracy of this document. Safe Software Inc. regrets any errors and omissions that may occur and would appreciate being informed of any errors found. Safe Software Inc. will correct any such errors and omissions in a subsequent version, as feasible. Please contact us at:

Safe Software Inc.
Phone: 604-501-9985
Email: train@safe.com
Web: safe.com
GitHub: github.com/safesoftware/FMETraining

Safe Software Inc. assumes no responsibility for any errors in this document or their consequences, and reserves the right to make improvements and changes to this document without notice.

Trademarks

FME® is a registered trademark of Safe Software Inc. All brand or product names are trademarks or registered trademarks of their respective companies or organizations.

Document Information

Document Name: FME Server REST API Training Manual 2018.0
Course Overview

The FME Server REST API training is a brand new course that teaches FME users how to master the FME Server REST API and build their own web applications to solve data challenges. This course will teach users what the REST API is, how to use it, and how to build custom web applications that leverage the power of FME.

Course Structure

The course is made up of two sections:

- **Interacting with the FME Server REST API**
  - Getting Started with the REST API
  - Using a REST Client Tool to Make Calls to the FME Server
  - Interacting with Workspaces
  - Web Services, Direct URLs, and the REST API
  - Server Administration Tasks
  - Job Management

- **Developing with the FME Server REST API**
  - Best Practices when Building a Web Application
  - Creating a Dynamically Populated Form
  - Data Distribution Using ArcGIS
  - Data Visualization Using ArcGIS
  - Data Uploads and Downloads
  - Creating a Data Validation Application
  - Course Wrap Up

Course Length

This course is estimated to take twelve hours to complete.

About the Manual

The FME Server REST API training manual not only forms the basis for FME Server REST API training – in-person or online – but is also useful reference material for future work you may undertake with FME. It is updated for each major release of FME.
Course Resources

A number of sample datasets and workspaces will be used in this course.

On Your Training Computer

The data used in this training course is based on open data from the City of Vancouver, Canada.

Whether it’s a local computer or a virtual computer hosted in the cloud, you'll find resources for the examples and exercises in the manual at the following locations:

<table>
<thead>
<tr>
<th>Location</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\FMEData2018\Data</td>
<td>Datasets used in examples</td>
</tr>
<tr>
<td>C:\FMEData2018\Resources\RESTAPI\</td>
<td>All other resources used in the training</td>
</tr>
</tbody>
</table>

You should also find FME pre-installed, plus a digital copy of this manual.

Please alert your instructor if any item is missing from your setup.

You can find the latest version of FME Desktop and FME Server for Windows, Mac, and Linux - together with the latest Beta versions - on the Safe Software web site.

Course Etiquette

For online courses, please consider other students and test your virtual machine connection before the course starts. The instructor cannot help debug connection problems during the course!

For live courses, please respect other students’ needs by keeping noise to a minimum when using a mobile phone or checking e-mail.
Chapter 1 - Overview of the REST API

The REST API is a powerful tool that can be used to interact with FME Server. This section, explains what it is and how to use it. By the end of this section you should be able to understand what a REST API is and how you may use it in the future.
1.1 What is a REST API?

A REST API allows a user to interact with the server by creating requests and receiving responses. REST stands for REpresentational State Transfer and API stands for Application Programming Interface. A REST API is a type of API that follows a REST architecture.

There are six architectural constraints that all REST APIs follow:

1. Client-Server Architecture- this indicates that the client and the server are separate from each other and must be able to function independently.
2. Uniform Interface - the structure of the call and response must be uniform across all REST APIs.
3. Stateless - This means the server will not store a history of previous calls made to the server. Each call will be considered new.
4. Cacheable - all responses must be cacheable.
5. Layered System - It is possible to have many servers. The client should not be able to tell if it is connected to the end server or an intermediary server.
6. Code on demand (optional) - the server may return executable code to the client.

These constraints are intended to make the REST API fast and reliable.

Definition:

A client is defined by the Merriam Webster dictionary as a computer in a network that uses the services (such as access to files or shared peripherals) provided by a server.

How does the REST API interact with the FME Server?

The REST API acts like a intermediary directly communicating with the FME Server and relaying the responses back to the client. You can see the relationship between the FME Server and the API in the diagram below:

Diagram of the relationship between the FME Server and the client

The REST API reads the call which is composed of the HTTP Method, URL, and the body of the call. The FME Server will respond with JSON.
**Definition:**

JSON is a common data interchange format that has become one of the leading choices for supporting web sites and mobile device applications. For more information see the Tutorial: Getting Started with JSON article on the Knowledge Center.

---

**What is the purpose of a REST API?**

The REST API allows a user to interact with the FME Server without the use of the FME Server Graphical User Interface. This enables the user to create web applications that allow a client access to the FME Server without ever having to log in or directly access the server. It can also be used to automate processes within FME Server. For instance, an FME user can create a script to run health checks regularly on the Server.

---

**Ricky RESTless says...**

I use the FME Server REST API for everything! From sending notifications to running jobs and viewing job history. It seems like the possibilities of the FME Server REST API are endless.

---

**Examples of web applications created with the REST API**

If you are curious about all the possibilities of using the REST API, please visit the FME Server Demos. The FME Knowledge Center has live demonstrations that display the vast capabilities of using the FME Server and the REST API.

The example below demonstrates how to gather data from the FME Server using a web application. It allows a user to request public transit information within an area specified by the user and you can view this Live Online demo.

This demonstration allows the user to draw a polygon on an area. Then, the user will set the parameters for the call, this includes which layers to download and the coordinate system. The user hits request data and will receive a zip file with the requested information. If you wish to rebuild this demo, visit the Data Distribution with Web Maps demo.
1.1 What is a REST API?
1.2 What Components are in a Request?

Requests are made through URLs. Responses are formatted in JSON.

Diagram of the components in a request

Before getting started, it is important to know what an example request to the server looks like. For a full list of example calls, please visit the REST API documentation.

HTTP Method:

HTTP Methods are essential to every call to the REST API. **HTTP Methods represent an action to the FME Server that you would like to complete.**

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Action</th>
<th>Uses in FME Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>Create</td>
<td>Create a publication, project, notification.</td>
</tr>
<tr>
<td>GET</td>
<td>Read</td>
<td>Perform a health check, get information on the server, get notifications on the publication.</td>
</tr>
<tr>
<td>PUT</td>
<td>Update and Replace</td>
<td>Replace a publication, update a project, update user information</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete</td>
<td>Delete topics, users, roles.</td>
</tr>
</tbody>
</table>

**Ricky RESTless says...**

*I often use GET and POST requests. GET requests get information already on the FME Server. POST requests post new information to the FME Server. For instance, I can GET a list of projects on the FME Server. Then, I can use a POST request to create a new project.*

**Request URL**

The Request URL is how you can make the call. It is composed of keywords that indicate to the server what you are trying to do. In this tutorial, we will discuss the different requests you can make. Each request will have the same base URL:

http://<yourServerHost>/fmerest/v3/
Parameters influence the response of the REST API by specifying a constraint on the call. There are two main parameters within the Request URL.

**Path Parameter**- Path parameters are required by the call and act as a part of the endpoint. The following call requires the name of the publication to work. The publication name is My Sample Publication. Because we cannot enter spaces in the URL, %20 indicates the space.

```
```

**Diagram representing the path parameter**

**Query String Parameters**- Query string parameters are specified after the question mark in the endpoint in the Request URL.

```
```

**Diagram representing the query string parameter**

In the above request URL, there are two parameters, the limit, and the offset. The parameters are separated by the ampersand (&). The order does not matter for the query string parameters so you may place the offset before the limit and the request will produce the same result.

A request URL may contain both path parameters and query string parameters.

**Request Header**

REST Headers set the context for the type of call you are trying to place. They provide the required information for the call to take place. To begin there are three main headers we will use.

1. **Authorization**: Almost every call to the FME Server requires authorization. Authorization ensures that the call being placed to the FME Server is made with permission. Authorization is given by a token. To receive a token visit, http://<yourServerHost>/fmetoken/

   To learn more about authorization visit, Configuring Authentication for Security Resources.

2. **Accept**: Accept dictates the language of the response being returned. This ensures the client is prepared for the response. During this tutorial, the accept will be application/json.

3. **Content-Type**: The content type tells the server what language the request body is in. During this tutorial, we will use JSON, x-www-form-urlencoded, and binary.

**Request Body**

The request body defines the parameters of what is being completed. It specifies the information to be modified, created, or deleted. We will provide examples later in the tutorial.

**Example Request**

The example below exports a project as a download and has a total of four parameters. In the request URL, there is a path parameter after projects. FME_PROJECT_TEST is the name of the project that will be downloaded. The Query String Parameter (accept=contents) specifies the content type. There are two form parameters found in the body of the call. There is the excludeSensitiveInfo which determines if any sensitive information will be returned in the downloaded package. There is also export ProjectName which is where you can specify the name of the project when it is exported.
HTTP METHOD
POST
REQUEST URL
http://<yourServerHost>/fmerest/v2/projects/FME_PROJECT_TEST/export/download?
accept-contents
REQUEST HEADER
Content-Type: application/x-www-form-urlencoded
Accept: application/json
REQUEST BODY
excludeSensitiveInfo=false&exportPackageName=ProjectPackage.fsproject

Diagram representing the content of a call
This is the same request in the HTTPCaller, this is a transformer used within FME to access REST APIs. We will be working through exercises later in the course using the HTTPCaller, but for now, it's just important to know the structure of the call.

The same call in the HTTP Caller
1.3 Authorization and Authentication

Security is always a significant consideration when working with any online service. The REST API works with security in two ways: authorization and authentication. **Authentication proves that the user is who they say they are. Authorization verifies that the user is authorized to make the call.**

Most calls to the FME Server require authorization. The REST API uses tokens to prove the user is permitted to make the call to the server. A token is a string of encrypted information that is sent between the client and the server. Token Security will not be as secure as other methods, as the security of the system depends on controlling access to the tokens.

When replacing `<yourServerHost>` with your hostname. You can use localhost if you are working on the machine where FME Server is installed. If you are accessing from a remote machine, you need to specify an accessible URL either by IP address or server hostname.

**WARNING**

*Tokens can hold a lot of power. Tokens will have the same permissions as the owner of the token. For instance, an admin token will have the power to create a fmesuperuser. Before using a token, consider what permissions that token has.*
1.4 Token Management

There are many ways to get a token to use for authentication.

Ways to get a token

1) Through the FME token page

To generate a token visit:

http://<yourServerHost>/fmetoken/
or http://localhost/fmetoken/ if you are using a training computer.

Once you access your server, you will be asked to verify your username and your password and be asked to specify when you would like your token to expire. The maximum amount of time you can issue a token for is two years.

2) Through your FME Server

http://<yourServerHost>/fmeserver/#/home

On the FME Server, click on the user icon. Then click Manage Token.
3) Through the REST API Homepage

http://<yourServerHost>/fmerest/apidoc/v3/

On the top right-hand corner of the page you will see a Green Button that says "Get Token". Click here, to get a token or look up your existing token.

Tokens and Users

You may also request and manage tokens through the web application. Often users will hard code the Guest user into the application. This prevents the user from having to log in to access the data. The methods of this will be discussed in a later tutorial on building a web application.

Using a Token in a Call

There are two methods of including a token to the FME Server. You may include the token in the query parameter and the authorization header. The actual methods of using these different practices will be discussed later in the tutorial. However, for now, it's important to know the pros and cons of each method.

Query Parameter- You may include the token right in the Request URL of the call. This is a quick and easy way to use a call that requires a token.

http://<yourServerHost>/fmerest/v3/info?fmetoken=<yourToken>

However, this is not recommended because the token will be visible in the URL.

Authorization Header- You may also include the token in the Request Header. This is the preferred method because the token will be more hidden inside the call.

Troubleshooting Authorization Errors

If a call is not authorized you will receive an error code back from the system. You may receive a Forbidden or Unauthorized message. Forbidden (code 403) means the request was understood but refused. This indicates you do not have the correct permissions to complete the request. Unauthorized (code 401) indicates the credentials were missing or incorrect. This may occur
if the request needs a token and it is not present, or the token is expired.
This exercise is meant to teach users how to create a new user with limited permissions. When developing with the REST API it is advised not to use an admin token. The admin token has a lot of power that could be dangerous if exposed. By creating an account with limited permissions we are limiting risk if the token is exposed.

1) Open FME Server

Our first step will be to visit the FME Server. We can do this by going to:

<yourServerHost>/fmeserver

If you are using a training computer go to: localhost/fmeserver

If you are using a training computer login as the admin using these credentials:

**Username:** admin  
**Password:** admin

**WARNING**

On some training machines the FME Server will not automatically work. If once you log in, you see an error message:

"Could not connect to FME Core. Please ensure that it is running."

Please open Task Manager, find all PostgreSQL tasks, select End task for each PostgreSQL task you see. Then restart your FME Server. You may restart your FME Server by finding FME Server 2018.1 in the Windows Start Menu. Then, select Restart FME Server. After the FME Server has restarted, try logging in again.

2) Go to the Users page

Now, we are going to navigate to the Users page. We are going to the left-hand panel find **Security** and then click **Users**.

3) Create a new user

Click the New button at the top right-hand corner of the page.
When prompted, create a new user with the following parameters:

- **Username**: restapi
- **Full Name**: Forest Apier
- **Password**: restapi

4) Scroll down and assign permissions

When assigning permissions for your future users visit User Permissions to view the full documentation.

Now assign the following permissions:

<table>
<thead>
<tr>
<th>Permission</th>
<th>Level of Permission</th>
<th>Description of Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Workspace</td>
<td>Advanced</td>
<td>Can access the Run Workspace page and access Job Directives when running workspaces.</td>
</tr>
<tr>
<td>Jobs</td>
<td>Manage</td>
<td>Access and manage the jobs of all users. You can, cancel any job that is currently running, remove the history of jobs that were previously run, and manage Job Queues. (Also requires Manage permission in Engines &amp; Licensing.)</td>
</tr>
<tr>
<td>Repositories</td>
<td>Create</td>
<td>Access the Repositories page and create repositories.</td>
</tr>
<tr>
<td>Individual Repositories</td>
<td>Samples = Download, Read, Run</td>
<td>Can download workspaces and other repository items from FME Server into Workbench. Can view repository information. Can run repository workspaces from FME Server.</td>
</tr>
<tr>
<td>Workspace Viewer</td>
<td>Access</td>
<td>Can access the Workspace Viewer.</td>
</tr>
<tr>
<td>Resources</td>
<td>Create</td>
<td>Access the Resources page and create new resources.</td>
</tr>
<tr>
<td>Individual Resources</td>
<td>Data = Full access</td>
<td>Can access, read and download a file. Can list the folders and files of a resource. Can write to files, upload files, and delete files.</td>
</tr>
<tr>
<td></td>
<td>Temp= Full access</td>
<td></td>
</tr>
<tr>
<td>Projects</td>
<td>Create</td>
<td>Access the Projects page and create projects.</td>
</tr>
<tr>
<td>Dashboards</td>
<td>Access</td>
<td>Access the Dashboards page.</td>
</tr>
<tr>
<td>Engines &amp; Licensing</td>
<td>Manage</td>
<td>Configure engines and licensing, except job queues (Also, requires Manage permission in Jobs).</td>
</tr>
</tbody>
</table>

If you are creating a web application the permissions for the token should be limited to a specific repository.
The settings should look like this:

<table>
<thead>
<tr>
<th>Permissions</th>
<th>Access</th>
<th>Create</th>
<th>Manage</th>
<th>Advanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Workspace</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jobs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Schedules</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Repositories</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Workspace Viewer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Publications</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subscriptions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Topics</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resources</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connections</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Projects</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dashboards</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Engines &amp; Licensing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Cleanup</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Services</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Individual Repositories should look like this:

![Repositories Page]

The Individual Resources should look like this:

![Resources Page]

5) Click OK to create the restapi User

![Security Page]

6) Log into the FME Server as the restapi user and find your token

Log onto the FME Server as the restapi user.

- **Username:** restapi
- **Password:** restapi
Once we log into our FME Server it is important to check that all of the correct components are there.

In the left hand panel we should see, Run Workspace, Jobs, Repositories, Workspace Viewer, Resources, Projects, Dashboards, and Engines & Licensing.

Next, we can locate the user icon. This is the area we access if we wish to find more information about the user.
After we have selected the user icon, you can click Manage Token. To find the token.

Manage Token

You will get a new token if you log into FME Server and you do not have a valid token.

Current Token: b574cd3354c3f80577c0de99c99de32
Expiration: Today at 16:23:09

Then, we can see the Current Token. Or, we can get a new token in this area. Click, New Token to extend the Expiration time of the current token.
Copy the Current Token.

7) Save this token
Open a Notepad++ Document and paste the token in the document for easy access.

CONGRATULATIONS

By completing this exercise you have learned how to:
- Create a new user in FME Server
- Assign specific permissions to the user
- Get the new users token
Chapter 2- Getting Started with the REST API

This chapter is gets the user started with the REST API. It introduces the REST API Homepage and teaches the user potential responses from the FME Server.
2.1 Exploring the REST API Homepage

The FME Server REST API Homepage is an essential tool for any API user. This website provides the documentation needed to learn about the FME Server REST API. It also allows the user to view all calls available on the REST API. Each call has detailed documentation about the call and any parameters connected to the call. It also allows the user to practice a call before implementing it.

Now that you know the basic information on using the REST API, we can explore how to use it. The first step is to visit your homepage for the REST API. To find this, go to

http://<yourServerHost>/fmerest/

If you are doing this course on the training computer go to http://localhost/fmerest You may also access this same page through your FME Server. Simply, go to your FME Server homepage. Locate the help button and click on the REST API button. This page provides helpful resources when first learning the REST API. This includes sample calls you can make to the server. On the main toolbar click on the API Link.

Here you can explore the various calls you can make to the server. Find the second category called health check and expand the call. You should see documentation similar to this.

This page outlines everything you need to know about the call and its potential. Click “Try it out!”, this connects to your server and will provide a response to your call. However, before you receive a response you will have to connect to your server.

Here is the message you should receive. You will be prompted for the server username and password. This is where you will have to request a token to complete your call.
First, click Lookup Existing Token then click Generate Token. You should receive a message with your token and a statement that you may now try out the examples. Copy the token and paste it into any text editing software we will use it in later examples. It is important to note that the token acts as a verification you have permission to gain access to the server. It is a security feature, and you can gain access to tokens on your FME Server Rest API page.

http://<yourServerHost>/fmerest/

After you have made a call to the FME Server, you will receive a response from the FME Server.
2.2 Understanding the REST API Response

The response structure is very similar to the request structure. This features the response body, response code, and the response header.

Response Body

Most calls to the FME Server will result in a response. This is what makes the FME Server REST API a very powerful tool. With this response, you may automate another process to run. The Response Body will always be in JSON.

```
{
    "status": "ok"
}
```

Response Code

The request above should result in a code 200 which is outlined in the example call:

<table>
<thead>
<tr>
<th>Response Status Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HTTP Status Code</strong></td>
</tr>
<tr>
<td>200</td>
</tr>
</tbody>
</table>

Potential Response Codes

Below there is a table listing common response codes. For a full list, please visit the [Rest API Documentation](#).
Chapter 3- Using a REST Client Tool to Make Calls to the FME Server

Now that you understand what the REST API is and how the calls and responses work; it's time to start interacting with the FME Server.

For the purposes of this training course, we will use an external tool named Postman which can be downloaded here: https://www.getpostman.com/apps (this is already on the training computers).

If there is another REST client tool you are familiar with you may use that as an alternative, but the instructions in this manual will refer to the Postman interface.

Why use a REST Client tool?

A REST Client is a medium to practice calls with. The client accepts the REST calls and will communicate directly with the FME Server. There are several reasons why a user will use a REST Client tool:

- Easy to use interface to input calls
- Save the calls and use them later
- Can generate code snippets from the calls to use in applications

Ricky RESTless says...

I use Postman to test calls before using them in the HTTPCaller Transformer in FME Workbench. By using the REST API calls in FME Workbench I can automate many FME Server processes.
Exercise 2  Using Postman to Make an HTTP Request

<table>
<thead>
<tr>
<th>Data</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>Retrieves information from the Server Using the REST API</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>The healthcheck call from the REST API</td>
</tr>
</tbody>
</table>

This exercise demonstrates how to make a call to the FME Server using the REST API. Postman is a REST Client Tool. REST Client Tools allow a user to enter in a Request URL and receive messages directly from the server. This exercise demonstrates the healthcheck call which will check if the FME Server is up and running properly.

We are about to make the following call in Postman.

| GET | http://<yourServerHost>/fmerest/v3/healthcheck?textResponse=false |

This call requires no authorization and should run immediately. Notice that there is a string query parameter in this call. This call will automatically return a JSON response; however, you may override this by setting the text response to true. This will return a text response instead of JSON.

1) Download Postman- If you are not already using a virtual machine

The first step is to download Postman. Simply click this link https://www.getpostman.com/apps and download the app. It is available on Windows, Linux, and Mac.

2) Open Postman

Welcome to Postman! We are now ready to make our first call to your FME Server using the URL above. Once you open Postman you should see this page.

3) Click on Request.
Alternately, click on New > Request

4) Fill in the request and click Save.

From here, we receive a page to save our request. This is used if you would like to save the call to use it in the future. Typically, the name of the call would be much more descriptive. However, for the purposes of this demonstration, make up something creative, such as “MyFirstCall”. For this call, you can create a new collection for training purposes.

To create a new training collection click +Create Folder. Then name and save your collection.
5) Copy the URL into the toolbar.

