How To: Use the Ed-Fi API Bulk Load Services

The Ed-Fi ODS / API contains endpoints that allow client applications to send XML data files through the API for bulk loading. This is useful for a number of scenarios. Bulk loading is often the easiest way to populate a new instance of the ODS / API. In addition, some implementations only require periodic uploads from clients. Bulk loading is useful for these "batch loading" scenarios.

This article provides overview and technical information to help platform hosts and client developers use the bulk load endpoints.

Note that platform hosts have an alternate way of bulk loading files directly from disk (i.e., not through the API) using the Ed-Fi Console ODS Bulk Loader. See the article How To: Use the Ed-Fi Console ODS Bulk Loader for more information.

Downloads

The following link is the sample XML file used in the walkthrough below.

SampleStudent.xml

Platform hosts and client application developers may find this useful for testing their implementations.

Overview

A bulk operation can include thousands of records across multiple files to insert and update.

A few key points about the API surface worth understanding are:

- **Clients must post a representation of the files to be uploaded.** This includes the format, the interchange type, and the file size. We'll look at an example below.
- **Clients must send bulk data that conforms to the Ed-Fi Data Standard XML definitions.** See the Interchange Schema section of the Ed-Fi Data Standard documentation for details.
- **Error Handling.** Records are parsed and operations are executed as individual transactions. This means that one failing entity record will not fail the entire batch. The errors from individual transactions are logged. The error log can be inspected per bulk operation.
- **API Profiles do not impact bulk data loads.** Platform hosts can optionally implement API Profiles to create data policies for the API JSON endpoints. Those policies do not affect data sent through bulk load services.

Before we dive into the details, it's useful to understand the differences between the transactional operations of the API surface and the bulk load services discussed in this article. The following table provides a compare/contrast of the major differences:

Transactional API Surface	Bulk Load Services
JSON	Ed-Fi Data Standard XML
Synchronous responses	Asynchronous responses
Near real-time, as data is changing in client applications	For initial load or batch mode updates
Full range of create, read, update, and delete operations	Upsert (i.e., create and update) only
Create and retrieve UniqueIds	No ability to create or retrieve UniqueIds

Platform Setup and Testing

This section outlines the basics of setting up and testing bulk loading through the ODS / API surface.

- **Microsoft Message Queue (MSMQ).** Bulk load services work against Microsoft message queues, and the console workers share the same internal logic. The internal logic is covered by unit tests that verify the ability to process messages from one queue to the next.
- **Smoke Testing.** A "smoke test" is typically all that is required for these services. Platform hosts basically perform a bulk upload operation as outlined in this article, and verify that the data hits the ODS (either by inspecting the data tables directly or calling the API surface to search for the information loaded).
- **Troubleshooting.** When troubleshooting the services, the bulk worker and upload services can be temporarily stopped, and their associated message queues examined for unprocessed messages. Turning on the associated service should eventually clear the service's source queue. If there are messages building up in either of the queues, the problem is typically one of credentials, and an appropriate error will be in the event log. Proper credentialing of the services is covered in the deployment documentation. See, e.g., the sandbox deployment information and production deployment information in the Platform Developers' Guide.

Client Walkthrough Example

This walk-through demonstrates the sequence of operations clients use to load bulk data via the API. We'll use an XML file with student data as an example.

The high-level sequence of operations from the client is as follows:

Detail on each step follows.

Step 1. Create the Operation

POST a representation of the files to upload to `/bulkOperations`.

```
{
  "uploadFiles": [{
    "format": "text/xml",
    "interchangeType": "student",
    "size": 699
  }]
}
```

Create one `uploadFiles` entry for every file you're including. The `format` should always be "text/xml", `interchangeType` should be the type of interchange, and `size` is the total bytes of the file you're uploading. You can easily get the file size by using new `FileInfo(filePath).Length` or using the `Length` property of the file stream if you're opening a file stream to send it up.