Copy the URL into the toolbar and hit send! Notice, we are using the GET HTTP Method for this call. To understand what this method means, please review the table in Section 1.2.
GET

http://<yourServerHost>/fmerest/v3/healthcheck?
textResponse=false

Please note, you will need to replace yourServerHost with localhost if you are working on a training machine

6) Review the response from Postman.

Review your call. This is the response you receive.

Click the Headers section to see a more detailed view of the call.

TIP

If you are using a training computer it may be possible that your FME Server is not working correctly upon opening. If the FME Server is not available, go into Windows Task Manager, quit all Postgres tasks. Then, got into Windows Services and restart the FME Server Database and FME Server core. If this does not resolve the issue please visit the FME Server Troubleshooting Guide

Important Notes:

- The response is in JSON. This is found in the header Response Header Content-Type.
- In the top right-hand corner, you should see Status: 200 OK. This indicates it has successfully interacted with the server. This call required no authorization. However, most calls to the server do require authorization. The next step will be to make a call that does require authorization from the server.

CONGRATULATIONS

By completing this exercise you have learned how to:
• Set up and use Postman
• Create a call in Postman
This shows how to create an authorized call in the FME Server REST API. Authorization is an essential component in the REST API. Most calls do require authorization as a way to verify that the user may access information from the server. Authorization in the FME Server REST API is handled with tokens. This will be demonstrated and further explained below.

1) Create the Call Below

Copy and Paste this URL into the Toolbar in Postman and then click send!

This call will require a token. This next part of the demonstration will show you what happens when you make an unauthorized call to the server.

```
GET
http://<yourServerHost>/fmerest/v3/info
```

You should receive a message that says 401 Unauthorized. This is because we have not used a token to authorize the call. The next part of the guide will go over how to make a request on the server using a token.

2) Get A Token

There are many ways to get a token in the FME Server. The first way is to go to FME Server web interface. The right-hand corner features a user symbol. That’s the user settings, then go to manage token.
Another way to get a token is by visiting,

http://<yourServerHost>/fmetoken.

3) Use the Token in a Call

Now that we have a token we may use it in the request. Next to "GET" paste the following URL into Postman. Replace "yourTOKEN" with the token you have received from the server.

GET

http://<yourServerHost>/fmerest/v3/info?fmetoken=<yourTOKEN>

You should receive the 200 response code from the FME Server. Please note, this is not the preferred method of authorization. By putting the token into the URL it is visible and will be less secure.

4) Use the Token in a Call Using the Preferred Method

The preferred method is to use the Authorization Header. To find where to place the Authorization Header, look underneath to the URL find the Headers section. In here, under key write Authorization. Then set the value to fmetoken=your token. The URL should be

GET

http://<yourServerHost>/fmerest/v3/info

Now try running the call by clicking the Send button.

5) Review the Response

Once you click Send, you will see this response in Postman. This call returns the build and version.
In this section, we went through the basics of making an authorized call to the FME Server.

6) Access the Presets section to add a preset

On the right-hand side of the screen, there is a section to create a Preset. Presets allow a user to save this authorization information for the future.

7) Create a new preset

Click add to add the preset.

Quickly add groups of header key/value pairs to the request. Start typing the name of the preset name and it’ll show up in the dropdown list.

Fill in the values to create the new preset:

- **Header Preset Name**: Token
- **Key**: Authorization
- **Value**: fmetoken token=YourToken
- **Description**: Token Expires

To use this Header Preset in future calls simply type token in the key section and select the preset. Then your header will automatically fill with the preset.

**Other Authorization Option**
You may also use the standard authorization within Postman. However, this course will use the token in the header.

**CONGRATULATIONS**

By completing this exercise you have learned how to:

- Use a call with authorization in Postman
- Create a Postman preset to save the token for later calls
Chapter 4- Interacting with Workspaces

In this chapter, we will look at the various ways to run a workspace on your FME Server using the REST API.

There are two main ways a user can submit a job on the REST API. The first is synchronously and the second is asynchronously.

A **synchronous** task has to be completed before the next task will begin. Running a call synchronously is good when you need to know the results immediately, and it is a short task.

An **asynchronous** task will not wait for the job to complete before returning a response. Instead of the job results, it will return a job ID immediately then you can use the ID to obtain the status of the job. Running a call asynchronously is good if you don't need an immediate response or if it is a large task.
Exercise 4  Running a Synchronous Job With Standard Parameters

<table>
<thead>
<tr>
<th>Data</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>To run a job synchronously using the FME REST API</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>How to use the Transact call</td>
</tr>
</tbody>
</table>

This exercise demonstrates the Transact call, which is used to run a job synchronously on the FME Server. To find more information on the Transact call visit your FME REST API page and look for the transformations section or visit the Rest API Transformations documentation.

An example of running a job synchronously would be:

```plaintext
POST http://<yourServerHost>/fmerest/v3/transformations/transact/Samples/austinDownload.fmw
```

Notice the Transact statement indicates to the FME Server that you will wait until the process is complete before receiving a result. In Postman you will need to set up the headers section and the URL as well. Carry out the following steps to set up this call in Postman.

---

**WARNING**

*Please note that all calls to submit a job - whether synchronous or asynchronous - requires authorization*

---

**Note**

*While synchronous calls are easier to use in a web application compared to asynchronous calls, the application will be stalled until the call is completed. The more data being processed in a workspace, the longer it will wait. While some jobs will be completed almost instantaneously, jobs involving large data amounts or complex computations can take a significant amount of time. Synchronous calls are best saved for short jobs. This is a trade-off that should be considered while developing with the FME Server REST API.*

---

1) Enter in the URL

Click the plus sign to open a new tab in Postman. The first step is to add in the URL in Postman. Paste the URL into the toolbar, then change GET to POST using the drop-down.

```plaintext
POST http://<yourServerHost>/fmerest/v3/transformations/transact/Samples/austinDownload.fmw
```

Users on the training computers can change <yourServerHost> to localhost and other users should change yourServerHost to their server's host name.

The Transact call documentation can be found here.

2) Enter in the Headers
In this call we have a request body, that will be in JSON. This is displayed in the Content-Type section. Additionally, we are requesting JSON back which is under Accept = application/JSON. The Authorization is where you would enter your token.

- **Content-Type:** application/json
- **Accept:** application/json
- **Authorization:** fmetoken token= <yourTOKEN>

To enter in the token we can use the preset created in the last exercise. In the key-value type, the word token and the preset created will appear.

To enter in the headers required for this call. Click on the Headers tag as demonstrated below.

3) **Enter in the Body and select the ”raw” option**

Next, we need to enter the body of the call. Click on the body section of the call and click on the raw button and paste in the body provided. In Postman you will need to use the keyboard shortcuts to copy and paste.

Body:

```json
{
  "publishedParameters": [
    {
      "name": "MAXY",
      "value": "42"
    },
    {
      "name": "THEMES",
      "value": [
        "airports",
        "cenart"
      ]
    }
  ]
}
```

4) **Click Send**

Click Send! Now, wait until the call is returned (approximately 5-10 seconds) and you will receive a message like this.
5) View in the FME Server

Now that the job has been submitted, you can view it in the FME Server. Go to your FME Server and log in as the restapi user and click Jobs > Completed page on the left-hand sidebar.

You should be able to notice a few things. The job was run by the restapi because the token that was used belonged to the restapi.

6) Find the Job Log and find THEMES

Click on the Job.

The jobs page provides a more detailed description of the job. Here, we can find the job log. The job log provides information such as, what features were written, how many features are written, and what engine was used to complete the job.

Scroll down to the Jobs Log

The Job Log will always be after the completed section. The completed section gives a quick overview of the job where the log is much more detailed.

Find the THEMES in the Log

Simply, hit ctrl + f, to find THEMES in the log. It should display 'airports cenart'.

We know that the job was completed with the requested parameters.
Let's update the parameters and try the call again.

7) Find the Published Parameters for the Workspace

Go to the RunWorkspace Page. Then, select Samples as the Repository and austinDownload.fmw as the workspace. Scroll down the page until you see the advanced section. Click on the plus sign, then find the published parameters. Here, we have all the parameters we can modify in the workspace. Look at the options, this time we will run the call with the railroad and streetcl options.

8) Open Up Postman and modify the body

If you kept Postman open simply click on the body tab again. Now change the THEMES values to railroad and streetcl.

```json

"name": "THEMES",
"value": [
  "railroad",
  "streetcl"
]
```

Then click SEND.

9) Find the job in your FME Server

Open your FME Server, then click on the left-hand panel of the Server where it says, Jobs. Click the latest austinDownload job. Scroll down to find the job log.

Here, you can investigate if the job was successfully completed and what parameters were used. Here, we can see the correct features were written.
CONGRATULATIONS

By completing this exercise you have learned how to:

- Create a call that runs a job synchronously
- Check the Job Log to see if the job was run correctly and with the right parameters
- Change the parameters in the call
Exercise 5 Running a Job Asynchronously

<table>
<thead>
<tr>
<th>Data</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>To run a job asynchronously using the FME REST API</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>How to use the submit call</td>
</tr>
</tbody>
</table>

This exercise demonstrates the submit call, which is used to run a job asynchronously on the FME Server. To find more information on the Submit call visit your FME REST API page and look for the transformations section or visit the Rest API Transformations documentation.

If you run a job asynchronously you do not have to wait for the job to be completed before making the next call. By running the job asynchronously you will receive an ID back from the FME Server. You may use this ID to call the FME Server again to get a status update on the call.

We will run the exact same call, but instead of Transact, we will change the URL to Submit. By, submitting you will submit the job for processing.

1) Paste the URL below into Postman

Click the plus sign to open a new tab in Postman. Copy the URL below and paste it into Postman.

| POST | http://<yourServerHost>/fmerest/v3/transformations/submit/Samples/austinDownload.fmw |

2) Enter in the Headers

Fill in the Headers in Postman:

- **Content-Type**: application/json
- **Accept**: application/json
- **Authorization**: fmetoken token= <yourTOKEN>

To enter in the token we can use the preset created previously. In the key-value, type the word token and the preset created will appear.

To enter in the headers required for this call. Click on the Headers tag as demonstrated below.

3) Enter in the Body

Next, we need to enter the body of the call. Click on the body section of the call and click on the raw button and paste in the body provided. In Postman you will need to use the keyboard shortcuts to copy and paste.

Body:

```json
{
    "publishedParameters": [
        {
            "name": "MAXY",
            "value": "42"
        },
        {
            "name": "THEMES",
            "value": [
                "airports",
                "cenart"
            ]
        }
    ]
}
```
4.2 Exercise 5- Running a Job Asynchronously

Click Send

4) Review the Response

The response from the call should be:

```
1 { "id": 1502 }
```

Please note that you may receive a different number back from the FME Server. This is okay, it is just the job ID.

5) Find the Job in your FME Server

Open your FME Server, then click on the left-hand menu of the Server where it says Jobs. Click the Job ID that matches the number you received from the FME Server. Note that the translation might still be running, even though you already received a response. This is because this job is running asynchronously. Click the Job ID to see its full status.

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Workspace</th>
<th>Repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>1502</td>
<td>austinDownload.fmw</td>
<td>Samples</td>
</tr>
</tbody>
</table>
```

Then, find the Log. If you scroll to almost the bottom of the page, you can see a summary of the features written.
Here, you can investigate if the job was successfully completed and what parameters were used. Here, we can see the correct features were written.

CONGRATULATIONS

By completing this exercise you have learned how to:
- Create a call that runs a job asynchronously
- Check the Job Log to see if the job was run correctly and with the right parameters
Exercise 6  
Retrieve Job Results with the REST API

<table>
<thead>
<tr>
<th>Data</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>To use the REST API to receive information about a completed job</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>How to use the jobs call</td>
</tr>
</tbody>
</table>

After you have submitted the call and received the ID you can check the job status with this call:

To find the documentation on this call, please visit the Rest API Transformations - GET Jobs documentation.

Please note that this call will require authorization

1) Enter in the below URL into Postman

Click the plus sign to open a new tab in Postman. To enter in this URL, you will need to modify the last number to the job id you received from the response in the previous call:

```
GET
http://<yourServerHost>/fmerest/v3/transformations/jobs/id/<JobID>
```

2) Enter the following header into Postman

- **Accept**: application/json
- **Authorization**: fmetoken token=<yourToken>

```
<table>
<thead>
<tr>
<th>Authorization</th>
<th>Headers (2)</th>
<th>Body</th>
<th>Pre-request Script</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accept</td>
<td>application/json</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authorization</td>
<td>fmetoken token= YOUR TOKEN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

This will return with a response that the job has been completed.

This information will be displayed at the bottom on the response in Postman.

```
},
"id": 1502,
"timeFinished": "2018-05-31T16:18:38-07:00",
"engineName": "localhost_Engine1",
"timeSubmitted": "2018-05-31T16:22-07:00",
"status": "SUCCESS"
}
```

**CONGRATULATIONS**

*By completing this exercise you have learned how to:*
- Find the results of a job based off the job id.*
4.3 Exercise 6- Retrieve Job Results
**Exercise 7**

Run a Job Synchronously and Upload Data: Transact Data

<table>
<thead>
<tr>
<th>Data</th>
<th>C:\FMEData2018\Resources\RESTAPI\Chapter4Exercise7\12656-datapoints.csv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>To use a single call to upload data and run a job.</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>How to use the Transact Data Call</td>
</tr>
<tr>
<td>Starting Workspace</td>
<td>C:\FMEData2018\Resources\RESTAPI\Chapter4Exercise7\Chapter4Exercise7.Start.fmw</td>
</tr>
<tr>
<td>Ending Workspace</td>
<td>C:\FMEData2018\Resources\RESTAPI\Chapter4Exercise7\Chapter4Exercise7.Complete.fmw</td>
</tr>
</tbody>
</table>

The Transact Data call works by uploading data and running a workspace with one call to the server. It will simply run the workspace as it is and write the output to the location specified in the writer. That’s good for inserting new data into a database. Once the call is completed the FME Server will return a 200 OK message. This indicates that the call was submitted successfully. However, it will not provide feedback on whether the job was completed successfully on the FME Server. It is recommended to periodically check the server to ensure your jobs are running successfully on the FME Server. Getting the status of jobs can be accomplished through the REST API as well. These calls will be displayed in Exercise 8.

Let’s start by creating a workspace that is compatible with the Transact Data call.

1) **Open the Chapter4Exercise7.Start Workspace**

Start FME Workbench and open the Chapter4Exercise7.Start.fmw workspace.

This is a very simple workspace that reads a CSV File and produces a Shapefile.

However, there is one important change that is required for this translation to work in this call. The CSV has to be an optional parameter. This is because the data will be sent in the body of the POST request to the URL and not by setting the path to the source dataset parameter in the workspace. The source dataset parameter will need to be left unset for this to work, so we need to make it optional to prevent the workspace from throwing an error about this.

2) **Set the Source Dataset Published Parameter to Optional**

Set the Source Dataset Published Parameter to optional by right-clicking on the published parameters and selecting Edit Definition.
Then select the optional setting in the within the parameter settings.

If you are not using a training machine you may also set the Default Value to navigate your Reader Dataset

C:\FMEData2018\Resources\RESTAPI\Chapter4Exercise7\12656-datapoints.csv

3) Create a folder in your FME Server for your output

First, go to your FME Server and log into the restapi account. On the left-hand side of the page locate the "Resources" and click on it. Next, click on Data. Finally, click on Create Folder and entitled RESTTraining

After we create the folder, we can find the path to the folder by checking the folder. Then, go to Actions and select Properties.
Select the RESTTraining File then select Actions

Select Properties

File Properties

Name: RESTTraining
Date: Today at 14:01:49
Type: DIR
System Path: $(FME_SHAREDRESOURCE_DATA)/RESTTraining

Copy the System Path.

4) Update the Shapefile destination to a Shared Resource File in your FME Server

Back in the FME Workbench change the Shapefile to a destination within your FME Server.

To do this right click the DestDataset published parameter, then click Edit Value. Then, insert a file path to your shared resources.

$(FME_SHAREDRESOURCE_DATA)/RESTTraining
Save the Workspace before publishing.

5) Upload the Workspace to the FME Server

Now, it’s time to upload the workspace to your FME Server. This can be done by clicking the Publish to FME Server Button or by selecting File > Publish to FME Server from the menu bar.

As this is the first time we’ve connected to our FME Server, we’ll need to create a new connection, so in the Publish to FME Server wizard select Add Web Connection from the drop-down menu.

In the dialog that opens enter the parameters provided by your training instructor. In most cases the parameters will be as follows:

- **FME Server URL**: http://localhost
- **Username**: restapi
- **Password**: restapi

Click Authenticate to confirm the connection and return to the previous dialog. Make sure the newly defined connection is selected and click Next to continue.

For this exercise, we’ll create a new repository by clicking the New button. When prompted enter the name RESTTraining.

Save the Workspace as Chapter4Exercise7.Complete and select the checkbox to upload data files.
Then click Next to continue the wizard.

In the final screen of the wizard we can register the workspace for use with various services.

Select the Job Submitter service as this is the only service we are using for now.

Click Publish to complete publishing the workspace.

### 6) Run the Workspace in your FME Server

Now we will go to our FME Server as the restapi user to prove this works.

On the left-hand menu, we will see a Run Workspace tab.

Now select the following options:

- **Repository**: RESTTraining
- **Workspace**: Chapter4Exercise7.Complete.fmw
- **Service**: Job Submitter
Now click Run!

**RESTTraining/Chapter4Exercise7.Complete**

Repository: RESTTraining

Workspace: Chapter4Exercise7.Completa.fmw

Service: Job Submitter

Email results to: 

Published Parameters

Source CSV (Comma Separated Value) File(s) (optional): $(FMF_MF_DIR)126656-datapoints.csv

Destination Esri Shapefile Folder: $(FMF_SHAREDRESOURCE_DATA)/RESTTraining

Advanced

Run

7) Check that the Shapefile was successfully produced

Go to Resources > Data > RESTTraining and ensure a Shapefile was produced.

Delete the Shapefile. This is done by selecting all files and then select Actions and selecting delete. We are deleting the Shapefile because we are going to use the REST API to submit this job again.
8) Use Postman to complete the Transact Data Call

*Please note that this call requires authorization*

Click the plus button to create a new tab. We'll now use the call to transact data to upload a CSV file and run the workspace through the REST API.

The call is:

```
```

*If you saved the workspace to a different Repository you will have to modify the call above*

In Postman set up the call with the following parameters:

- **Authorization**: fmetoken token=<yourToken>

To enter in the token we can use the preset created previously. In the key-value, type the word token and the preset created will appear.

- **Accept**: application/json

The file to be uploaded will be sent in the body of the call. Click on the body tab and then the binary button. This will give you an option to upload a file. Select the 12656-datapoints.csv found in the C:/FMEData2018/Resources/RESTAPI/Chapter4/Exercise7. Then run the call by pushing the SEND button.
You should receive a message 200 OK this indicates the call has completed. Check the Jobs page in FME Server to confirm the job was correctly submitted. Also, check the output location to see if the Shapefile was created.

CONGRATULATIONS

By completing this exercise you have learned how to:

- Use the transact data call to upload a file and run a workspace in one call
- Modify the parameters of a workspace to be used with transact data
- Use a shared resource as the file output location
4.5 Using REST API Commands in a Workspace

All of the calls that have been practiced in this tutorial can be used in an FME Workspace. This is an excellent way to work with the REST API without any coding experience.

There are some transformers you should be aware of before we get started.

**Creator**

The **Creator** transformer is an excellent tool for testing a workspace. It will trigger the workspace without the need to add data.

**HTTPCaller**

The **HTTPCaller** transformer is used to connect to REST APIs. It separates the call into categories similarly to Postman. Simply enter in the Request URL, HTTP Method, Query String Parameters, Header, and Body. The body is broken down into several parts. You can upload data, or write the body of the call as we have before.

**JSONFlattener**

The **JSONFlattener** transformer takes the JSON responses from the FME Server and can break down the components so they can easily be used for future calls to the server. JSON response can be quite long so it is essential to place the response into attributes that can be used in the workspace.

**AttributeKeeper**

The **HTTPCaller** will produce attributes that you may use in your calls. However, you may only wish to keep one. The **AttributeKeeper** transformer acts as a filter, it will only keep the attributes you specify.

---

**Ricky RESTless says...**

All of the FME Server REST API calls can be used in Workbench. This can allow you to create workspaces to run jobs on the FME Server, create repositories, and handle server admin tasks. You may also use other REST APIs in Workbench.
Exercise 8

Using REST API Commands in a Workspace

<table>
<thead>
<tr>
<th>Data</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>To use the REST API in a workspace</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>How to use the HTTP Caller to access the FME Server</td>
</tr>
<tr>
<td>Starting Workspace</td>
<td>None</td>
</tr>
<tr>
<td>Ending Workspace</td>
<td>C:\FMEData2018\Resources\RESTAPI\Chapter4Exercise8\Chapter4Exercise8.Complete.fmw</td>
</tr>
</tbody>
</table>

A very simple workspace you can create would be one that triggers multiple workspaces in the FME Server. In this workspace, we are going to step up a workspace to run asynchronously, wait for a response, if the response is positive the next workspace will run. So let's begin!

`Runs a Job asynchronously and finds the job id`

Create a call to run asynchronously

1) Create a new workspace in Workbench and add a Creator Transformer to the canvas

Open a new FME workspace and then add a Creator transformer to the canvas. Keeping the default settings are okay. The Creator creates a single feature which will initiate the HTTP caller.

2) Add an HTTPCaller to the canvas

Next, add the HTTPCaller. Use the Request URL to write a submit command. You may use the call below or test it out with another workspace.

There are a few different ways to authorization with FME Server in the HTTPCaller:

1. Use the token in the URL as a query string parameter
2. Use the token in the authorization header
3. Check use authentication and set to basic authentication and enter your username and password
4. Check use authentication and set up an FME Server Web Connection

**Note**

*At the beginning of the course, we discussed the difference between authentication and authorization. This was referring to the response codes sent from the FME Server. There is a difference between an authentication error and authorization error. However, within the HTTP Caller authenticating or authorizing your call can be used interchangeably.*

Please note that all calls in this section will require authorization.

The first sections in the HTTPCaller to fill out is the Request and Headers. The layout in the HTTPCaller is very simple because all the areas to fill are clearly labeled.

Please note this is an example call which can be found in the **Rest API POST** documentation.
First, paste the Request URL in. Then, change the HTTP Method to Post. Update the Headers.

**POST**

<table>
<thead>
<tr>
<th>Headers:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accept:</strong> application/json</td>
</tr>
<tr>
<td><strong>Authorization:</strong> fmetoken token=&lt;yourToken&gt;</td>
</tr>
</tbody>
</table>

When you insert the body section of the call find **Upload Body** and click on the drop-down menu. Click on Open Text Editor and paste the upload body into the call.

```json
{
  "publishedParameters": [
    {
      "name": "MAXY",
      "value": "42"
    },
    {
      "name": "THEMES",
      "value": [
        "airports",
        "cenart"
      ]
    }
  ]
}
```

Next, find the **Content Type** section and specify JSON (application/json). The body should look like this:

### 3) Check the Output From the HTTPCaller

**NEW to FME Workbench 2018**

*In this exercise, we are using Feature Caching. Feature Caching is a tool that allows a user to inspect the output of every transformer in their workspace after it has run. It also allows for partial runs of a workspace. This means you do not have to start at the beginning of a workspace and run to the end. You may choose where the run begins and ends. For more information on feature caching, please read the Safe Blog post on Feature Caching*

Attach a Logger Transformer to the Output port on the HTTPCaller and run the workspace with Feature Caching turned on. To do this click Run in the top toolbar and select Run with Feature Caching.
After, the workspace has completed, select the magnifying glass to inspect the output of our last call.

This will open up the FME Data Inspector. We can view the response_body of the last call to the FME Server. This will contain the job id of the last call.

**Ricky RESTless says...**

*While the FME Server REST API does not limit the number of calls you can make. Other APIs may set a limit on the number of calls you can make for free. Feature Caching is a super handy tool in this case. You may inspect the output and adjust your workspace without making additional calls to the server.*

4) Add a JSONFlattener to the canvas

Now you can add the JSONFlattener. The JSONFlattener allows you to select a part of the JSON to expose, in our case we should expose the id, so we can use it for the next call.

The parameters should look like this:

**JSON Document:** _response_body

**Recursively Flatten Objects/Arrays:** yes

**Attributes to Expose:** id

**List Attribute for JSON Parsing Errors** _json_error
5) Select the JSONFlattener and Click Run To This

We want to review the JSONFlattener and ensure that it is producing an attribute with just the Job ID.

Once, we rerun the workspace, we can select the Data Inspector symbol to review the output.

Click the magnifying glass to review the results

<table>
<thead>
<tr>
<th>_creation_instance</th>
<th>_response_body</th>
<th>http_status_code</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{&quot;id&quot;:8}</td>
<td>202</td>
<td>8</td>
</tr>
</tbody>
</table>

Here we can see that the new attribute was successfully created

Get Job Status Overview
The goal is this next section is to send a call out to the server to check if the previous job was successful. If the call is successful, then we can run a new job. However, this workspace runs so quickly that the FME Server would not have time to process the previous job before the new call is made. So in this section, we will be creating a custom transformer that will continually get the job status until the job has been completed.

6) Add an HTTPCaller

We will add a **HTTPCaller** and attach it to the JSONFlattener. It is easiest to copy the original HTTPCaller and then modify the parameters. In the HTTPCaller, click on the drop-down menu next to the Request URL and click open Text Editor. Paste the request and for the id double-click on this icon ♦️ id from the FME Feature Attributes side panel. Your call should look like this:

| GET | http://<yourServerHost>/fmerest/v3/transformations/jobs/id/@Value(id) |

**Headers:**

- **Accept:** application/json
- **Authorization:** fmetoken token=<yourServerHost>

Now for the response in the HTTPCaller, we are going to change the Response Body Attribute to job_status. So your HTTPCaller should look like this:
7) Check the Output From the HTTPCaller

Now, right click on the HTTPCaller_2 and select Run to This. From here we can click on the magnifying glass to view the response in the FME Data Inspector.

As you can see from the above image, the response is quite long in its raw form.

However, at the very end, we can see the status. Here you should see "status": "PULLED."

So know we need to extract that section of the response to test whether the call was successful. For that, we need another JSONFlattener.
8) Add a JSONFlattener

Back in FME Workbench, add another JSONFlattener. So, for the input parameters under JSON Document, we are going to select job_status. Then under attributes to expose write status.

![JSONFlattener Parameters]

9) Add a Tester

Now we are going to add a Tester to determine if the workspace is pulled or run. If the status is "PULLED" then the job has not been run yet. So for the job to be completed status must not equal PULLED.

The Tester should be named Tester_JobNotPulled. By changing the name of a transformer it allows a user to quickly understand what the Tester is testing for.

- The left value should be the Attribute Value status.
- The operator is !=. This means not equal.
- The right value is PULLED.

The Tester is testing if the job status is not pulled. If it is pulled it will be sent to the Failed port. If it is not pulled it will go to the Passed port.

The tester should be set up like this:

![Tester Parameters]

If the status is PULLED, then the job has not been run, and we need to rerun the call to check the status. To do this, we need to make a custom transformer with a looper.
10) Create a custom transformer

Select HTTPCaller_2, JSONFlattener_2, and JobNotPulled, right click and select Create Custom Transformer.

Enter in the following details:

- **Name**: JobStatusTester
- **Category**: Web
- **Overview**: This transformer will continue to check if the job has been passed from the PULLED stage.

Click OK. The custom transformer should look like this:

11) Add a Looper Transformer

Right click on the workspace and select Insert Transformer Loop. You will receive a dialog that says: Select Input to Loop to. The default will say Loop to: HTTPCaller_2_Input.

This means that is will loop the HTTPCaller_2 which will check the job id again. Connect this loop to the JobNotPulled failed output.

Run the workspace!
Once we have run the workspace we should see something like this:

![Diagram of workspace](image)

The custom transformer has to run 57 times before it passes the last tester. However, the total translation is only 3.8 seconds.

TIP

This loop is checking in on the FME Server as fast as possible. While the transformer ran 57 times, if this was a slower job it could hit the FME Server thousands of times. This is where the Decelerator could come in handy. The Decelerator would slow down the custom transformer so it will make less calls to the FME Server.

12) Add an Output to the Custom Transformer.

Right-click and select Insert Transformer Output and attach it to Tester_JobNotPulled passed. Now, we can switch back to the main canvas.

13) Add a Tester

Now we are going to add a Tester to determine if the workspace was successfully run.

The Tester should be named Tester_Successfully_Run. By changing the name of a transformer it allows a user to quickly understand what the Tester is testing for.

- The **left value** should be the Attribute Value status.
- The **operator** is `=`
- The **right value** is SUCCESS.

This testing if the status of the job is equal to success.

The tester should be set up like this:

![Tester Parameters](image)

If the status is SUCCESS, then the job was successfully run on the FME Server, and the next workspace will be triggered.

*New Job is Run*

14) Add another HTTPCaller

Finally, use a third HTTPCaller to run a new job synchronously. To do this use the same call that we had previously used, this is the Transact Call.

| POST | http://<yourServerHost>/fmerest/v3(transformations/transact/Samples/austinDownload.fmw) |
Headers:

- **Content-Type**: application/json
- **Accept**: application/json
- **Authorization**: fmetoken token= <yourServerHost>

Body:

```json
{
    "publishedParameters": [
        {
            "name": "MAXY",
            "value": "42"
        },
        {
            "name": "THEMES",
            "value": ["airports", "cenart"
        ]
    }
}
```

While this is not the most practical example because we are running the same workspace twice, but in two different ways, it demonstrates how FME can be used. It demonstrates the capabilities of running one workspace and using the response to determine the next action.

The final workspace should look like this:

---

**CONGRATULATIONS**

By completing this exercise you have learned how to:

- Use the HTTPCaller transformer to use the FME Server REST API in a workspace
- Create a custom transformer that continually checks the FME Server to see if a job has been completed
- Use the FME Server REST API to run multiple jobs in a row
Chapter 5- Web Services, Direct URLs, and the REST API

A Direct URL looks similar to the REST API Request URL; however, it exists outside of the REST API. To find an example of the Direct URL log into your server and located a workspace.

Open the advanced tab and scroll down until you find the Direct URL example:

The authentication process of the Direct URL is different from the REST API. With the Direct URL you may add a token to the end of the URL like this:

http://<yourServerHost>/fmejobsubmitter/<yourRepository>/<yourWorkspace>?<yourWorkspaceParameters>/token=<tokenValue>

Alternatively, the Direct URL will default to the permissions set on the guest user account. If your guest account has no permissions to run workspaces once you enter the Direct URL you may be asked to log in. If the guest account has permission, you may run the Direct URL without logging in.

Once you click the link, your job will run, and you will receive the following notification.

Like the REST API, the Direct URL can run a job synchronously and asynchronously. The job will automatically run synchronously. To run the job asynchronously enter an email in the section that says Email results to. By doing this the Direct URL will automatically be updated to run asynchronously. Once the job has completed an email will be sent.
Just like in the REST API there are parameters you may change in the Direct URL.

The following table was copied from the following manual on the Data Download Service. For full documentation see the Data Download Service documentation.

**Direct URL Request Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opt_responseformat</td>
<td>xml or json&lt;/br&gt;Default: xml</td>
<td>The language of the response. The text must be in lowercase</td>
</tr>
<tr>
<td>opt_geturl</td>
<td>The URL to a&lt;/br&gt;dataset</td>
<td>The URL of the source dataset to be used for transformation</td>
</tr>
<tr>
<td>opt_showresult</td>
<td>true or false</td>
<td>Whether the XML/JSON responses include the FME transformation result. The default value is true if this parameter is not present.</td>
</tr>
<tr>
<td>opt_servicemode</td>
<td>sync or async or schedule</td>
<td>Toggles between synchronous and asynchronous modes of the service. When jobs are submitted asynchronously (async), the response of submission success or failure is returned immediately. When set to synchronous (sync), the response is not returned until the job completes. Or, schedules a data download request to run at a specified start time (one-time only). If a schedule is specified, see below for additional parameters.</td>
</tr>
<tr>
<td>opt_requesteremail</td>
<td>Comma separated email addresses</td>
<td>Addresses to which the notification e-mail messages are sent.</td>
</tr>
</tbody>
</table>

The Direct URL and REST API can be combined together in an application to utilize the best components of both. An example is in the Easy Translator demo.

**Easy Translator**

Upload or use the browse tab to select any file-based format to translate data immediately into the required format and coordinate system.

Files are uploaded with the REST API; however, the job is run using a Direct URL.

This is a section of code for the dataUpload function from the FME REST API JavaScript Library. Please note it is not the full function but should provide some context to how the calls look like in a JavaScript function.

```javascript
dataUpload : function(repository, workspace, files, jsid, callback) {
    var url = buildURL('{{svr}}/fmedataupload/' + repository + '/' + workspace);
    var token = getConfig('token');
    url += '?token=' + token;
    ajax(url, callback, 'POST', params);
}
```

The URL is built with the with the call we see in the REST API, then the token is added to the end of the URL. The URL is submitted with the POST method.
Luckily, this call and almost all of the calls within the FME Server REST API exist within the REST API JavaScript library, so the user does not have to write the functions. A user can successfully create an application by simply calling the functions already created.

The Direct URL used in a JavaScript function would look like this:

```javascript
var submitUrl = BuildForm.host + '/fmedatadownload/' + BuildForm.repository + '/' + BuildForm.workspaceName + '?' + SourceDataset_GENERIC + ' = files;
submitUrl = submitUrl + ' &SourceFormat=' + sourceFormat;
submitUrl = submitUrl + ' &DestinationFormat=' + destFormat;
submitUrl = submitUrl + ' &COORDSYS_Dest=' + outputCoordSys + ' &opt_responseformat=json';
```

Here the Direct URL is built, then the user would click the DirectURL to activate it.
At the beginning of the tutorial, we discussed a web application that allows a user to select an area in Vancouver and receive data back from the Server with public transit information. We now have all the necessary information to use the query as a call to the FME Server. The URL in this example is a direct URL.

**Setting up the Query**

1) **Open the URL**


2) **Draw a polygon**

Click Draw Polygon, and then draw a polygon within the Vancouver area. Double click to end your drawing and close your polygon.

3) **Select Parameters**

On **Layers to Download** select Bus Routes and Bus Stops

**Output Coordinate System** should be set to WGS84 Lat/Longs [LL-WGS84]

**Output Format** should be set to Adobe Geospatial PDF

4) **Copy the query from the from the web application.**

Highlight the query that was automatically generated and copy it. If you are having issues you may copy the query down below:

```
```

Using the Query in Postman

5) Paste the URL into Postman
Paste the full URL into Postman, the query belongs in the top bar that says "Enter request URL". This call will not require any header or body as all the information needed is in the URL. Set the HTTP Method to GET.

6) Submit the call
Click send!

Understanding the Response

7) Review the Response in the bottom of Postman
This call is pretty advanced so you will receive a big chunk of JSON. This JSON would be interpreted by the Web Application and it would send back the correct data to the user.

8) Review the Headers
You can explore the headers you receive back from the call. By hovering over the bold text Postman provides a description of the response header.

The following shows the headers that this call returns.
<table>
<thead>
<tr>
<th>Body</th>
<th>Cookies</th>
<th>Headers (8)</th>
<th>Test Results</th>
</tr>
</thead>
</table>

**Connection** → keep-alive

**Content-Encoding** → gzip

**Content-Type** → application/json; charset=UTF-8

**Date** → Fri, 02 Mar 2018 19:42:24 GMT

**Server** → nginx/1.10.3 (Ubuntu)

**Strict-Transport-Security** → max-age=15768000

**Transfer-Encoding** → chunked

**Vary** → Accept-Encoding

By hovering over the header it will provide a description of the response header.
Generating Code Snippets

Postman has the ability to generate code snippets from previous requests. These snippets can be useful when developing your own web application.

9) Generate Code Snippets

On the page where you submitted your request click on the code button by the cookies.

There are various languages that Postman can generate, but for this shows JavaScript.
CONGRATULATIONS

By completing this exercise you have learned how to:

- Submit a Direct URL using Postman
- Generate code snippets using Postman
Chapter 6- Server Admin Tasks

Most server administration tasks can be handled through the REST API. In this chapter there are exercises for the most commonly used calls. This chapter focuses on working with resources. It gives an overview of how to upload a resource, list resources available, and download a resource. These calls are very important if you would like to create web applications using the REST API.
Exercise 10  Working with Resources

<table>
<thead>
<tr>
<th>Data</th>
<th>C:\FMEData2018\Data\CellSignals\CellSignal.csv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>To demonstrate how to upload a file and download a file using the REST API.</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>The calls for managing resources</td>
</tr>
</tbody>
</table>

Through the REST API, you will have access to the Resources available in the FME Server. You can list files, upload files, and delete files all using the REST API. This is helpful in a web application if you would like to direct data to a certain area in the FME Server.

Ricky RESTless says...

Most file management can be handled through the REST API. This means in a web application can have users upload files to the server, run a translation, and then return a file to the user. These calls are also in the FME JavaScript API and will be used later in the course.

Uploading a File

This call will upload a file to the data folder in your resources folder. The filename is specified in the Content-Disposition parameter. If you wanted to specify a directory in the data folder simply write the folder name after filesys. Uploading a file could be useful in a web application if it is requesting data from a user and inputting that data into a workspace.

1) Enter the following URL and Headers into Postman

In Postman click the plus symbol to open a new tab.

<table>
<thead>
<tr>
<th>POST</th>
<th>http://&lt;yourServerHost&gt;/fmerest/v3/resources/connections/FME_SHAREDRESOURCE_DATA/filesys/</th>
</tr>
</thead>
</table>

Headers:

- **Content-Type**: application/octet-stream
- **Accept**: application/json
- **Content-Disposition**: attachment; filename = "uploadingfiles.csv"
- **Authorization**: fmetoken token = <yourToken>

2) Upload a File in Postman

To upload a file in Postman. Switch to the Body tab. Select binary. The press Choose Files and navigate to

C:\FMEData2018\Data\CellSignals\CellSignal.csv
Click Send! Then, review the response

Optional Query String Parameters

Query string parameters are added at the end of the URL. If you wished to include these it the URL would look like this:

http://<yourServerHost>/fmerest/v3/resources/connections/FME_SHAREDRESOURCE_DATA/filesys?createDirectories=true&overwrite=false

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createDirectories</td>
<td>True</td>
<td><strong>false</strong></td>
</tr>
<tr>
<td>overwrite</td>
<td>True</td>
<td><strong>false</strong></td>
</tr>
</tbody>
</table>

Listing Files

This call will list the files in a resource. Listing files in a resource could be used in either a workspace or a web application. A possible use case could be creating a workspace to list all files in a resource and create an excel spreadsheet that is organized by the date uploaded.

3) Enter in the Following URL and Headers in Postman

<table>
<thead>
<tr>
<th>Headers:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Accept</strong>: application/json</td>
</tr>
<tr>
<td><strong>Authorization</strong>: fmetoken token=&lt;yourToken&gt;</td>
</tr>
</tbody>
</table>

4) View the Response in Postman
6.1 Exercise 10- Working with Resources

**Downloading a File**

The downloading file call is particularly useful in a web application. This can be used to return file to a user. This call can also be used in a workspace to download text files.

This call downloads a file from a resource connection either as an inline resource or as an attachment. The ‘disposition’ parameter is only relevant for some REST clients. In particular, Internet Explorer behaves poorly without an attachment disposition header. This web page will only display the response body instead of downloading the file, so REST clients should inspect the Content-Disposition header and handle the download.

This call will look into the Resources/Data directory to find uploadingfiles.csv.

**5) Enter in the Following URL and Headers in Postman**

headers:
- **Accept**: application/octet-stream
- **Authorization**: fmetoken token=<yourToken>
- **Content-Type**: application/x-www-form-urlencoded

**6) Enter in the body of the call**

To enter in the body of the call. Click the body tab, check raw and then paste the contents below.

```json
accept=contents&disposition=inline
```

Hit Send! Then, review the response in Postman
### Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disposition</td>
<td>inline/attachment</td>
<td>Inline will display how to retrieve the resource using the 'Content-Disposition' header. This parameter is only relevant for some REST clients (primarily web browsers). If no disposition is selected, then a header is not set.</td>
</tr>
<tr>
<td>Path</td>
<td>Path, relative to the resource connection (for example, uploading.csv)</td>
<td></td>
</tr>
<tr>
<td>Resource</td>
<td>Name of a resource connection. Can contain only alphanumeric characters. (for example, FME_SHAREDRESOURCE_DATA)</td>
<td></td>
</tr>
</tbody>
</table>

---

**CONGRATULATIONS**

*By completing this exercise you have learned how to:*

- Upload a file using the REST API
- List files using the REST API
- Download files using the REST API*
Exercise 11 | Using a Shared Resource as a Workspace Output Location

<table>
<thead>
<tr>
<th>Data</th>
<th>C:\FMEData2018\Data\Planning\BusinessLicenses.xlsx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>To demonstrate how to upload a file and download a file using the REST API.</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>The calls for managing resources</td>
</tr>
<tr>
<td>Workspace</td>
<td>C:\FMEData2018\Resources\RESTAPI\Chapter6Exercise11.fmw</td>
</tr>
</tbody>
</table>

Ricky RESTless says...

This next exercise demonstrates a typical workflow if you were to create a web application that were to run a translation and download a file that is created from the translation. First, you would upload a workspace to your FME Server using Workbench. Then, the user would be instructed to upload a file, this should go to a Shared Resource folder within the FME Server. After the user uploads a file, the application would run the workspace which would produce a file to a shared resource location. Then, the user would be able to download the file.

If you wish to retrieve the result dataset from a workspace through the REST API, the best method is to write it to a shared resource location that is accessible to the client. In this example we will make a workspace, then run a call while changing the output to the Temporary Repository. We will use a very basic workspace to test this out.

Create the Workspace

1) Open FME Workbench and start a blank workspace.

2) Add a Microsoft Excel reader, with the BusinessLicenses File.

There are many ways to add a Reader to a workspace. We can add the Reader by clicking on the canvas and typing Excel. Once we have selected the Excel Reader, we click the ... button to navigate to the Dataset.

The Dataset is located here:

C:\FMEData2018\Data\Planning\BusinessLicenses.xlsx
3) Then add CSV writer to the workspace.

Next, we can add the CSV Writer. The CSV Writer can be added in the same way. Click the canvas and start to type CSV. Select the CSV writer.

Accepting the default parameters is okay.

![Add Writer](image)

Then, connect the Excel file to the CSV file.

4) Set the Destination Folder.

The next step is to identify the Published parameter of the destination dataset. Published Parameters are located in the Navigator panel of FME Workbench. It should be called DestDataset_CSV2. Right click on DestDataset_CSV2 and select Edit Value. Then using the drop-down arrow, expand the FME Server Parameter menu and select $(FME_SHAREDRESOURCE_TEMP).
When using calls through the REST API it is important to have the file being written to an area that your FME Server has access to. The temporary folder is a great place to write files to for testing because the files will be cleaned.

5) Set the Source Dataset

We will upload the data using the API later. For now change the source dataset folder to:

```
$(FME_SHAREDRESOURCE_DATA)/SharedResourceTest/BusinessLicenses.xlsx
```

The source dataset also has to be in a location that the FME Server has access to. We are specifying a folder here that will be created through the REST API.

6) Set the Source Dataset to a Private Parameter

The next step is to set the Source Dataset to a private parameter. This can be accomplished by going to the navigator panel and then right-clicking on the parameter. Then select "Convert to Private Parameter."

Converting Public Parameters to Private Parameters ensures that the call is easy to use. Since, the source of the dataset will not change it is a best practice to convert it to a private parameter.

7) Publish to FME Server

*If you have not set up the Server Connection the instructions can be found here*
First, click the publish to FME Server button. Then, select the FME Server Connection created previously and click Next. Now, set the Repository name to RESTTraining. Set the Workspace name to Chapter6Exercise11.fmw and click Next. Finally, select the Job Submitter Service and publish the Workspace.

Upload the Data Using the Resources call

8) Enter the following URL and Headers into Postman


Headers:

- **Content-Type**: application/octet-stream
- **Accept**: application/json
- **Content-Disposition**: attachment; filename="BusinessLicenses.xlsx"
- **Authorization**: fmetoken token=<<yourToken>>

9) Add the Body of the call in Postman

First, click on the body tab in Postman. Then, select Binary and click Choose Files. Find C:\FMEData2018\Data\Planning\BusinessLicenses.xlsx as the file to upload!

10) Click Send! Then, review the response

```
1
2  "date": "2018-06-08T16:25:07-07:00",
3  "path": "/SharedResourceTest/",
4  "size": 6550765,
5  "name": "BusinessLicenses.xlsx",
6  "type": "dir"
7  }
```
Run the Job

11) Enter the following URL and Headers into Postman

| POST | http://<yourServerHost>/fmerest/v3/transformations/submit/RESTTraining/Chapter6Exercise11.fmw |

Headers:
- **Accept**: application/json
- **Content-Type**: application/json
- **Authorization**: fmetoken token=

12) Add the Body of the call in Postman

First, click on the body tab in Postman. Then, click the raw button. Then, paste the following information.

```json
{
    "publishedParameters": [
        {
            "name": "DestDataset_CSV2",
            "value": "$(FME_SHAREDRESOURCE_TEMP)"
        }
    ]
}
```

13) Review the Response from the FME Server

```
{
    "id": 29
}
```
Please Note. The Job ID pictured may be different to what your machine returns.

Test the Job was Successful

14) Enter the Following call into Postman. Then, Click Send!
This call will run asynchronously so in return you will receive an id. To check that your call has been completed, use this call.

| GET | http://<yourServerHost>/fmerest/v3/transformations/jobs/id/<Jobid> |

Headers

- **Accept**: application/json
- **Authorization**: fmetoken token= <yourToken>

15) Review the Response to ensure the job was completed successfully

Download the Resulting File

16) Enter in the Request URL and Headers into Postman
Then you can download the file using the call below.

| POST | http://<yourServerHost>/fmerest/v3/resources/connections/FME_SHAREDRESOURCE_TEMP/download/Business_Licences.csv |

Headers:

- **Accept**: application/octet-stream
- **Authorization**: fmetoken token= <yourToken>
- **Content-Type**: application/x-www-form-urlencoded
17) Enter in the Body in Postman
Click the body tab in Postman. Then select raw. Then, paste the following information. Click Send!

```
accept=contents&disposition=inline
```

18) Review the Response in Postman!
While Postman can process the call, the CSV file cannot be downloaded. If you wanted to download a copy of the file you can do so through FME JavaScript API. We will walk through this process later in the course.

CONGRATULATIONS

By completing this exercise you have learned how to:
- Create a workspace compatible with the Submit call
- Upload a file to a shared resource using the REST API
- Run a submit call and test if it was successful
- Download files using the REST API
Chapter 7: Job Management

Job queues are a mechanism for sending jobs to specific FME Engines. The reasons for using job queues include:

- Controlling the priority of job requests.
- Sending a job to an FME Engine in close proximity to a data source.
- Sending a job to an FME Engine that supports a particular format.
- Reserving an FME Engine for a scheduled task.
- Reserving an FME Engine for quick jobs.

When you create a job queue, you assign one or more FME Engines to the queue. (You can assign an FME Engine to more than one queue.) Then, when you run a job, you can specify which queue and, therefore, which engines to handle the job. If you create multiple job queues with different priority levels, higher priority queues submit job requests before lower-priority queues, depending on timing.

By extension, you can also assign a repository to a queue. By default, all jobs are assigned to the queue of their respective workspace repository, unless another queue is specified for the job.

All engines and repositories must be assigned to a queue. If an engine or repository is not assigned to a queue explicitly, it is assigned to the Default queue.
Exercise 12  Creating a Job Queue

Data  None

Overall Goal  To demonstrate how the FME REST API can be used to manage job queues in FME Server

Demonstrates  How to create a queue

Workspace  austinDownload.fmw. Stored in the Samples Repository

NEW

Previously, job management was configured through engine tags. This has been deprecated and now we use job queues. However, the REST API endpoints are the same but manage queues now.

Create a Job Queue

The following call allows you to create a new queue for a job. The following tag created will be called High_Priority. Any jobs with this tag will be sent to FMETRAINING_Engine1. You may need to add a tag to multiple engines.

**FMETRAINING_Engine1 is specific to the training machines**

1) Enter the following URL and Headers into Postman

<table>
<thead>
<tr>
<th>POST</th>
<th>http://&lt;yourServerHost&gt;/fmerest/v3/transformations/jobroutes/tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Headers:</td>
<td></td>
</tr>
<tr>
<td>Content-Type:</td>
<td>application/x-www-form-urlencoded</td>
</tr>
<tr>
<td>Accept:</td>
<td>application/json</td>
</tr>
<tr>
<td>Authorization:</td>
<td>fmetoken token=&lt;yourServerHost&gt;</td>
</tr>
</tbody>
</table>

2) Switch to the Body tab. Click raw and paste the following information

```
engines=FMETRAINING_Engine1&name=High_priority&priority=1
```

```
Additional Parameters

Below are some additional parameters you may use for future calls.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Description of the tag</td>
</tr>
<tr>
<td>engines</td>
<td>Engine assignment for the tag</td>
</tr>
<tr>
<td>name</td>
<td>Required - Unique name of the tag to create</td>
</tr>
<tr>
<td>repositories</td>
<td>Repository assignments for the tag</td>
</tr>
<tr>
<td>priority</td>
<td>Priority for the tag. Priority values must be integers between 1 and 10.</td>
</tr>
</tbody>
</table>

3) Click Send and Review the Response from the FME Server.

You should receive a blank response with a status code of 201 Created.

Assigning an Engine to the Job Queue

4) Enter the Following URL and Headers into Postman

```
POST http://<yourServerHost>/fmerest/v3/transformations/jbroutes/tags/High_priority/engines

Headers:
- Content-Type: application/x-www-form-urlencoded
- Accept: application/json
- Authorization: fmetoken token= <yourServerHost>
```

5) Switch to the Body tab. Click raw and paste the following information:

```
engines=FMETRAINING_Engine1
```

6) Click Send and Review the Response from the FME Server.

Assigning a Repository to the Job Queue

In the next stage of this exercise, we are taking the job queue we just created and assigning it to a repository. This ensures that every time a job in the repository is run it is assigned to this queue.

7) Enter the Following URL and Headers into Postman

Request
POST  http://<yourServerHost>/fmerest/v3/transformations/jo
broutes/tags/High_priority/repositories

Headers:
- **Content-Type**: application/x-www-form-urlencoded
- **Accept**: application/json
- **Authorization**: fmetoken= <yourServerHost>

8) Switch to the Body tab. Click raw and paste the following information:

```
repositories=RESTTraining
```

Additional Parameters
Below are some additional parameters you may use for future calls.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repositories</td>
<td>Repository assignments for the queue</td>
</tr>
<tr>
<td>tag</td>
<td>Name of the job queue</td>
</tr>
</tbody>
</table>

9) Click Send and Review the Response from the FME Server.
You should receive the following response.

204- Success. The assigned repositories were assigned.
10) Check the FME Server to see if the Repository was successfully assigned

Go to your FME Server and Click Repositories.

To the right of the repository name, you should see a drop down list of potential queues. The High_Priority queue should be assigned.

Submit a Job and Include the Tag in the Request

When you submit a job, you may specify the queue in the request, under the TMDirectives. The TMDirectives specify how the job should be run by the server. Below is the full list of parameters.

**WARNING**

Please note that the priority has been deprecated and queues will be honored instead.

```
TMDirectives {
rtc (boolean, optional): Runs a job until it is explicitly canceled. The job will run again regardless of whether the job completed successfully, failed, or the server crashed or was shut down.,
ttc (integer, optional): Time (in seconds) elapsed for a running job before it's canceled. The minimum value is 1 second, values less than 1 second are ignored.,
ttl (integer, optional): Time to live in the job queue (in seconds),
description (string, optional): Description of the request,
priority (integer, optional): The priority of the job. Priority values must be integers between 1 and 200. If a request’s priority value is less than 1, greater than 200, or is not specified, then FME Server sets it to 100 .,
tag (string, optional): The job routing tag for the request
}
```

Below is an example of what the TMDirectives can look like:

```
"TMDirectives": {
  "rtc": false,
  "ttc": 60,
  "description": "This is my description",
  "tag": "linux",
  "ttl": 60
},
```

Now, we are going to use a Submit call and alter the parameters to include the job queue (also called a tag in the REST API) previously created.

11) Enter the Following URL and Headers into Postman

**POST**

```
http://<yourServerHost>/fmerest/v3/transformations/submit/Samples/austinDownload.fmw
```

**Headers:**

- **Content-Type:** application/json
- **Accept:** application/json
- **Authorization:** fmetoken token= <yourServerHost>
12) Switch to the Body tab. Click raw and paste the following information

```
{
    "publishedParameters": [
        {
            "name": "MAXY",
            "value": "42"
        },
        {
            "name": "THEMES",
            "value": [
                "airports",
                "cenart"
            ]
        }
    ],
    "TMDirectives": {
        "tag": "High_priority"
    }
}
```

13) Click Send and Review the Response from the FME Server.

```
{
    "id": 32
}
```
Please Note. The job ID that is returned by the Server will be different based on how many jobs the FME Server has completed.

Now we can check the job to see if the job ran successfully under the High_priority tag.

```
14) Enter the Following URL and Headers into Postman

| GET | http://<yourServerHost>/fmerest/v3/transformations/jobs/id/<yourJobID> |
```

Headers:
- **Accept**: application/json
- **Authorization**: fmetoken token=<yourToken>

15) Click Send and Review the Response from the FME Server

In the response, you can find the Engine that the call was submitted to and the parameters of the call.

```
{  
  "workspacePath": "\\Samples\austinDownload\austinDownload.fmw",
  "TMDirectives": {
    "rtc": false,
    "ttc": -1,
    "description": "",
    "tag": "High_priority",
    "priority": -1,
    "ttl": -1
  }
}
```

As you can see here, the job was submitted under the High_priority tag. However, the priority demonstrated was still -1. The tag will override this priority, so it was still submitted with a priority of 1, it is just not displayed in the FME Server.

**CONGRATULATIONS**

By completing this exercise you have learned how to:
- Create a job queue using the REST API
- Assign an engine to a job queue using the REST API
- Assigning a repository to a job queue using the REST API
- Run a job using the REST API and assign a queue within the call
7.2 Retrieve Job History

Pagination of results is useful for endpoints that return a collection of objects, especially when the collection has large amounts of results. To help users work with pages of results, we improved the response body to be the following structure:

- **items (array[Object]):** Items in this results page
- **limit (integer):** limits how many items are returned by this call. Could set it to 10 to only retrieve 10 jobs at a time.
- **offset (integer):** Offset of this results page
- **totalCount (integer):** Total amount of items available

It is possible to hit the max number of responses allowed by the REST API. Currently, there are only 1000 responses allowed in one request. If you are retrieving your entire job history it is likely that is will be over 1000 responses.

So, we can use the offset to gather 1000 responses at a time. The offset starts from the newest and subtracts from the offset number. Let’s say you have a busy server with 10,000 jobs and you wanted to retrieve the full job history. For the first call, you would want to set the limit to 1000 and the offset to 0. This will retrieve the newest 1000 jobs. The next call would have to set the limit to 1000 and the offset to 1000. This would retrieve the next newest 1000 jobs.

The next exercise demonstrates how you could retrieve job history and create an HTML report with the information.
Exercise 13  Create a Workspace to Retrieve Job History

<table>
<thead>
<tr>
<th>Data</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>To Create a Workspace that can retrieve the last 1000 jobs and display it in an HTML.</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>How to use the Get Jobs call in a workspace</td>
</tr>
<tr>
<td>Workspace</td>
<td>C:\FMEData2018\Resources\RESTAPI\Chapter7Exercise13.Complete</td>
</tr>
</tbody>
</table>

For this exercise, we are going to create a workspace that retrieves the job history from your FME Server and creates an HTML file with the results. The first workspace we will create retrieves the latest 1000 jobs from the FME Server. Then the results are formatted into a table with the results and beneath this, there will be a chart with the number of results by date.

**Use the HTTPCaller to Get Jobs and then Format the Attributes**

1) **Add a Creator**

In a blank FME Workspace add a Creator transformer. It is okay to accept the default parameters.

2) **Add the HTTPCaller**

Add the HTTPCaller and set the parameters as listed below:

Request URL: http://<yourServerHost>/fmerest/v3/transformations/jobs/completed  HTTP Method: GET

Next, fill in the Query String Parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>limit</td>
<td>1000</td>
</tr>
<tr>
<td>offset</td>
<td>0</td>
</tr>
</tbody>
</table>

Then, under Headers fill in the Accept Header.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>application/json</td>
</tr>
</tbody>
</table>
Check the box next to Use Authentication. Then select Basic as the Authentication mode and use the admin/admin username and password. You can also set the Authentication Method to Web Connection and then use the FME Server connection that was created in a previous exercise.

3) Run with Feature Caching

Now, we can save the workspace and then run with Feature Caching. Once, the workspace has run select the magnifying glass on the HTTPCaller. This will open up the FME Data Inspector.

Once the FME Data Inspector is open. Select the response_body. This response_body contains all the job obtained by the API.

We need to parse the response to get the information we want into attributes we can use. For this exercise, we are going to keep the id, timeSubmitted, and status. We want to find the id, timeSubmitted, and status in the JSON returned from the FME Server.

Close the FME Data Inspector.

4) Add a JSONFragmenter

We are going to use the JSONFragmenter to parse the attributes.
The JSON Attribute is the response body which is produced from the previous call.

JSON queries should start with `json` and then the category where the attributes are stored. This may take some trial and error at first.

For the JSON Query, we want to find all the responses in the `items` category therefore, we use this statement `json["items"][*]`

The code block below demonstrates a sample of the JSON that is returned by the call. By using the JSON Query: `json["items"][*]`, we can then use the Flattening to expose the attributes within the code we would like to use.

```json
{"offset":0,"limit":1000,"totalCount":5,"items":[{"request":{"publishedParameters":{"name":"CellSpacing","raw":"1"},"name":"FEATURE_TYPES","raw":"04_05_NO"},"name":"Compression","raw":"0"},"name":"FEATURE_TYPES_2","raw":"Private Other NonCity"},"name":"OutputCoordSys","raw":"BCALB-83"},"name":"Neighborhood","raw":"Mount Pleasant"],"name":"OutputFormat","raw":"JPEG"},
```

Next, we need to set up the Flattening Parameters in the `JSONFragmenter`.

Here, we enter the attributes we would like to keep from the query. You should keep the `id`, `timeSubmitted`, and `status`.

5) Add an Attribute Keeper
Next, add the AttributeKeeper. Attach the AttributeKeeper to the JSONFragmenter. Under parameters and Attributes to Keep: write timeSubmitted, id, status.

![AttributeKeeper Parameters](image)

Format Dates

![Format Dates](image)

6) Add the DateTimeConverter

Add a DateTimeConverter and attach it to the AttributeKeeper. Use the DateTimeConverter to format the dates into the %Y-%m-%d format. The datetime format that is returned from the FME Server can be difficult to read. By modifying the Output Format we can make our chart more visually appealing.

The parameters should be filled in as below:

- **Datatime Attributes**: timeSubmitted
- **Input Format**: Auto detect FME and ISO formats
- **Output Format**: %Y-%m-%d
- **Repair Overflow**: No
Create a Table with the Job History Results

If we compiled a table with the result currently, it would create a table. However, it would not have an order so we use the Sorter transformer.

7) Add the Sorter transformer

The Sorter transformer will be connected to the DateTimeConverter. Under the Sort By section list the following parameters:

- **Attribute**: id
- **Alpha/Num**: Numeric
- **Order**: Descending

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Alpha/Num</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Numeric</td>
<td>Descending</td>
</tr>
</tbody>
</table>

8) Add the HTMLReportGenerator

Add the HTMLReportGenerator and attach it to the Sorter transformer. In the HTML Report Generator, we can add custom HTML or a table with information gained from the FMEServer.
Under the Page Contents, add Custom HTML to the HTMLReportGenerator.

First, we want to add Custom HTML to specify the header on our HTML file.

```html
<h2><strong>Monthly Server Report</strong></h2>
```

If you had an introduction or any formatting, you could place it here as well.

Next, under page contents, add a Table. Specify the column contents and headings the table will use.

<table>
<thead>
<tr>
<th>Column Contents</th>
<th>Column Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Job ID</td>
</tr>
<tr>
<td>status</td>
<td>Job Status</td>
</tr>
<tr>
<td>timeSubmitted</td>
<td>Time Submitted</td>
</tr>
</tbody>
</table>

To view the table before it is created, click the Preview in Browser button.

---

### Create a Bar Chart with Jobs by Date

9) Add the Aggregator transformer

Add the Aggregator transformer and attach it to the DateTimeConverter.
We need to aggregate all jobs by date to create the bar chart so we need the Aggregator.

Here, we are grouping the results by timeSubmitted which is the date the attributes were submitted. Then under Count Attribute, we create a new Attribute named DateCount. Ensure the Accumulation Mode is set to Merge Incoming Attributes.

10) Attach the HTMLReportGenerator

Next, we attach another HTMLReportGenerator. Under Page Contents select Chart (Bar). Then under Content Settings fill in the X and Y labels. The X axis label is Date Job Submitted. The X Tick Label Attribute is timeSubmitted. The Y Axis Label is Number of Jobs per Date. The Data Attribute is DateCount and the Color is 0,0,255. The parameters are set up below:
Format the Layout and Write to HTML

11) Add the HTMLLayouter
Attach both the HTMLReportGenerators to the HTMLLayouter.

12) Add an HTML Writer
Finally, add an HTML Writer. It is okay to accept the default values. Save the file to:
C:\FMEData2018\Output\Training\JobHistory.html
13) View the results!

Run the workspace and view the results.

This example demonstrates an FME Server with many completed jobs, if you are using a training computer this graph will be more simplified.

Monthly Server Report

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Job Status</th>
<th>Time Submitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
<td>SUCCESS</td>
<td>2018-05-12</td>
</tr>
<tr>
<td>84</td>
<td>SUCCESS</td>
<td>2018-05-12</td>
</tr>
<tr>
<td>83</td>
<td>SUCCESS</td>
<td>2018-05-12</td>
</tr>
<tr>
<td>82</td>
<td>SUCCESS</td>
<td>2018-05-12</td>
</tr>
<tr>
<td>81</td>
<td>SUCCESS</td>
<td>2018-05-12</td>
</tr>
<tr>
<td>80</td>
<td>SUCCESS</td>
<td>2018-05-12</td>
</tr>
</tbody>
</table>

CONGRATULATIONS

By completing this exercise you have learned how to:
- Use a workspace to call the FME Server REST API to retrieve jobs
- Review the response from the FME Server
- Use the HTMLReportGenerator to create a report of the results
Chapter 8- Developing with the FME Server REST API

This chapter takes a look into developing with the FME Server REST API. It will provide an overview of the best practices of creating a web application, how to host an application, and how to set up an application.
8.1 Best Practices

Setting Permissions

The ability of a user to access data on your FME Server is dependent on the permissions granted to the user. When the application uses functions through the JavaScript API, like Data Download or Data Upload, the user first has to initialize the application with a token. Each token is connected to a user. It is a best practice to create a new user for each application and use that token.

It is important to note, that if your web application uses a Direct URL, then the permissions are automatically set to the guest account. The repository that holds the workspace has to have guest permissions.

To give the repository guest permissions, you will have to access your FME Server. First, look on the left-hand panel of your FME Server. Find Security and then look for Users.

Once on the Users page click guest.

Scroll down to the permissions and expand the repositories tab. Then check the read and run permissions.

Now the Direct URL will work.

Using the Namespace Variable For the Data Upload Function

The Data Upload function is displayed in Exercise 18. When a user uses the web application, the permissions go through the account that is authorized at the beginning of the application. Each user will be anonymous. If two users upload the same file, the files will end up in the same location and be overwritten. To prevent this, each upload is given a namespace which is a randomly generated key. When using the Data Upload function, it is best to use the getSession function first. This will be demonstrated later on in the course.

Use Direct URLs when needed

Direct URLs are great, particularly for Data Streaming. Direct URLs can be used to stream data directly back to the web application. This enables a user to view up to date information that has just been processed by the FME Server.

Consistent Naming Conventions

Web applications may contain many resources and workspaces. Keeping workspaces organized on the FME Server is important. Each workspace or other items should be named so that a user can quickly tell what workflow it belongs to.
Additionally, when working with a large number of workspaces, it is important to publish workspaces with descriptions. These descriptions will help you figure out the purpose of the workspace quickly and easily. You can also view the workspace in the Workspace Viewer for a quick overview.

**Create a Project**

Projects are a way to bundle multiple resources on the FME Server together in one area. This is a great way to keep resources together if you ever migrate servers. If you have a web application that requires, a specific repository and a resource folder, you can combine these in a project. This project can be shared and moved to a different FME Server. We are using a project in this course to demonstrate how easy it is to use a project to migrate workspaces and resources to another FME Server.

**Further Code Resources**

If you are interested in increasing your knowledge of CSS, JavaScript, and HTML there are many online resources. [W3Schools](https://www.w3schools.com) and [Mozilla Development Networks](https://developer.mozilla.org) have some excellent guides.
8.2 Hosting Web Applications

There are several ways to complete this training and view your web application.

If you don't have your own web server, you can use one of the following options for this course.

**Local Host Server**

This is a good method if you do not have an outward facing server. This is a simple method to create a page and host it locally. Please note, that if your server is only locally hosted you will not be able to use the data streaming to Google Maps and ESRI functions. Exercise 14 is a step by step guide on how to set up a local host server on a training computer. The instructions below are meant to be more of a universal guide.

1. **Install Python**

Go to the following website [https://www.python.org/](https://www.python.org/) and find the download link to download the newest version of python and install on your computer.

2. **Locate Terminal (Mac/ Linux) or the Command Prompt(Windows)**

Using the commands to navigate through the file folders, locate where your files are stored.

Here are the useful commands for this:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd[folder]</td>
<td>Change directory e.g. cd Documents</td>
</tr>
<tr>
<td>cd</td>
<td>Home directory</td>
</tr>
<tr>
<td>ls (Mac) or dir (Windows)</td>
<td>List Files</td>
</tr>
</tbody>
</table>

3. **Type in the Python command**

*For Python 2.7*

**WARNING**

Python 2.7 is being deprecated from FME products. If you are using Python 2.7 in your FME Workspaces please read Python 2.7 Deprecation.

```python
python -m "SimpleHTTPServer" 8080
```

Please note, this may not work if you paste the command in due to formatting. If you type the command, it will work.

*For Python 3.6*

```python
python -m http.server
```

4. **Open up a browser and type in:** [http://localhost:8080/](http://localhost:8080/)

5. **Select your file (optional)**

If you did not directly navigate to the file, you might have to select it using the prompts.

**Google Apps Script**

This is a good method if you have an outward facing server but no pre existing web directory structure. GoogleAppsScript is free to use but does require a Google login.

Simply, go to [https://script.google.com/home](https://script.google.com/home) and click New Script to create your web application.
Delete what is currently in Code.gs and paste:

```javascript
function doGet() {
    return HtmlService.createHtmlOutputFromFile('index');
}
```

Then, create a new HTML file named index.html and start working.

After you have created a webpage, you can view it through the application. Click Publish, then Deploy as Web App.

---

**How Safe Software Deploys Web Applications**

While you won’t be deploying web applications in the course, you may be curious how Safe Software deploys web demonstrations.

All of Safe Software’s Web Demos are hosted in the following way. First, the application folders are posted to GitHub. GitHub has the advantage of versioning. So, once an edit to the code has been made it will display who made the changes and when they were made.

Safe, then uses Deploy, which is notified when the code on GitHub is updated. The application is then pushed to the AWS S3 Web Server.

For more detailed information on this, please visit the Deploying a Web Application on Top of FME Cloud article.
Setting Up the Web Directory

<table>
<thead>
<tr>
<th><strong>Data</strong></th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overall Goal</strong></td>
<td>To properly configure a desktop to host web applications locally</td>
</tr>
<tr>
<td><strong>Demonstrates</strong></td>
<td>How to use Python to properly set up a local server</td>
</tr>
</tbody>
</table>

When testing web pages, we need to create a simple local HTTP server to access its functionality. This simple server will be closed once the computer is shut down. These are the specific instructions if you are using a training computer. The http server is a command in Python 3.x. Additionally, you may wish to navigate to a different directory based on where your training data is stored.

**Optional Step**

If you are not using a training computer you may have to download Python. To check if you have Python installed on your computer use the search function and search for Python.

If Python3.x is installed on your computer proceed to Step 1.
1) Open the Command Prompt

On the training computer the Command Prompt can be found by clicking the Start Menu, then finding the Most used category.

2) In the command prompt type:

```
cd /n```

This directs you to the C drive.

3) In the command prompt type:

```
cd FMEData2018/Resources/RESTAPI
```

4) In the command prompt type:

```
python -m http.server
```

This is the command to start the server which will be hosted locally.
5) Open up browser and type in: http://localhost:8000/

Directory listing for /

- .DS_Store
- Chapter4Exercise7/
- Chapter4Exercise8/
- Chapter6Exercise11/
- Chapter6Exercise12/
- Chapter7Exercise12/
- myFifthApp/
- MyFirstApp/
- myFourthApp/
- MySecondApp/
- MyThirdApp/
- readme.md

6) Navigate to My First App

Select myFirstApp. Then, select myFirstApp.html. This is a fully functioning web application that we will be creating in the next exercise.

Repository: Samples
Workspace: jaulinDownload.fmw

Generate Full Form

Layers to Download
- airports
- cenart
- railroad
- streetcl

Output Coordinate System: WGS84 datum. Latitude-Longitude. Degrees [EPSG #4326]

Output Format: Esri Shape

Minimum X: -100
Minimum Y: 25
Maximum X: 99
Maximum Y: 35

Search Envelope Coordinate System: LL84

Run Data Download

Download Result

CONGRATULATIONS

By completing this exercise you have learned how to:
- Use Python to create a simple server
Chapter 9- Building Custom Web Applications

This chapter is intended to teach users how to create custom applications to match their needs.
9.1 Creating a Dynamically Populated Form

FME has built a JavaScript API that allows a user to access their FME Server when developing a web application. In this exercise, you will create a basic form, in which you will select a workspace to run. Once you hit enter, the form will appear with the parameters populated. Once you hit return, there will be a link to download the requested file.
Exercise 15  Dynamically Generate a Form Using the REST API

<table>
<thead>
<tr>
<th>Data</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>To display the capabilities of the FME Server JavaScript API. This exercise allows you to dynamically generate a form based on the parameters in a workspace and have a user fill in the parameters and run the workspace.</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>How to use getWorkspaceParameters, generateFormItems, and runDataDownload functions.</td>
</tr>
<tr>
<td>Starting HTML</td>
<td>C:\FMEData2018\Resources\RESTAPI\myFirstApp\myFirstAppStart.html</td>
</tr>
<tr>
<td>Completed HTML</td>
<td>C:\FMEData2018\Resources\RESTAPI\myFirstApp\myFirstApp.html</td>
</tr>
</tbody>
</table>

In this next exercise, we will dynamically create a form from a workspace that the end user will select. Once, the user has selected a workspace a form will be generated with the user parameters from the workspace. The user can then select the parameters they'd like to use and run the workspace. Once, the workspace has run a link will appear on the page with a download link for the user. This code could be used if you wanted to create an application for clients to access data without going through the FME Server UI.

**WARNING**

*For this exercise to work properly you will need to have completed Exercise 14 or have an existing web server for testing.*

1) Open the myFirstAppStart.html

Go to C:\FMEData2018\Resources\RESTAPI\myFirstApp\myFirstAppStart.html. Then, right click on the file myFirstAppStart.html and select Edit with Notepad++. The file contains an almost complete web application but is missing some key information. Follow the steps below to get this app up and running. In the HTML document, any information that requires updating is indicated by //.

2) Add the FME Server JavaScript API Link

In the head section of the page, we link to the external stylesheet and the FME Server JavaScript API.

On line 11 delete the comment <!---Connect to the JavaScript API --->
and paste "https://api.fmeserver.com/js/v1.2/FMEServer.js"

The link to the JavaScript API must be included in all applications that use the JavaScript API. It's also a great resource to use if you would like to learn more about the JavaScript API because it lists all the available functions.
Next, we will build the body of the HTML. We need to create a form that will display the initial information needed to locate the workspace.

3) Update the Example Form

Scroll down to line 21 and find the exampleForm.

In the form, we will create a label and an input. The label is entitled Repository: which will appear on the page. Then as with the form, we need to give the input an id so it can be called upon later. The input type is text. **We need to edit the value to be "Samples".**

We will repeat the same input process for the workspace. **The value should be updated to "austinDownload.fmw".**

In the next line of code before the end of the form we need to create an input type for the next section of code. Underneath the two labels we would like to display a button.
Delete "// Input Type" and then replace it with "button".

```
<input type="button" onclick="processForm();" value="Generate Full Form" />
```

Having this button will activate the "processForm();" function. The button will have the value or label of Generate Full Form.

4) Update the Form Id

After the first form has been activated, we are going to use this blank form to hold the parameters dynamically generated by the following JavaScript.

Locate line 36 and 37, and replace "// Form id" with "output-form".

```
35
36
37
38

<form id="// Form id"></form>
```

The form id should now read "output-form".

```
35
36
37
38

<form id="output-form"></form>
```

If we looked at the website now, it would display a form. However, the form would not be functional. We can create a website using HTML code, however the website will not be able to do much without any JavaScript. JavaScript is needed to have a functional and interactive website.

5) Update the FME Server URL and Token

Go to the start of the JavaScript section (line 39).

```
<script type="text/javascript">
FMEonload = function() {
FMEServer = {url: "https://example-software-download.com", //Update with your Server hostname - on FME training machines this is https://fmetraininghost server: "https://example-software-download.com", //Update with your Server hostname namespace: "austin\Download\fmw", //Update with your Server namespace}
```

If your FME Server is externally facing you can use your Server hostname in this section. The FME training machines are also externally facing so will be ok for this exercise. If your web server (for the web applications) and FME Server are within the same domain or same machine you can use the internal IP address or hostname to access FME Server.

However, please note that Google Maps and Esri functionality will not be available if the FME Server is not accessible externally.
If you are not using a training machine and do not know your IP address, this will be the first result returned if you Google "my IP address".

Use this information to update the server and token section to your own. If you wish to locate your token. Go to this link:

```
http://<yourServerHost>/fmetoken/
```

If you have created a restapi user earlier in the course please use this token.

6) Update the processForm function

Next, we can view the processForm function this is located on line number 47. This function uses the FME Server JavaScript API to get the Workspace Parameters using the repository and workspace values entered by the user. We use the function `getWorkspaceParameters` to get the workspace parameters created in the call.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repository</td>
<td>String</td>
<td>The name of the repository that houses the workspace.</td>
</tr>
<tr>
<td>workspace</td>
<td>String</td>
<td>The workspace name.</td>
</tr>
<tr>
<td>callback</td>
<td>Method</td>
<td>The signature of the method or the function to run after the server has processed the request.</td>
</tr>
</tbody>
</table>

This function requires three parameters, the repository, workspace, and callback.

The repository and workspace will be entered by the user. To get the values entered by the user we need to create a variable and get the element by the id we gave it earlier.

```
function processForm() {
  var repository = document.getElementById("Find the repository ID and update it here").value;
  var workspace = document.getElementById("Find the workspace ID and update it here").value;

  // Get the workspace parameters from FME Server
  FMEServer.getWorkspaceParameters(repository, workspace, generateform);
}
```

We need to change "Find the repository ID and update it here” to "repository-name".
We also need to update "Find the workspace ID and Update it here" to "workspace-name".

```javascript
function processForm() {
    var repository = document.getElementById("repository-name").value;
    var workspace = document.getElementById("workspace-name").value;
    
    // Get the workspace parameters from FME Server
    FMEServer.getWorkspaceParameters(repository, workspace, generateForm); 
}
```

7) Modify the generateForm function to produce a form

Go to the generateForm function, this is located on line 55. The next function is the callback function. A callback function is specified in the previous function. It takes the json returned from the FME Server and uses it.

In our case we have the processForm function and which gets the workspace parameters. With this information we can use the generateFormItems function to create a form for the user to fill in and submit.

Currently, the generateFormItems parameters are empty so we will need to fill them in.

```javascript
55 function generateForm(json) {
56     // Clear the output form
57     document.getElementById("output-form").innerHTML = ";";
58     // Fill in the parameters here
59     FMEServer.generateFormItems("output-form", json);
60 }
```

The first part of the code will clear the output form if there is anything in there previously. Then the generateFormItems function is called but currently there are no parameters in the function so it will not work.

The id is the form container to put the elements into. In our case this is the blank form called "output-form". The json is the object containing the form data and in this case ours is just json.

The new line should read:

```javascript
FMEServer.generateFormItems("output-form", json);
```

The correct parameters should look like this.

```javascript
function generateForm(json) {
    // Build the form items using the API
    document.getElementById(\"output-form\")\.innerHTML = \"\"; // Clears the output form
    FMEServer.generateFormItems("output-form", json); // Fill in the parameters here
}
```

8) Modify the generateForm function to Create a New Button

In the next section of the code we are creating a button. This button will be used to activate the next function to send the job to the FME Server.
In the first line of code in this section we are creating a variable with the output form in it. Then, we are creating another variable and creating an input element.

Next, we need to specify what the input will be. In our case it should be a button. Line 64 should now read:

```javascript
submitButton.type = "button"; // Fill in the type
```

Next, we can specify what the value should be the value of the button is what text is displayed on the button. This should be descriptive of what the user should do. Line 65 should now read:

```javascript
submitButton.value = "Run Data Download"; // Fill in the value
```

9) Review the runDataDownload function

We are now going to skip the showResults function for now and go to RunDataDownload beginning on line 82.

```javascript
function runDataDownload() {
    var repository = document.getElementById("repository-name").value;
    var workspace = document.getElementById("workspace-name").value;
    var form = document.getElementById("output-form");
    var params = ";";
    ...
}
```

In the beginning of the function we are creating variables for the repository, workspace, and form. We are also creating a variable for the params. This is currently a string with nothing inside.

The next part of the function is a loop. The purpose of the loop is it find all the parameters set by the user and to format them in a way that FME will use in the runDataDownload Function.

Any workspace specific parameter values must be written as a string with this format: name1=value1&name2=value2 etc...

We are going to examine the for loop (line 88). In the next step
In the beginning of the for loop, we are specifying that the loop will begin at the first form element and continue until the form is completed. The variable element is created with the form elements used in the output form.

```
for (var i = 0; i < form.length; i++) {
  var element = form.elements[i];
  if (element.type == "") { /* Fill in Element Type */
    params += element.name + "=" + element.value + ";";
  } else if (element.type == "") { /* Fill in Element Type */
    if (element.checked) {
      params += element.name + "=" + element.value + ";";
    } else {
      params += element.name + "=" + element.value + ";";
    }
  } else {
    params += element.name + "=" + element.value + ";";
  }
}
```

Now save the myFirstAppStart.html file.

10) Find the element types used in the Output Form

Next, we have to fill in the various element types in the form. The element type is the type of form element used, this can include a drop down list, check list or a text box. To find what elements exist within our form go to http://localhost:8000/MyFirstApp/myFirstAppStart.html

Please note this will only work if you have completed Exercise 14.

Once at the page right click and select inspect.

This will open up the Google Chrome Developer Tools and allow us to investigate the components on the page.

Next, click the Generate Full Form button on the page.
Once we have selected that button we can then start inspecting components on the page.
We are going to go through the form and right click on the elements on the form and inspect them.
In the Layers to Download form click one of the boxes and select inspect.

Once, we inspect on element on the page. It will highlight in the element in the developer tools. Here, we should see the input type is checkbox.

```
   ... <input type="checkbox" value="cenart" name="THEMES" value="0" />
   </label>
```

Next, inspect the drop down menu beside Output Coordinate System.

Here, we can see the element type is select.

Finally, inspect the text box next to Minimum X.
Here, we can see the element type is input.

11) Update the runDataDownload function

Go to line 88 and review the for loop.

```javascript
// Loop through unique parameters and build the parameter string
for (var i = 0; i < form.length; i++) {
  var element = form.elements[i];
  if (element.type == "input") // Fill in Element Type
    params += element.name + "=" + element.value + ";";
  else if (element.type == "select") // Fill in Element Type
    if (element.checked)
      params += element.name + "=" + element.selected.value + ";";
    else
      params += element.name + "=" + element.value + ";";
  else if (element.type == "text") // Fill in Element Type
    if (element.checked)
      params += element.name + "=" + element.value + ";";
    else
      params += element.name + "=" + element.value + ";";
...
```

In line 91 we see the first if statement. This statement should say, if the element type is equal to select. Then add the element name and the selected value to the params variable. At the end of the params variable we are adding an ampersand or & so we can add another parameter.

```javascript
if (element.type == "select") // Fill in Element Type
  { params += element.name + "=" + element.selected.value + ";";
  }
```

The element type should be select.

```javascript
else if (element.type == "checkbox") // Fill in Element Type
  
  if (element.checked)
    params += element.name + "=" + element.value + ";";
  else
```

Next, we need to fill in the element.type in the else if statement.

```javascript
if (element.type == "checkbox")
  { params += element.name + "=" + element.value + ";";
  }
```

If the element is checked it is added to the params string. The element.type should equal checkbox.

```javascript
else if (element.type == "checkbox")
  
  if (element.checked)
    params += element.name + "=" + element.value + ";";
  else
```

The last part of this loop is for every other element type. This should be the input box we inspected in the previous step.

After the parameters have been compiled, there will be an additional ampersand (&) at the end of the string. This is removed by the following statement,

```javascript
// Remove trailing & from string
params = params.substr(0, params.length - 1);
```
Finally, we can use the runDataDownload function.

```javascript
FMEServer.runDataDownload(repository, workspace, params, callback, showResults);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repository</td>
<td>String</td>
<td>The name of the repository that houses the workspace.</td>
</tr>
<tr>
<td>workspace</td>
<td>String</td>
<td>The workspace name.</td>
</tr>
<tr>
<td>parameters</td>
<td>String</td>
<td>Any workspace specific parameter values as a string: name1=value1&amp;name2=value2 etc...</td>
</tr>
<tr>
<td>callback</td>
<td>Method</td>
<td>The signature of the method or the function to run after the server has processed the request.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>The JSON Object returned from the server:</td>
</tr>
</tbody>
</table>

```json
{
  "serviceResponse": {
    "statusInfo": {
      "mode": "mode", "status": "status"
    },
    "fmeTransformationResult": {
      "id": "id", "jobStatus": "status", "result": "result_string", "resultSuccess": true_or_false, "serviceMsg": "msg", "serviceSuccess": true_or_false, "timeRequested": timestamp, "timeStarted": timestamp, "timeFinished": timestamp, "requesterResultPort": port, "requesterHost": ip, "request": "request_string", "requestKeyword": service_name, "priority": priority, "description": description, "fmeEngineResponse": Object, "jobID": id, "url": download_link
    }
  }
}
```

The final parameter in this function is the callback we named showResults. A callback is a function that needs another function to occur first before it can be used. In this case, we need the runDataDownload function to happen before showResults, the information created from runDataDownload will be passed to showResults. showResults is the function where the download will be displayed.

12) Review the JSON produced by the runDataDownload function

Next, we need to find the function above the runDataDownload function. This is showResults function and should be found on line 70.

```javascript
function showResults(json) {
  // The following is to write out the full return object for visualization of the example
  console.dir(json);
  var hr = document.createElement("hr");
  var div = document.createElement("div");
  // This extracts the download link to the clipped data
  var download = // find where the url is located in the JSON Object Returned from the Server
  div.innerHTML = "Download Result:" + download + "<a href="/Download Result/ax">download link</a>"
  document.body.appendChild(div);
}
```

The first line of the function uses console.dir which is a built in method that will print the JSON object to the console. We will use this to find what was produced from the last function.

Save the file. Then, go to http://localhost:8000/MyFirstApp/myFirstAppStart.html

Click on GenerateFullForm. Then, click the Run Data Download button.

The Download Result URL will not work because we will fill in the URL after. We are going to find the JSON produced by the last function to find where the URL is stored.

To find the JSON, right click on the page and select inspect. Then, in the developer tools find the Console tab.
In the Console, we can see the JSON and the structure of the response returned. The URL is nested json, then the serviceResponse.

13) Update the showResults function with the URL location

In the showResults function, on line 77 there is an empty variable for the download link.

```javascript
var download = json.serviceResponse.url;
```

In the previous step we found where the download link is stored. This will be entered into the variable. The download variable should be equal to json.serviceResponse.url.
14) Test the final product
Test the app again, and it should be completed!
If you missed a step, the full code can be found in the FMedata2018 under myFirstApp.html
The results should look like this:
First, we will receive a page where you can enter in the workspace and repository of the job we would like to run:

Then, we can enter in the parameters of the job we would like to complete.
Select cenart and streetcl and click Run Data Download.

After this, you will receive a link to download the resulting files. Please be patient because this may require a minute to complete depending on the server.
CONGRATULATIONS

By completing this exercise you have learned how to:
- Use the FME REST API JavaScript library to create an application
- Use the getWorkspaceParameters, generateFormItems, and runDataDownload functions
In the first exercise, we built a form that allows a user to download requested data. However, it would be difficult for a user to gauge the coordinates needed for the download. ESRI Maps can be incorporated so the user can select an area on the map they would like to see.

This example allows a user to draw a point, line, or polygon using the drawing tools from the ArcGIS JavaScript API. When the geometry has been created on the map, it is then sent to a workspace as well known text (WKT). The workspace will convert that geometry into GeoJSON and stream it back to the web application.
In this exercise, we will be creating a website with a map that allows a user to select an area. This selection will be sent to a workspace. The workspace contains an HTTPCaller which connects to the DriveBC API. The DriveBC API has current data on any traffic incidents that may have occurred. In the workspace, the selected area will clip current accidents. The user will be sent a link with a Google Map with accidents in the selected area. This exercise is meant to demonstrate how to use other APIs in a workspace, how to integrate coordinates selected from a user into a workspace, and how to stream back data to the user.

**WARNING**

For this exercise to work properly you will need to have completed Exercise 14 or have an existing web server for testing.

**Set Up the HTML File**

In this application, we will use two separate documents. In the first application, we stored the HTML and the JavaScript in the same document and in this application we are splitting these documents into two. Many websites store their JavaScript, CSS, and HTML pages separately to promote a cleaner workflow.

1) **Open the index.html document**

Go to: C:\FMEData2018\Resources\RESTAPI\mySecondApp\www and find the index.html document. Once, you are in the folder right click on the index.html file and select Edit with Notepad ++.

2) **Modify the Head Section in the HTML file**

Review the head section of the code (lines 5-16). This head section looks very similar to one in the previous exercise. The difference is that there are a few additional lines of code:

```html
<title>ArcGIS Maps Example</title>

<link rel="stylesheet" href="https://playground_fmserver.com/case/FMEserverExamples.css" type="text/css"/>

<script type="text/javascript" src="https://api_fmserver.com/v1.2/FMEserver.js"></script>

<! -- The following are Required for ArcGIS Maps -->

<script type="text/javascript" src="https://js.arcgis.com/3.8/js/esri/esri.js"/>

<script type="text/javascript" src="https://js.arcgis.com/3.8/js/esri/arcgis.js"></script>

<! -- Connects to our JavaScript file -->

<script type="text/javascript" src="/Update the JavaScript file name"></script>

</head>
```

The section of code below (lines 11-13) connects to the Esri CSS. The Esri CSS adds the styling components to the map. Then, the JavaScript links to a JavaScript library made by Esri.
Line 15 links our HTML file to our JavaScript file. We need to update the src (source) parameter to the name and pathway of the file. We are changing the parameter to the name of the file indicating that the JavaScript file will be kept in the same folder as the HTML file. The JavaScript file will have to be kept in the same folder as the index.html file or the pathway will be invalid. Our JavaScript file will be called, arcgismap.js so the line of code will be:

```html
<script type="text/javascript" src="arcgismap.js"></script>
```

We will be updating a JavaScript file to make our webpage interactive. It is a best practice to separate your JavaScript from the HTML to ensure that each file is as clean and easy to follow as possible.

3) Review the body of the index.html file

Go to the body of the index.html file (line 18).

```html
<body>
  <h4>This example clips an area selected by the user to BC Traffic Data. </h4>
  <form id="exampleForm">
    <label>Step 1</label> - Draw the Polygon: <button name="draw" type="button" value="Draw">Draw</button> <br>
    <label>Step 2</label> - Submit the Request to FME Server: <button name="reset" type="button" value="Reset">Reset</button> <br>
  </form>
</body>
```

First, we have a very simple heading to explain the purpose of the application (line 19). The <h4> tag indicates that the sentence is a heading (specifically heading 4 which should be smaller than heading 2 or 3).

Then, we create a form (line 20-27), with buttons to activate different functions in the JavaScript. The draw button will allow the user to draw on the map. The reset button will reset the users drawing. The third button will use the coordinates selected from the user's drawing and clip the data to the selected area.

```html
<body>
  <h4>This example clips an area selected by the user to BC Traffic Data. </h4>
  <form id="exampleForm">
    <label>Step 1</label> - Draw the Polygon: <button name="draw" type="button" value="Draw">Draw</button> <br>
    <label>Step 2</label> - Submit the Request to FME Server: <button name="reset" type="button" value="Reset">Reset</button> <br>
  </form>
</body>
```

4) Review the components of the index.html file

Now we are going to view our webpage on the web directly. The index file will be located here http://localhost:8000/MySecondApp/www/

Right-click, on the page and select Inspect. We should see this:
Inspecting the page will open up the developer tools which are seen on the right.

As we hover the cursor over the page, it will select different elements on the page and highlight where they exist. It will also display any errors we may have. We currently have errors because we don't have the code required to initialize the map.

To make this page effective, we need code to:

- Initialize the map
- Draw the Polygon
- Reset the Polygon
- Submit the request to the FME Server

We can now modify the JavaScript to make these functions functional!

5) Open the JavaScript File

Now, we can open the JavaScript file. Go to C:\FMEData2018\Resources\RESTAPI\mySecondApp\www and locate the arcgismap.js file right click on the file and select Edit with Notepad++.

6) Modify the code to initialize the map

On line 3, the window.onload function begins, most of this code is from the Esri JavaScript API. In this course, we won't be covering the Esri JavaScript API in much detail. To learn more about using the Esri JavaScript API visit the ArcGIS API for JavaScript Page.

Review the first 15 lines of code of the document. The first line of code creates the variables for the map, toolbar and clipping geometry. Next on line 3, we have a function that will be activated when the window loads.

In the ArcGIS API there are many functions, objects, and classes stored in modules. These modules have to be called to use the functions listed. On lines 5-9 the modules are loaded. On lines 10-14 the code references the classes stored in the modules.
On line 17 we are using the map variable (created on line 1) to create the map object we see on screen. We are going to set four parameters on the map, these include the the basemap type, the coordinates to center the map, the default zoom level, and the minimum zoom.

```javascript
map = new Map("mapDiv",
{
  basemap: "ENTER IN THE BASEMAP HERE", // Select the basemap
  center: [], // Enter the coordinates to center the map
  zoom: 7, // Set the default zoom
  minZoom: 4, // Set the minimum zoom level
  smartNavigation: false
});
```

On line 19 we are going to select the streets basemap. If you'd like to view all the possible basemaps visit Esri Basemaps.

```javascript
basemap: "streets", // Select the basemap
```

On line 20, we are going to select the center of the map using the coordinates. We are going to center the map around Vancouver, BC.

```javascript
center: [-123.114166, 49.27], // Long and lat.
```

On line 21, we are going to set the default zoom to 7. Then, on line 22 we are going to set the minimum zoom to 4.

```javascript
zoom: 7,
minZoom: 4,
```

The addToMap(geometry) function (line 36), is from the Esri JavaScript API. This function creates symbols that are added to the map when a user draws an area to clip. As the user selects areas on the basemap, the function will record the coordinates of the areas selected.

```javascript
function addToMap(geometry)
{
  var symbol = new SimpleFillSymbol(
    SimpleFillSymbol.STYLE_SOLID,
    new SimpleLineSymbol(
      SimpleLineSymbol.STYLE_DASHDOT,
      new Color([255, 0, 0]), 2
    ),
    new Color([255, 255, 0, 0.25])
  );
}
```

From line 38-45, we are specifying the appearance of a symbol that will be added when a user selects an area to clip.

On line 46, we are specifying which coordinate system to record the geometry in. In our case this will be Web Mercator.

```javascript
geometry = webMercatorUtil.webMercatorToGeographic(geometry);
```

From line 47-51, we are creating a graphic. A graphic is comprised of the geometry and symbol we specified on the previous lines. We then declare the clippingGeometry is the variable which will store the geometry.

```javascript
geometry = webMercatorUtil.webMercatorToGeographic(geometry);
var graphic = new Graphic(geometry, symbol);
map.graphics.clear();
map.graphics.add(graphic);
toolbar.deactivate(Draw.POLYGON);
clippingGeometry = geometry.rings[0];
```

From line 54-64 we have two functions. These functions are activated when a user selects the draw or reset buttons.
function drawPolygon()
{
    drawReset();
    toolbar.activate(esri.toolbars.Draw.POLYGON);
}

function drawReset()
{
    toolbar.deactivate(esri.toolbars.Draw.POLYGON);
    map.graphics.clear();
}

7) Refresh the index.html page and test it out

This code uses the Esri JavaScript API to display a map on the page and activate the drawing and reset functions. Currently, the user can draw a bounding box on the map but this box is not used in any other functions.

8) Update the FMEServer Init function to include your FMEServer URL and token

Find the FMEServer.init (line 67-72). Use your Server Hostname and token to update the server and token section to your own. If you are on an FME training machine the URL should be http://fmetraining. If you wish to locate your token again. Go to this link:

http://<yourServerHost>/fmetoken/

9) Uncomment the processClip Function

The processClip function (line 90-114), takes the coordinates selected by the user and formats them in a way that can be used as a published parameter.

Highlight the processClip function. Then, right click and select Block Uncomment.
First, we have not uploaded the workspace to our server, so this will not currently work but let's discuss how we could recreate these functions. Our goal with the processClip function is to set the proper parameters so we can later use the runDataDownload function.

This code can be found in the ArcGIS Maps Integration on the FME Server Playground.
To use this the DataDownload function, we need the following variables:

- Repository
- Workspace
- Parameters
- Callback

This function is located at the end of our code (line 113).

```java
  112  // Use the FME Server Data Download Service
  113  FMEServer.runDataDownload(repository, workspace, params, showResults);
  114
```

However, to be able to use this function we first need to specify the variables. We are going to begin by specifying the repository and the workspace. This is easier than the previous exercise because we can hard code this in.

The repository and workspace have to specified on lines 92 and 93 respectively. We will upload this workspace later on in the exercise. However, these values should match the actual repository and workspace values to ensure this exercise works.

**We need to update var repository = "" to var repository = "RESTTraining"**

**We need to change var workspace = "" to var workspace = "webapp.bc.roads.fmw"**

```javascript
var repository = "RESTTraining"; //Enter repository name
var workspace = "webapp.bc.roads"; //Enter the workspace name
```

Next, we can create the parameter variable. With the workspace (created later in the exercise), we need to know all the parameters in advanced to be able to use this call.

Therefore, if you were using this call in a real situation, it is best to create the workspace first. However, the only parameter in the workspace is GEOM, which will accept Well Known Text (WKT) from our application.

Well Known Text is a format that creates a string that looks like this:

```javascript
POLYGON((-123.31968482260129 49.05758039932133, -123.64927466635129 50.269944861577876, -119.33714087728879 50.34712587270931))
```

In JavaScript we can add bits of strings together to create the full parameter for example we can create a function:

```javascript
a = "app";
b = "le";
apple = a + b;
```

apple will then equal the string "apple."

This can also be done with these statements:

```javascript
a = "The REST API is ";
a += "fun";
```

a will then equal "The REST API is fun".

We are going to use this method to create the parameter string needed for this call to run.

First, we start with the declaring that the input will be a Polygon.

```javascript
  95  // Process the clippingGeometry into a WKT Polygon string
  96  var geometry = "POLYGON(";
```

Now, we need to input the coordinates of that were selected by the user. Earlier in the program, we specified that the clippingGeometry would contain the geometry selected by the user.

We need a function to sort through all the data in the clippingGeometry variable. For this, we will use a for loop. This loop will end when there is no more data left.

```javascript
  98  for (var i = 0; i < clippingGeometry.length; i++)
  99  {
 100     var lat = clippingGeometry[i][1];
 101     var lng = clippingGeometry[i][0];
 102     geometry += lng + " " + lat + ",";
 103  }
```
It's important to note that the index of elements in an array in JavaScript start from 0 instead of 1. The lng, we are assigning the first coordinate read, and then the lat is in the second position. Then the geometry string is added to by this statement:

```javascript
geometry += lng + " " + lat + ",";
```

This will leave us with the following statement:

```javascript
POLYGON([-124.26773045312264, 49.13799011659428, -124.20181248437265, 50.13407018172526, -121.21353123437346, 50.056543673436465, -121.3563534999984, 48.83518734769823, -124.26773045312264, 49.13799011659428],
```

To complete the geometry string we need to remove the trailing comma and close the parentheses.

```javascript
105  |    geometry = geometry.substr(0, geometry.length - 1);
106  |    geometry += "";
```

We have now created the statement for the parameters.

Each parameter you create will have a name associated with it. In our case, our parameter name will be GEOM.

```javascript
110  |    var params = "GEOM=" + geometry;
```

Now finally, we have completed the parameters required to run the DataDownload function. We just need a function to accept the json that the DataDownload function produces, so we use the showResults function.

10) Uncomment the showResults Function

Find the showResults function (line 74-88). Then, highlight it right click and select block uncomment.
On lines 78-79 the code is creating a new div and a new divider.

```javascript
var hr = document.createElement("hr");
var div = document.createElement("div");
```

Next, the function will need to find the URL returned by the Server. This URL links directly to a file the user can download. In the previous exercise we used the console to find the location of the URL. However, this information is also in the FME Server JavaScript API.

Using the documentation, we can review what the JSON will look like from the FME Server:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repository</td>
<td>String</td>
<td>The name of the repository that houses the workspace.</td>
</tr>
<tr>
<td>workspace</td>
<td>String</td>
<td>The workspace name.</td>
</tr>
<tr>
<td>parameters</td>
<td>String</td>
<td>Any workspace specific parameter values as a string: name1=value1&amp;name2=value2 etc...</td>
</tr>
<tr>
<td>callback</td>
<td>Method</td>
<td>The signature of the method or the function to run after the server has processed the request.</td>
</tr>
</tbody>
</table>

Using the `FMEServer.runDataDownload(repository, workspace, parameters, callback, return Object)

To find the download link we use the following statement:

```javascript
// This extracts the download link to the clipped data
var download = json.serviceResponse.url;
```

This is stating where the URL download link is, it is within the JSON, in the serviceResponse, and under URL. You can find this yourself here:

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The serviceResponse is within the json</td>
<td>The JSON Object returned from the server:</td>
</tr>
</tbody>
</table>

Now, that we have the URL, we can display the URL in a link on line 85.
We now have an application that will accept coordinates selected by a user then use the DataDownload service to return features in that area. Now we can create a workspace to use.

**Create a Workspace**

11) Open a blank workspace in FME Workbench and add a Creator Transformer to the workspace

12) Add the GeometryReplacer

The first step is to get the user input from the web application. If you look at this section of the code, you can see that we are setting that the code is accepting a well-known text string from the user input and placing it in a parameter called GEOM.

```javascript
var params = "GEOM=" + wktString;
```

So, to bring this information into the workspace, we will use the GeometryReplacer. In the GeometryReplacer fill the parameters out below:

The Geometry Encoding will be set to OGC Well Known Text. This is what is returned from user selecting points on the map. The Geometry Source is a user parameter which will be created. On the geometry source click on the dropdown list, find the user parameter, then click on Create User Parameter.

The dialog that opens allows us to create a new parameter. Create one using the following parameters:
13) Add the CoordinateSystemSetter

At this point in the workspace, we have the coordinates into the system, and it has been converted to geometric features. However, there is no way for FME to know what the coordinate system is. So we are going to set it to LL84. Connect the CoordinateSystemSetter to the Output port of the GeometryReplacer.

14) Add the HTTPCaller

Add a HTTPCaller to the canvas and attach it to the Creator.

In this section we are using the Drive BC API. The Drive BC API retrieves up to date traffic information and will relay it to your map. If you wish to find the more information visit the Drive BC API website.

Next, we need to add the HTTPCaller to contact the Drive BC API. The Drive BC API is open to the public, so no authorization or authentication is needed. Under Request URL put:

http://api.open511.gov.bc.ca/events?format=json&status=ACTIVE

The HTTP Method is GET

The HTTPCaller Parameters should look like this:
15) Run with Feature Caching

Now, we are going to test the workspace. This will ensure that the workspace is connecting to the DriveBC API. This will also let us examine what is being returned by the API.

First, save the workspace as, webapp.bcroads.fmw in the mySecondApp folder. To use feature caching click on Run in the toolbar. Then click run with feature caching.

Then, select the play button to run.

Click, on the magnifying glass on the HTTPCaller to examine the output from the REST API. The JSON returned by the API is stored within the response_body.
Within the response_body you should see should be structured in a similar way to what's below.

```json
{
  "events": [
    {
      "jurisdiction_url": "http://api.open511.gov.bc.ca/jurisdiction",
      "url": "http://api.open511.gov.bc.ca/events/drivebc.ca/DBC-1301",
      "id": "drivebc.ca/DBC-1301",
      "headline": "CONSTRUCTION",
      "status": "ACTIVE",
      "created": "2018-09-04T13:18:13-07:00",
      "updated": "2018-09-04T13:18:13-07:00",
      "description": "Highway 1, westbound. Paving operations planned between Exit 146 and Exit 135: Highway 9 for 11.0 km (Chilliwack). Starting today at 9:00 PM PDT until tomorrow at about 6:30 AM PDT. 28 km west of Hope between Herrling Island and Hwy 9. Last updated today at 1:18 PM PDT. (DBC-1301)",
      "+ivr_message": "Highway 1, westbound. Paving operations planned between Exit 146 and Exit 135: Highway 9 for 11.0 km (Chilliwack). Starting Tuesday, September 4 at 9:00 PM until tomorrow at about 6:30 AM. Last updated Tuesday, September 4 at 1:18 PM.",
      "schedule": {
        "intervals": [
          "2018-09-05T04:00/2018-09-05T13:30"
        ]
      },
      "event_type": "CONSTRUCTION",
      "event_subtypes": ["ROAD_MAINTENANCE"],
      "severity": "MINOR",
      "geography": {
        "type": "LineString",
        "coordinates": [
          [-121.760359,
           49.181117
          ]
        ]
      }
    }
  ]
}
```

16) Add JSONFragmenter

On the canvas add a JSONFragmenter and attach it to the HTTPCaller.

The response body will be returned as a long section of JSON. We need to break this down and expose the attributes needed. This JSONFragmenter will be used to expose the attributes listed under "events" as in the above code.

Under Source fill in the parameters:

**Input Source:** JSON Attribute

**JSON Attribute:** response_body

Next move onto the Parameters section.

All the important information is stored in the category of events to gain access to it under the JSON Query write:

```javascript
json["events"][*]
```

**Fragment as Format:** JSON

**Reject Features which Produce No Fragments:** Yes
Now we can flatten the JSON to produce the attributes needed for the table.

In the Attributes to Expose Section click on the ... and write the following attributes: status, headline, description, created, and updated.

The Flattening Parameters should look like this:
17) Add another JSONFragmenter
Add another JSONFragmenter to the canvas and attach it to the previous JSONFragmenter. Now, we need to expose the Geography component to create features.

Fill in the following parameters:
- **Input Source**: JSON Attribute
- **JSON Attribute**: response_body

Next, move onto the Parameters section.
- **JSON Query**: json["geography"]
- **Fragment as Format**: JSON
- **Reject Features which Produce No Fragments**: Yes

Finally, move onto the Flattening Parameters.
- **Flatten Query Result into Attributes**: No

18) Add a GeometryReplacer
Add a GeometryReplacer and attach it to the second JSONFragmenter.

The GeometryReplacer is an amazing transformer. It can read the JSON from the Response Body and will automatically create the geometry associated with it.

Fill in the following parameters:
- **Geometry Encoding**: GeoJSON
- **Geometry Source**: response_body
- **Remove Attribute**: Yes

19) Add the CoordinateSystemSetter
Add a CoordinateSystemSetter and attach it to the GeometryReplacer.

Now, we need to set the coordinate system used. The coordinate system is referenced in URL returned by the call. So we can set it to LL-WGS84.
Currently, the workspace should look like this:

20) Add the Clipper Transformer

Add a Clipper Transformer to the canvas and accept the current defaults.

Now, we are going to use the Clipper transformer to clip the area selected area. The clipper is the area selected by the user, and the clippee is the geometry from the BC Drive API.

The workspace should look like this:

21) Run with Feature Caching

Now, we are going to test the workspace. This will ensure that the workspace is working and clip the correct area.

Select the play button to run.

Click, on the magnifying glass on the Inside port on the clipper to view the area selected.

This will open the FME Data Inspector.

The FME Data Inspector will display the points and lines created. To display the background map go to the top menu bar. Find Tools, select FME Options. This will open the FME Options window.

Background Format should be changed from None to Stamen Maps. Then, select Parameters...

Select, the terrain map list. Select OK. Then, in the FME Options window select OK. This will activate the background map.

All the results should be in BC. After exploring the features returned close the Data Inspector. Then, return to FME Workbench.

22) Add the NoFeaturesTester
Start typing NoFeaturesTester in FME Workbench and select the custom transformer. The NoFeaturesTester should be attached the Inside port of the Clipper. This transformer will test if there was an output or not. If there is no input, this means all the features were outside or rejected.

23) Add the DateTimeConverter

Add a DateTimeConverter and attach it to the OUTPUT port of the NoFeaturesTester. The DateTimeConverter is used to update the time and date, so it is more readable for the user.

Fill in the following parameters:

**Datetime Attributes**: updated created

**Input Format**: Auto detect FME and ISO formats

**Output Format**: %Y:%m:%d %H:%M:%S (Exif datetime)

**Repair Overflow**: No

24) Add the HTMLReportGenerator

Add the HTMLReportGenerator and attach it to the DateTimeConverter. Now, we are ready to create a webpage with the following information. We need three sections, a title, a map, and a table.

Under Page Settings find the Page Title box. The Page Title should be Drive BC API.

Then, create the title. Under Page Contents select Custom HTML. Under Content Settings copy and paste the following:

```html
<h1>Drive BC Road and Weather Conditions</h1>
```

Next, we will add a Map Component. The Page Contents should be Map(Google). Then, the Content Settings should be set up as follows:

**Label Attribute**: description

**Layer Color**: 255,55,5

**API Key**: << blank >>
For now, we are not going to fill in the API key. As this is for development purposes this is okay. If you want to use this in a production setting you would require an API key.

If you wish to obtain an API key, visit this website, Google API Key

Next, create a table with the status, created, updated, description, and headline columns.

25) Add another HTMLReportGenerator

Add another HTMLReportGenerator and attach it to the NOINPUT port of the NoFeaturesTester.

In the Page Title box write:

Drive BC API

Under Page Contents select the Header. Then, under Content Settings write:

There are no incidents in the area you selected.

26) Add the HTML Writer

Finally, add the HTML writer to write to a file of your choice. Attach the HTML writer to both HTMLReportGenerators.
The full workspace should look like this:

27) Save and Run the Workspace

Now, we can save and run the workspace to ensure that everything is working properly. Since, we do not have the API key in place we will receive a message that says that this is for development purposes only. If we added the key, we would have a much more sophisticated page.

This API works with live data, therefore, your results will vary from the screenshot.

28) Upload the Workspace to your Server.

Publish the workspace to FME Server. Upload under to the RESTTraining repository and save the workspace as webapp.bc.roads.fmw. Register the workspace with the Data Download service.
Test the Application

Step 1. Draw the Area (double-click to close the polygon)

Step 2. Select Clip Data to Area

This example clips an area selected by the user to BC Traffic Data.

Step 1 - Draw the Polygon: Draw | Reset

Step 2 - Submit the Request to FME Server: Clip Data To Area

Step 3. Download the Data

This example clips an area selected by the user to BC Traffic Data.

Step 1 - Draw the Polygon: Draw | Reset

Step 2 - Submit the Request to FME Server: Clip Data To Area

Download Result

Step 4. Review the File
CONGRATULATIONS

By completing this exercise you have learned how to:

- Use ArcGIS maps to send coordinates to your workspace
- Use coordinates from an online resource in your workspace
- Use the DataDownload function in the FME Server JavaScript API
- Use an external API in a workspace
### 9.4 Data Visualization using ArcGIS Map

In this example, we will create a map using an Esri basemap and use the Esri JavaScript functions. In this example when a box is checked the a KML layer is loaded. When the box is unchecked, then the layer will be invisible. This is a great example of streaming real-time data to a map. While parks and Skytrain stations may not be updated frequently, this demo may be applied to real-time data. For more information on the Esri JavaScript, please refer to the ArcGIS Developers Documentation.

Please note, the repository that the workspaces are uploaded to will have to have guest permissions. If you need to know how to do this go to Section 8.1 and see Setting Permissions.
Exercise 17  Data Visualization

<table>
<thead>
<tr>
<th>Data</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>To use the Esri JavaScript and the Power of FME to create and easy to use and interesting map with live data.</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>How to use Esri JavaScript, Data Streaming Direct URLs, and online resources in a workspace.</td>
</tr>
<tr>
<td>Starting HTML</td>
<td>C:\FMEData\Resources\RESTAPI\myThirdApp\myThirdApp.html</td>
</tr>
<tr>
<td>Completed HTML</td>
<td>C:\FMEData\Resources\RESTAPI\myThirdApp\myThirdAppComplete.html</td>
</tr>
<tr>
<td>Projects</td>
<td>C:\FMEData\Resources\RESTAPI\myThirdApp\WebApplication.fsproject C:\FMEData\Resources\RESTAPI\myThirdApp\BestPracticeAnalysis.fsproject</td>
</tr>
</tbody>
</table>

This exercise was made to demonstrate how to stream KML Layers produced from workspaces to an online map. When a user selects a layer, a workspace is activated to produce the KML Layer. However, by default Esri will cache the results of the KML Layer for 30 minutes. Esri caches layers to enhance the performance. However, if your data needs to be refreshed on a shorter interval this code will have to be modified.

**WARNING**

For this exercise to work properly you will need to have completed Exercise 14 or have an existing web server for testing.

1) Open the myThirdApp.html file

Go to C:/FMEData\Resources\RESTAPI. Then, right click on the myThirdApp.html and select Edit with Notepad++.

2) Review lines 1-12 in the code

This is the code needed to access the ESRI JavaScript that we will be using to host the KML Layers streamed to the application through FME Server.

```html
<!---- For this exercise to work properly you will need to have completed Exercise 14 or have an es

<meta charset="utf-8">
<meta name="viewport" content="initial-scale=1,maximum-scale=1,user-scalable=no">
<title>Data Loading Example</title>
<link rel="stylesheet" href="https://js.arcgis.com/4.7/esri/css/main.css">
<script src="https://js.arcgis.com/4.7"></script>
```

Notice, in this application there is no connection to the FME Server JavaScript API. The ArcGIS JavaScript API will be used to create the map the objects and turn the layers on and off.

The layers are created from workspaces hosted on the FME Server. We are using a Direct URL to activate the layers.

3) Review the Body of the HTML

The body of the HTML (lines 110 - 126), holds the map div (viewDiv), a heading, and the map layers.
The above code creates a title and checkbox list on the web page. These checkboxes will represent the layers of the map. Once a checkbox is activated; this will trigger a function to enable the data streaming.

4) Review the CSS

TIP

*The following CSS is sourced from ArcGIS Introduction to Layers*

The styling components (lines 128-161), describe how each HTML element is displayed on the screen.
For the heading, we are giving it a z-index of 99 to ensure that it is placed in front of the map (line 132). It has a white background (line 133) and has an absolute position on the screen (line 130). Which means as the screen size changes the heading will remain in the same place.

The viewDiv which contains the map is given a height and width of 100% (line 147-148). The map will be displayed on the entire screen regardless of the screen size.

The layer toggle which contains the layers available will be formatted similarly to the heading. It will be placed on top of the map and has a white background.

5) Navigate to the Projects in your FME Server

Open the FME Server by visiting localhost/fmeserver. Sign into the account as the admin. The username will be admin and the password will be admin as well. Next, look on the left-hand panel and click Projects.

6) Import the Project Folders

Click the Import button.
The next step is to upload the Project Folder that contains the repository and workspaces needed for this web application. Go to the File Explorer window. Navigate, to the FMEData2018 folder, click Resources, select REST API, then myThirdApp.

In this folder, there are two .fsproject files. The WebApplication project contains the workspaces and the repository needed for the web application.

Additionally, upload the BestPracticeAnalysis.fsproject file. This project folder contains additional images for use with the KMLStyler. This allows the user to have more symbolization options when creating KML.

Drag and drop the WebApplication.fsproject, then select Import. This will Import the WebApplication project into your FME Server. Then repeat the same process to upload the BestPracticeAnalysis.fsproject file.

7) Give the Guest Account Access to the WebApplication repository

To give the repository guest permissions, you will have to access your FME Server. First, look on the left-hand panel of your FME Server. Find Security and then look for Users.

Once on the Users page click guest.

Scroll down to the permissions and expand the repositories tab. Then check the read and run permissions. Then click OK found in the bottom right of the page.
8) Give the restapi Account Access to the WebApplication repository (if you created this user earlier in the course)

The restapi user was created in exercise 1 of the course to reduce any security risks associated with the using the admin token.

On the same Users page click restapi.

Scroll down to the permissions and expand the repositories tab. Then, check the read and run permissions. Even if the restapi account imported the Project. It isn't guaranteed that the user will automatically be granted access to the files.

9) Review the require and the function section

In the ArcGIS API there are many functions, objects, and classes stored in modules. These modules have to be called to use the functions listed. On lines 14-20 the modules are loaded. On lines 22-23 the code references the classes and stored in the modules.

10) Add Direct URLs for each new KMLLayer Created

Line 26-47 contains the layers of the map. Here, we are creating a new KMLLayer, this will create a layer from the direct URL from the server. For each layer, an id is required. Additionally, the visibility should be initially set to false.

Now, we need to access our FME Server and find the Direct URLs for each layer.

Log into your FME Server as the admin. On the left hand side of the screen find the Repositories tab. Select repositories and then select the WebApplication repository. Find webapp.skytrains.fmw, go to the Advanced dropdown menu.

Copy the Direct URL Example.

Paste the Direct URL into the URLs section of the transportationLyr (line 28).

```javascript
var transportationLyr = new KMLLayer(
    url: "http://localhost/fmedatastreaming/WebApplication/webapp.skytrains.fmw?",
    // Skytrain Layer
    id: "skytrainlayer",
    visible: false
);`
Repeat this process for the other empty URLs.

Find the `webapp.skytrainbuffer.fmw` in the WebApplication repository. Then, copy the Direct URL from the Advanced menu and paste it into the `skytrainbufferLyr` URL (line 33).

```javascript
var skytrainbufferLyr = new KMLLayer({
  url: "http://localhost/fmedatastreaming/WebApplication/webapp.skytrainbuffer.fmw?", // Skytrain Buffer Layer
  id: "skytrainbufferLayer",
  visible: false
});
```

Find the `webapp.parks.fmw` in the WebApplication repository. Then, copy the Direct URL from the Advanced menu and paste it into the `parksLyr` URL (line 38).

```javascript
var parksLyr = new KMLLayer({
  id: "parksLayer",
  visible: false
});
```

Find the `webapp.foodTrucks.fmw` in the WebApplication repository. Then, copy the Direct URL from the Advanced menu and paste it into the `foodLyr` URL (line 43).

```javascript
var foodLyr = new KMLLayer({
  url: "http://localhost/fmedatastreaming/WebApplication/webapp.foodTrucks.fmw?DestDataset_OGCKML=%24(FME_SHAREDRESOURCE_DATA)%2FSienna%2FOutput%2FfoodTrucks.kml", // Food Truck Layer
  id: "foodLayer",
  visible: false
});
```

Now, all of the URLs are in place. However, the URL with localhost will not work. We need to replace localhost to your Public IP Address.

**11) Update localhost to your Public IP Address in the URLs**

Now, we are going to find our Public IP Address by visiting `whatismyip.com`.

Next, we are going to copy the URL listed as the Public IP Address.

Now, we can modify the URLs we previously inserted in the last step. In Notepad ++, use the command Ctrl + H to open the Replace menu.

In the Replace menu, under Find what: write localhost, under Replace with paste the IP Address found previously.
Find and replace all of the localhost instances with your Public IP Address.

12) Update the basemap used in the map

The code to initialize the basemap (lines 50-55) needs to be updated with the basemap name.

```javascript
var map = new Map({
    basemap: "topo"
});
```

On line 54, add "topo" after basemap:

```javascript
var map = new Map({
    basemap: "topo"
});
```

13) Add the Layers to map.add

Currently, there are no layers in the map.add function (lines 60-63). We need to update the map.add to include the layer variables we assigned previously.

```javascript
map.add(transportationLyr);
map.add(skytrainbufferLyr);
map.add(parksLyr);
map.add(foodLyr);
```

14) Update the center of the map and the zoom level

Next, in the code (line 67-76) we can modify the code to adjust the center and zoom level. In the square brackets on the center variable (line 74) insert the coordinates -123.10, 49.268. The zoom level (line 75) should be set to 14.

```javascript
var view = new MapView(
    { container: "viewDiv",
      map: map,
      center: [-123.10, 49.268],
    });
```
15) Review the code to create the toggle variables

In the next section of code (lines 78-86), Variables for the toggles are created using dom.byId, this will get the layer objects created previously.

```javascript
/****************************
* Variables are created for the Toggles, these
* toggles are attached to the ids assigned to the checkboxes
****************************/
var streetsLyrToggle = dom.byId("streetsLyr");
var skytrainbufferLyrToggle = dom.byId("skytrainbufferLyr");
var parksLyrToggle = dom.byId("parksLyr");
var foodLyrToggle = dom.byId("foodLyr");
```

16) Add the Code to toggle the visibility of the food layer

The next section of code (line 90-107), changes the visibility of the layer based on if the toggle is checked. The code for the foodLyrToggle has been omitted.

On line 105 write `foodLyr.visible = foodLyrToggle.checked;`

```javascript
/****************************
* The following code manipulates the visibility of the layer.
* When the layer is checked then the layer becomes visible.
****************************/
on(streetsLyrToggle, "change", function()
{
  transportationLyr.visible = streetsLyrToggle.checked;
});
on(skytrainbufferLyrToggle, "change", function()
{
  skytrainbufferLyr.visible = skytrainbufferLyrToggle.checked;
});
on(parksLyrToggle, "change", function()
{
  parksLyr.visible = parksLyrToggle.checked;
});
on(foodLyrToggle, "change", function()
{
  foodLyr.visible = foodLyrToggle.checked;
});
```

17) Test the Page!

When you test out the page, you should be able to select and de-select different layers. Each layer runs a workspace on your FME Server.
CONGRATULATIONS

By completing this exercise you have learned how to:

- Use the Esri JavaScript API to create a web application
- Use a workspace to generate a KMLLayer
- Use directURLs to generate layers on a map
9.7 Data Uploads and Downloads

Upload an AutoCAD file for Quality Analysis

This web page will accept a file uploaded by a user. Then, using the FME Server, the file will be used in a validation workspace. If the file passes the validation and the user clicks "Display Results on Map" then the geometry will appear in all blue. If the file passes the validation and the user clicks "Download Results" then a download link will appear. The download link will contain an HTML file that says "All data has passed the validation tests." If the file fails the validation and the user clicks "Display Results on Map" then the geometry will appear in all red and the errors will appear in red. If the file fails the validation and the user clicks "Download Results" then a download link will appear. The download link will contain an HTML file with a full report on what components did not pass the tests.

File List (Be patient, some files may take a while to upload): 

File List [Refresh File List]

tabl_data.dwg 163108 bytes

Source: Autodesk AutoCAD DWG/DXF File(s) Choose File: tabl_data.dwg

Upload File

Display Results on Map Download Results Reset Session

The final results link will appear here:

This exercise builds a Web Application that can accept a file to upload. Then, the application will run the file through a workspace to validate the file. If the file passes, it will stream directly to the map in the application, and a KML file will be downloaded. If the file does not pass, then no map will appear, and an HTML file will be downloaded with the results.
Exercise 18  Data Upload and Translation

<table>
<thead>
<tr>
<th>Data</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Goal</td>
<td>To display how to use the FME Server JavaScript API to upload files to the FME Server and run them through a workspace.</td>
</tr>
<tr>
<td>Demonstrates</td>
<td>How to use the getSession, generateOptions, dataUpload functions in a web application.</td>
</tr>
<tr>
<td>Completed HTML</td>
<td>C:\FMEData2018\Resources\RESTAPI\myFourthApp\myFourthAppCompleted.html</td>
</tr>
<tr>
<td>Starting HTML</td>
<td>C:\FMEData2018\Resources\RESTAPI\myFourthApp\myFourthApp.html</td>
</tr>
</tbody>
</table>

This application is meant to demonstrate the data upload function and to explain how it works in the FME Server.

**WARNING**

For this exercise to work properly you will need to have completed Exercise 14 or have an existing web server for testing.

1) Open the myFourthApp.html file

Go to C:/FMEData2018/Resources/RESTAPI/myFourthApp.html then right click on myFourthApp.html and select Edit with Notepad++.

2) Review the head section

In the head section (lines 4-11). Here, like the past exercises, there are connections to the FME Server Example CSS and the FME Server JavaScript.

---

**Set up the Form within the Body**

3) Add a value to the example form

In this exercise we are creating an interface. Where a user can choose a workspace that requires data to be uploaded.

To complete this, first the user must choose which workspace they would like to select. Here, we are going to provide a sample value for the user. In our case this will be the Easy Translator found in the Samples repository.

Find the example form (line 13-18). On line 15, under value replace "// Fill in the value" with "Samples". On line 17, under value replace "// Fill in the value " with "easyTranslator.fmw".

The new text should appear as below.

```html
<form id="exampleForm">
  <label>Repository:  </label><br>
  <input id="repository-name" type="text" name="repository" value="Samples" /><br />
  <label>Workspace:  </label><br>
  <input id="workspace-name" type="text" name="workspace" value="easyTranslator.fmw" /> 
</form>
```

4) Modify the File List Section

Next, in the body section (line 20-22), we have an area to display the files uploaded to the FME Server. In this exercise, we are uploading files to the FME Server. It is important to note what files have been uploaded. Currently, the FME Server REST API can only handle one file at a time. So, it is best to ensure that only one file has been uploaded.

On line 21, we need to specify the input type for the input. The type should be "button".

```html
<header>File List </header>(Be patient, some files may take a while to upload) :  
<input id="refresh" type="button" onclick="getFiles();" value="Refresh File List" />
<div id="filelist"></div>
```
This will create a button to refresh the file list on the FME Server. Once, the button is clicked, it activates the getFiles(); function which we will be creating in a later step. After the button we have an empty div which will contain a list of the files uploaded to the FME Server.

5) Review lines 24 - 28

On line 24 of the file, we have an empty form with the id of options. This is blank currently, however we will use the buildOptions function later in the exercise to populate the form with workspace parameters.

On line 26, there is a button which the user will press to run the workspace with the data uploaded to the FME Server.

On lines 27 and 28 we have two empty divs where the results will appear.

6) Add your FME Server Credentials to FMEServer.init

When the window is loaded, there are a few functions that are initiated. Before any of these functions can begin we have to enter our FME Server URL and our token (line 35 and 36). If you are using a training machine you may use http://localhost.

If you are using your own FME Server and are having issues with connecting your server, ensure that your server URL is in the format http://yourserver. Issues can occur if you have a trailing statement like http://YOURSERVER/fmeserver.

```javascript
<script type= "text/javascript">
var jsid, path, fileInput, files, archives, repository, workspace;
window.onload = function() {

FMEServer.init({
  server: "<YOURSERVER>",
  token: "<YOURTOKEN>"
});

// Initialize variables
setWorkspace();

// Generate a JSID for the session
FMEServer.getSession(repository, workspace, setVars);

// Get options for the workspace
generateOptions();
};
```

This section of the code gives an overview of the steps taken to get information about the workspace before it is run. First, we connect to the server. Then we set the workspace based on the user's input, we generate a session id or JSID, and then we generate the parameters for the workspace.

7) Add the repository and workspace ID to the Set Workspace function

The first function called is the setWorkspace function. This section gets the information that the user input in the form to create variables for the repository and workspace.

On line 50 replace //Add repository ID with repository-name. On line 51 replace //Add workspace ID with workspace-name.

```javascript
function setWorkspace() {
repository = document.getElementById("repository-name").value.trim();
workspace = document.getElementById("workspace-name").value.trim();
}
```

8) Review the getSession function

Next, we call a function in the JavaScript 1.2 Library, `getSession` (line 43). This function requires the repository, workspace, and a callback to run. We need the getSession function to generate a random number from the server. This keeps the session unique and prevents multiple users from overwriting the same files. This random number is referred to as the JSID in the other functions. Then JSID is set as the namespace in the data upload function. If namespace specified, uploads are placed in a folder named after the random number generated by the FME Server. If not specified, uploads are placed in a folder named after the name of the uploading user. For more information please review the Data Upload Documentation.
9) Go to the webpage to review the session ID

Next, we can see the session ID that was created from the getSession and test that it is refreshing properly. In the setVars function which is the callback from the getSession function, there is console.log (line 62). This will log the JSON that was returned from the FME Server. This JSON will contain a number for the session. If we refresh this page the number should change. This number is used to create a folder in the Server with the Data Uploaded.

Now we can save our myFourthApp.html file.

Open a web browser and enter: http://localhost:8000/myFourthApp/myFourthApp.html

Right click on the page, click on Inspect. This will open the developer tools dialog.

In the developer tools dialog, click Console, then open the Object drop down menu. Here, we should see a Session ID.

Then, if we refresh the page this number should be updated.

Once, the page has been refreshed, we can see what else has been returned from the FME Server. Find path by expanding serviceResponse. Then, click the folder tab to find the path. This path is where the file uploaded to the FME Server will go to when using this application.
10) Modify the `setVars` function

The `setVars` function (line 61-69) finds the session from the response which will be a string and will be set in the JSID variable. The path is the location of the file to the server. In the last step we found the session from the Console. It was located inside the `serviceResponse`, under `session`.

Therefore modify line 64 to read: `jsid = json.serviceResponse.session;`

```javascript
function setVars(json) {
    console.log(json);
    if (json.serviceResponse.files) {
        jsid = json.serviceResponse.session;
        path = json.serviceResponse.files.folder[0].path;
    } else {
        showResults(json);
    }
}
```

11) Review the `generateOptions` function

`generateOptions` (line 54-59) is the third function that is activated as soon as the window loads. This function uses the `setWorkspace` function to set the variables. The `generateOptions` function finds all the parameters in a workspace and sends them to a callback function. In our case, the callback function will be called `buildOptions`.

12) Modify the `buildOptions` function

The `buildOption` function uses `generateFormItems` (line 73) to create a form with the parameters seen in the previous exercises.

However, now the parameters are more complex because we need add a button for the file input.

On line 76-77, we begin by getting all of the inputs created in `generateFormItems`. Then, setting `added` equal to false. This is indicating that the button has not been created yet.

On line 79 we start a for loop to loop through the inputs to check if there is a file input.

On line 80 we need to make a modification to the code. We would like to add the button if, we have a file and the button has not been added already.

Change `inputs[i].type == "// Write a file type here"` to read `inputs[i].type == "file"`

On line 82 we need to modify the code to specify what element we would like to add. In our case we would like to add an input element.

Change `var button = document.createElement("//Write the element type here ");` to `var button = document.createElement("input");`

The rest of the loop specifies that we are creating a button with a value of "Upload File". Once the button is clicked it will initiate the upload file function. The button is added and therefore the added value is set to true. This will prevent another button from being generated.

```javascript
function buildOptions(json) {
    // Use the API to build the form items
    FMEServer.generateFormItems("options", json);
    // Attach the upload button to the form file input
    var inputs = document.getElementById("options").getElementsByTagName("input");
    var added = false;
    for (var i in inputs) {
        if (inputs[i].type == "file" && added === false) {
            fileInput = inputs[i];
            var button = document.createElement("input");
            button.type = "button";
            button.value = "Upload File";
            button.setAttribute("onclick", "uploadFile();");
            fileInput.parentNode.insertBefore(button, fileInput.nextSibling);
            added = true;
        }
    }
}
```

13) Modify the `uploadFile` Function
Next, we have the `uploadFile` function (lines 92-97). This function will upload the file selected to the FME Server. We need to modify the `dataUpload` function to include the right parameters.

To find the function in the documentation visit the FME Server JavaScript API. This page was linked at the beginning of our HTML.

Once you are on the FME Server JavaScript API page, use Ctrl + F to find the `dataUpload` function.

```javascript
function uploadFile()
{
    setWorkspace();
    // Ask FME Server to upload the file
    FMEServer.dataUpload(repository, workspace, fileInput, jsid, processFiles);
}
```

The `dataUpload` function documentation can be found here.

### 14) Review the `getFiles` function

Once the user clicks the "Refresh File List" button, the next function (`getFiles`) will be activated. This function is located on lines 99-102.

This activates the FME Server JavaScript function `getDataUploads`.

14.8 Exercise 18- Data Uploads

This function is similar to the structure of the previous call. It just checks the server using the JSID to see what files have been uploaded.
15) Modify the processFiles function

Next, we can use the function `processFiles` which is the callback for the `getFiles` and `uploadFiles` function. The `processFiles` function displays the file name and size in the `fileList` div.

On line 105, we need to get the div `fileList` so we can modify the code to say:

```javascript
var list = document.getElementById("fileList");
```

```javascript
function processFiles(json) {
    var list = document.getElementById("fileList");
    if (json.serviceResponse !== undefined) {
        list.innerHTML = "";
        files = json.serviceResponse.files.file;
        for (var file in files) {
            list.innerHTML += "<p>", files[file].name, "", files[file].size, " bytes</p>";
        }
        archives = json.serviceResponse.files.archive;
        for (var archive in archives) {
            list.innerHTML += "<p>", archives[archive].name, "", archives[archive].size, " bytes</p>";
        }
        showResults(json);
    } else {
        setTimeout("getFiles();", 2000);
    }
}
```

This function takes the server responses from the previous functions and sends it to the `showResults` function. `showResults` displays what file was loaded and the file size.

Once the user has entered their parameters they can click the Run Workspace with Data function. This activates the `runWorkspace` function.

16) Review the runWorkspace function

```javascript
function runWorkspace() {
    if (files !== undefined || archives !== undefined) {
        if (archives !== undefined) {
            files = archives;
        }
        setWorkspace();
        var params = {
            filename: fileInput.name,
            files: files,
            params: processParams()
        }
        // Ask FME Server to run the workspace with the uploaded data
        FMEServer.runWorkspaceWithData(path, params, showResults);
    } else {
        alert("No Files Uploaded. Please upload a file.");
    }
}
```

`runWorkspace` (line 152-169) checks if the files are present currently or in the archives. If the files are in the achieved variable, they are set to the files variable.
From here the set workspace function is called which is already in our application.

Next, in the function, it sets up the parameters to run the workspace, the filenames and files are input. However, the params require a new function called processParams, which we will be editing in the next step.

Finally, we use the runWorkspaceWithData function to run the workspace. The path was set in the setVars function. The params are from the process params function found in the next function. Then, the callback function is in showResults.

17) Modify the processParams function
The processParams function (lines 123-150) gets the parameters from the options form and then sets the variables in a format the FME Server will understand.

First, we need to get the selects and inputs from the form created from the generateFormItems functions.

To do this we need to modify the processParams function on lines 124 and 125.

Here, we need to first get the options form where the parameters are held.

Change line 124 and 125 to say:

```
var inputs = document.getElementById("options").getElementsByTagName("input");
var selects = document.getElementById("options").getElementsByTagName("select");
```

If we needed to find where the parameters are kept we need to Inspect the page and find the form id="options".

Next, we need to get the inputs and selects from the options form. We are going to use.getElementsByTagName and writing select and input in the quotations. The output results should look like this.

```
var inputs = document.getElementById("options").getElementsByTagName("input");
var selects = document.getElementById("options").getElementsByTagName("select");
```

The full function will look like this:

```
// Convert HTML NodeList types to regular array types
inputs = Array.prototype.slice.call(inputs);
selects = Array.prototype.slice.call(selects);

// Merge the regular arrays
options = inputs.concat(selects);

for (var opt in options) {
    var option = options[opt];
    if (option.value && option.name != fileInput.name && option.type != "button") {
        properties += option.name + "=";
        if (option.type == "select") {
            properties += option[option.selectedIndex].value;
        } else {
            properties += option.value;
            properties += "&";
        }
    }
}
properties = properties.substr(0, properties.length - 1);
return properties;

The first section of the function sets the variables for the inputs, selects, options, and properties. Inputs and selects are turned into arrays, and the arrays are merged together into the options array. After this, we have a for loop.

This for loop scans the options. Then it formats the options, so they are in the proper notation for the params variable. The first if statement, checks to see if the option name and value do not equal the file input name. This indicates that the option is not the file. Then, it checks that the option type is not a button. If this is true, the parameter is an input box, and the properties and the option name are added in the proper format. If the option type is select, then the selected option value is added in the proper notation.

Then, in the properties the trailing, ampersand (&), is removed. After this function is run to properly set up the parameters. These parameters can now be used in the runWorkspace function.

18) Review the showResults function

The showResults (line 171-187) builds the call with the path and parameters and then puts the information in the showResults function.

This function creates a new div and presents the result to the user.

If there is a serviceResponse and URL, we know it contains the download link. So the function will create a button that links to the URL for the data download.
If not, then the program will still print out the JSON to the user under the heading results.

19) Run the Program!

Select the Choose File button.

Select the KML located here:

C:\FMEData2018\Data\Boundaries\VancouverNeighborhoods.kml

Then select Upload File.

Repository: Samples
Workspace: easyTranslator.fmw

**File List** (Be patient, some files may take a while to upload): [Refresh File List]

[1] Source File: Choose File | No file chosen | Upload File
[2] Source Format: “Have FME guess from the file extension”

Input Coordinate System

Destination Filename: translated_data

Destination Format: Autodesk AutoCAD DWG/DXF

Output Coordinate System

Run Workspace With Data

Under **Input Coordinate System**: write LL84
Under **Output Coordinate System**: write LL84

Then select Run Workspace with Data.
After Run Workspace with Data has been selected, we can click the Download Result function.

Repository: Samples
Workspace: easyTranslator.fmw

**CONGRATULATIONS**

By completing this exercise you have learned how to:
- Create an application to translate data
- Upload a file using the FME Server JavaScript API
- Use the getSession and generateOptions functions
This exercise begins from the last exercise and displays how you can modify these exercises to create an application specific to your needs. In this example, you are given a workspace and two tester files to work with and create a data validation app. In the project file we recently uploaded to the FME Server, it included an additional workspace for this exercise. We are now going to customize the web page for data validation.

**WARNING**

This exercise uses completed workspaces imported from the project folder that was uploaded to the FME Server during Exercise 17. If you did not do this, you may upload the workspaces located in the myFifthApp folder. Then, ensure the guest account has access to the repository. Additionally, please complete Exercise 14 before completing this.

1) Open the myFifthApp.html file

Go to C:/FMEData2018/Resources/RESTAPI/myFifthApp then right click on myFifthAppBegin.html and select Edit with Notepad++.

2) Review the head section

In the head section (lines 5-15). Here, like the past exercises, there are connections to the FME Server Example CSS and the FME Server JavaScript.

There are additional connections to ArcGIS JavaScript and the CSS file.

3) Review the body section

In the body (line 17-67) of the HTML file there is a brief explanation of what the application does.

In this exercise we are using the previous application as a template and adding functionality to it to meet the needs of a Data Validation application.

Lines 24-36 explain the purpose of the application. A user can upload a file to the FME Server. If the user wants, they can display the results on a map on the page. Or they can download the results and see what sections need to be fixed.

This will allow the user to check their data on the web map. Errors within data will be highlighted in red. The user can then click Download Results to receive a Data Validation report.

After the instructions we need to create buttons to activate the functions in the JavaScript.

Lines 37-42 are from the last exercise and serve the same function. On line 38, there is a button created so users can view the files previously uploaded to the FME Server. This function is called getFiles. Below, the button to getFiles, there is an empty div for the fileList (line 39). This div will be populated with the files uploaded to the server in the current session. After the fileList div, there is an empty form that will list the user parameters in the workspace (line 41). Line 43 contains a button which will activate the streamResults function. This will display the results of the Data Validation on the map on the website. On line 44 there is a button to download the results. Here, the results are returned to the user through a URL. The URL will contain a report on the errors present in the AutoCAD file.

After this, we have a div for the ArcGIS map which will be added to the website (line 53). Then, we have a div where the results of the translation will be displayed (line 57-59)
4) Review the beginning of the JavaScript

This exercise contains many components from previous exercises. However, new components have been added to specialize this application for Data Validation and to add extra functionality.

Line 69, has the same variables as the last application. However, line 71 has additional variables. These are the variables required to stream data to the ArcGIS map. These variables are the map and the layer.

5) Modify the window.onload function

The window.onload function (line 73-106), is similar to the last exercise however, additional code is added for the ArcGIS Maps components. On line 75-78 there are modules imported from the ArcGIS JavaScript, these will allow us to enable a web map.

Lines 81-85 of the initialize the FME Server. In the previous exercise we could use http://localhost for the Server URL. However, now because we are using ArcGIS Maps we need to use our Public IP Address.

To find your Public IP Address, visit What Is My IP. Find your Public IP and add it to the Server section on line 83. Then, on line 84 update the token to your fmetoken.

```javascript
FMEServer.init({
    token: "725246dd2a6b8701e2e269006825888e5a6bf09" //Update to your fmetoken
});
```

Please note the current IP Address and token are not valid. Please update to your own training machine.

Line 87-94 are the same as the previous exercise. The setWorkspace function (line 87) is used to get the variables for the Repository and Workspace. Next, the getSession function (line 91) is used to generate a JSID for the session. The JSID is used keep each data upload unique. The JSID is a randomly generated number. The data will be uploaded to a folder with this number. This prevents users uploading a file with the same name from overwriting previous files. Next, the generateOptions function is used, this function will get the Public Parameters in the workspace and display it for the user.

Next, the map is initialized (lines 96-103).

We now have to specify that we are using the "streets" basemap on line 98. We are centering the map where the AutoCAD file will be centered. If we centered the web map in the wrong location, the data would appear to have never loaded. The zoom should also be set to a value appropriate to the data. If the web map is too far zoomed in, the user may miss important data.

On line 98 set basemap: "streets",

Then on line 99 change center to center: [-97.650, 30.299],

Finally on line 100 set zoom: 14,

The results will look like the JavaScript below.

```javascript
map = new Map("mapDiv", {
    basemap: "streets",
    center: [-97.650, 30.299],
    zoom: 14,
    minZoom: 5,
    smartNavigation: false
});
```

6) Modify the setWorkspace function

The setWorkspace function (line 108-113) is different from the setWorkspace function in the previous exercise. In the original, the values were set by a form where the user could enter the workspace and the repository. However, in this application there is only one workspace and repository that the user is interested in. Therefore, we can hard code the values in. In this function the Public IP address is added as well. This Public IP will be used to create a data streaming link which will stream the data directly to the web map on our page. On line 110 set the repository to "WebApplication"; On line 111 set the workspace to "webapp.kml.fmw"; and finally set the server to your Public IP Address.

```javascript
function setWorkspace() {
    repository = "WebApplication";
    workspace = "webapp.kml.fmw";
    server = "http://18.206.217.227"; //Update to your Public IP address
}
```
7) Review the generateOptions function

The generateOptions function (line 115-121) is the same as the previous exercise. Here, setWorkspace is called to get the repository and the workspace. Then getWorkspaceParameters is used to get the published parameters from the workspace. The published parameters are sent to the callback function, buildOptions.

8) Review the buildOptions function

The buildOptions function (line 136-158) is the same as the previous exercise as well. generateFormItems is used to create inputs for the user to fill the published parameters. Then, a loop is used to create the button for the user to upload their data.

9) Review the uploadFile function

The uploadFile function (line 160-166) is the same as the previous exercise, the dataUpload function is used to upload the file to the FME Server. The file is uploaded to a folder named after the JSID. The JSID is a randomly generated number assigned when the getSession function has run.

10) Modify the streamResults function

The streamResults function (line 200-230) is new to this exercise. Its purpose is to stream a new layer to the web map. This layer is created in the workspace webapp.kml.fmw.

On line 202, the function begins by checking if there is a layer present already in the web map. If this is the case we would like to remove it. On line 204 the layer is removed if it present.

Insert layer into the if statement it should now read if(layer)

```javascript
if (layer) {
  map.removeLayer(layer);
}
```

Next, the function using the classes the Esri KMLLayer and Parser. The if statements find if there are files are defined or if the archives are defined. If archives are defined then the archives are assigned to the files variable.

```javascript
if (files !== undefined || archives !== undefined) {
  if (archives !== undefined) {
    files = archives;
  }
}
```

After, this a data streaming link is created. This is using the server, repository, workspace, jsid, and filename variables.

On line 220 replace //jsid variable with jsid

```javascript
resultUrl = server + '/fmedatastreaming/' + repository + '/' + workspace + '?SourceDataset_ACAD=%24(FME_SHAREDRESOURCE_SYSTEM)%2Ftemp%2Fupload%2F' + repository + '/' + workspace + '%2F' + jsid + '%2F' + filename;
```

This workspace has one public parameter, this is the where the file is stored. The full path where the file is stored will be in the temporary upload folder. Inside the upload folder, there is another folder called repository, then workspace, then jsid. Inside all of these folders there will be the file the user uploaded. The resultUrl will be a functioning data streaming link we need to incorporate into the new Esri layer.

On lines 224-226 a new layer is created using the resultURL created on line 220.

```javascript
layer = new KMLLayer(resultUrl);
map.addLayer(layer, 1000);
layer.on('load', function() {[...]
```
The downloadWorkspace function (line 232-254), is very similar to the streamWorkspace function. It is different because it no longer has the Esri components required for streaming the KMLLayer directly to the map. Here the user, will receive a link they can click to open up a page with an HTML Report.

Lines 246-248 have been added to append the finalResults section to include the directURL.

12) Modify the newSession function

The newSession function (line 275-306) is used to create a new session for the user. Without this function, the user could upload one file, check if it is valid and get the validation report. However, if they tried to upload a new file, it would upload to the same folder in the FME Server. If a user wanted to check multiple files, FME Server would not know which one to select if they were stored in the same folder. This is why the newSession function is used. It essentially repeats the same steps in the beginning of the web application.

The first function that should be used is getSession.

To do this on line 277 replace:

FMEServer.//Function to create the new session (repository, workspace, setVars);

With:

```java
FMEServer.getSession(repository, workspace, setVars);
```

After, we have created a newSession we need to refresh the file list so the user will not see the files.

To do this on line 278 replace:

FMEServer.//Function to get the new files(repository, workspace, jsid, processFiles);

With:

```java
FMEServer.getDataUploads(repository, workspace, jsid, processFiles);
```

We have also included the processFiles function to process refresh the current fileList.

Once the user selects the newSession function the fileList should be empty and they should receive a new JSID.

13) Test Out the Application


Once on the webpage select Choose File. Then navigate to C:\FMEData2018\Resources\RESTAPI\myFifthApp\Data\bad_data and select the file. Then select upload file.

After, the file has been uploaded to the FME Server, select Display Results on Map. The errors in the AutoCAD file are displayed in red. Then, select Download Results.
The Download Results button will return an HTML page with the following format.
CONGRATULATIONS

By completing this exercise you have learned how to:
- Use an FME workspace to create a data validation app
- Upload a file using the FME Server JavaScript API
- Use the getSession and generateOptions functions
Course Wrap-Up

Although your FME training is now at an end, there is a good supply of expert information available for future assistance.
Product Information and Resources

Safe Software Web Site

The Safe Software web site is the official information source for all things FME. It includes information on FME products, Safe Software services, FME solutions, FME support and Safe Software itself.

Safe Support Team

Behind FME are passionate, fun, and knowledgeable experts, ready to help you succeed, with a support team philosophy built on the principle of knowledge transfer.

You can request product support through a Support Case (web/email) or using a Live Chat.

Your Local Partner

Safe Software has partners and resellers around the world to provide expertise and services in your region and your language.

You can find a list of official partners on the Safe Software Partners Page.
Safe Software Blog

The Safe Software blog provides technical information and general thoughts about FME, customers' use cases, and spatial data interoperability. It includes articles, videos, and podcasts.

FME Manuals and Documentation

Use the Help function in FME Workbench to access help and other documentation for FME Desktop. Alternatively, look on our web site under the Knowledge Center section.
## FME Desktop

**FME Desktop Administrator's Guide**
Find out how to install and license your version of FME Desktop, and perform other administrative tasks. (PDF Version)

**FME Transformer Reference Guide (PDF)**
A quick reference describing each transformer's functionality.

**FME Integration Console**
Find out how to extend your "FME-ready" third-party applications so they will integrate with FME Desktop.

## FME Readers and Writers
A detailed technical guide to the many reader and writer formats available in FME.

**FME Workbench Transformers**
A detailed technical guide to the many transformers available in FME Workbench.

**FME Quick Translator**
Use this tool to perform simple, automatic data conversions.

## FME Workbench
A guide to FME’s primary graphical tool for creating and running data transformations.

**FME Data inspector**
A guide to FME's graphical tool for inspecting transformation results and other datasets.
Community Information and Resources

Safe Software actively encourages users of FME to become part of the FME Community.

The FME Community

The FME Community is a one-stop shop for all community resources, plus tools for browsing documentation and downloads.

Knowledge Base

The FME Knowledge Base contains a wealth of information; including tips, tricks, examples, and FAQs. There are sections on both FME Desktop and FME Server, with articles on topics from installation and licensing to the most advanced translation and transformation tasks.

Forums

FME community members post FME-related messages, ask questions, and share in answering other users’ questions. Members earn “reputation” and “badges,” and there is a leaderboard of the top-participating users. Join the conversation to see how the community helps each other with their FME projects!

Ideas Exchange

FME development is very much user-driven. The Ideas Exchange gives users the chance to post their ideas for new FME functionality, or improvements to existing functionality, and allows everyone to vote on the proposed ideas. The more votes an idea gets, the more likely it is to be implemented!

The FME Channel

This FME YouTube channel is for those demos that can only be properly appreciated through a screencast or movie. Besides this, there are a host of explanatory and helpful movies, including recordings of most training and tutorials.
10.2 Community Information and Resources
Feedback and Certificates

The format of this training course undergoes regular changes prompted by comments and feedback from previous courses.

Course Feedback

Ricky Restless says...

There’s one final set of questions – and this time you’ll be telling me if the answers are correct or not!

Safe Software greatly values feedback from training course attendees and our feedback form is your chance to tell us what you really think about how well we’re meeting your training goals.

You can fill in the feedback form now, but you’ll also be reminded by email shortly after your course. Safe Software’s partners who carry out training may ask that you fill in a separate form, but you can also use the official Safe Software form if you wish.

Certificates

Ricky Restless says...

In order to prove you have taken this training course, a certificate will be emailed automatically to anyone who was logged on for the duration of Safe Software hosted courses.
Troubleshooting

There are many ways you can troubleshoot any errors that may occur.

FME Server Troubleshooting Guide

This article on the knowledge center contains all of the FME Server related troubleshooting guides which should help diagnose and resolve most issues encountered with FME Server.

FME Server REST API Error Codes

The FME REST Service returns an HTTP status code for every request. For most GET requests, a response message is returned in your requested format, along with the status code. For most PUT and DELETE requests, only the status code is returned to indicate whether the operation is successful or not. Refer to the specifications reference for more details. All of these codes can be found on the REST API documentation page.

<table>
<thead>
<tr>
<th>Response Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 OK</td>
<td>Success; the results are rendered in the response body.</td>
</tr>
<tr>
<td>201 Created</td>
<td>Success; the resource has been created.</td>
</tr>
<tr>
<td>202 Accepted</td>
<td>Success; the operation has been started.</td>
</tr>
<tr>
<td>204 No Content</td>
<td>Success; the response body contains no contents.</td>
</tr>
<tr>
<td>207 Multi-Status</td>
<td>Success; the response body contains the status of each sub-request.</td>
</tr>
<tr>
<td>400 Bad Request</td>
<td>The request was improperly formatted. Typically, this means that your JSON or XML is not well-formed. It may also mean that the URI is invalid; most path components can only contain alphanumeric characters.</td>
</tr>
<tr>
<td>401 Unauthorized</td>
<td>Authentication credentials are missing or incorrect.</td>
</tr>
<tr>
<td>403 Forbidden</td>
<td>The request is understood, but it was refused. An accompanying error message explains why.</td>
</tr>
<tr>
<td>404 Not Found</td>
<td>The URI requested is invalid, or the requested resource does not exist.</td>
</tr>
<tr>
<td>409 Conflict</td>
<td>You attempted to create a new resource that already exists, or made changes or created a resource that depends on another resource that does not exist.</td>
</tr>
<tr>
<td>415 Unacceptable Media Type</td>
<td>The content-type is not allowed. Typically, only &quot;application/xml&quot;, &quot;application/json&quot;, or &quot;application/x-www-form-urlencoded&quot; is permitted.</td>
</tr>
<tr>
<td>422 Unprocessable Entity</td>
<td>The request was well-formed, but cannot be performed due to semantic errors.</td>
</tr>
<tr>
<td>503 Server Unavailable</td>
<td>Part or all of FME Server was temporarily unreachable.</td>
</tr>
</tbody>
</table>

Developer Tools

Most browsers have developer tools that anyone can use. Here is a link on how to debug JavaScript in Chrome, [Debugging Tutorial](#). However, most browsers should have this capability.
Thank You

Thank you for attending this FME training course.

All of us at Safe Software wish you fun and success as you use FME to speed your data on its way.