Sample Response (should have status code of `201 Created`):

```
HTTP/1.1 201 Created
Content-Length: 290
Content-Type: application/json; charset=utf-8
Location: http://localhost:54746/api/v2.0/2016/BulkOperations/1b8e6786-
53ef-4ec0-9ee1-2d1194e1374c
Server: Microsoft-IIS/10.0
X-SourceFiles: =?UTF-8?B?
QzpcR2l0XFBlcnNvbmFsXEVkRmlBbGxpYW5jZVxFZC1GaS1PRFMtSW1wbGVtZW50YXRpb25cQXB
wbGljYXRpb25cRWRGaS5PZHMuV2ViQXBpXGFwaVx2Mi4wXDIwMTZcYnVsa09wZXJhdGlvbnM=?=
X-Powered-By: ASP.NET
Date: Wed, 22 Jun 2016 18:51:57 GMT

{
  "id": "1b8e6786-53ef-4ec0-9ee1-2d1194e1374c",
  "uploadFiles": [
    {
      "id": "1b9e9c6d-56b1-4489-b485-454528b18602",
      "size": 699,
      "format": "text/xml",
      "interchangeType": "student",
      "status": "Initialized"
    }
  ],
  "status": "Initialized"
}
```

From the response, you can obtain the overall operation id (the root `id`), as well as individual fileIds for each file to be uploaded that will be used as part of uploading.

Step 2. Upload XML Files

For each file to upload, take the returned fileId and then submit the file as one-to-many "chunks." Each chunk of the file can be up to 150MB. For the attached example file, it can be submitted as a single chunk for simplicity.

POST the file to `/uploads/fileId/chunk?offset=offset&size=size`, where `fileId` is the value returned from creating the bulk operation, `offset` is the current offset in the file starting with 0, and `size` is the actual size of the chunk being uploaded. This POST must be submitted as `multipart/form-data` with the binary data streamed along in the body. An easy way to do this correctly is to use (or deconstruct) the code provided in the generated SDK for the UploadsApi, as it will handle submitting the appropriate headers and data.

The following is an example `HttpRequest` with Headers and embedded XML:

```
POST http://localhost:54746/api/v2.0/2016/uploads/1b9e9c6d-56b1-4489-b485-
454528b18602/chunk?offset=0&size=699 HTTP/1.1
Authorization: Bearer ea8110623bcb478c917aa30c7d65e392
Accept: application/json, application/xml, text/json, text/x-json, text
/javascript, text/xml
User-Agent: RestSharp/105.2.3.0
Content-Type: multipart/form-data; boundary=-----------------------------
28947758029299
Host: localhost:54746
Content-Length: 965
Accept-Encoding: gzip, deflate

-------------------------------28947758029299
Content-Disposition: form-data; name="1b9e9c6d-56b1-4489-b485-
454528b18602"; filename="1b9e9c6d-56b1-4489-b485-454528b18602"
Content-Type: application/octet-stream

<?xml version="1.0" encoding="UTF-8"?>
<InterchangeStudent xmlns="http://ed-fi.org/0200" xmlns:ann="http://ed-fi.
org/annotation" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
schemaLocation="http://ed-fi.org/0200 ../../Schemas/Interchange-Student.
xsd">
    <Student id="TEST_STUDENT_68">
        <StudentUniqueId>68</StudentUniqueId>
        <Name>
            <FirstName>Student</FirstName>
            <LastSurname>Sixty Eight</LastSurname>
        </Name>
        <Sex>Male</Sex>
        <BirthData>
            <BirthDate>1969-06-09</BirthDate>
        </BirthData>
        <HispanicLatinoEthnicity>false</HispanicLatinoEthnicity>
    </Student>
</InterchangeStudent>

-------------------------------28947758029299--
```

The expected response is a status code of `201`, with no body.

You would repeat this process until the entire file has been uploaded, adding the size of the chunk to the offset value for each subsequent upload. For example, submitting two 300-byte chunks, the first `offset` would be 0, the second would be 300, and both would have a `size` of 300.

The following is example code for handling a large file:

```
int offset = 0;
int bytesRead = 0;
var buffer = new byte[3 * 1000000];

this.Logger.DebugFormat("Uploading file {0}", filePath);
using (var stream = File.Open(filePath, FileMode.Open, FileAccess.Read))
while ((bytesRead = stream.Read(buffer, 0, buffer.Length)) != 0)
{
    if (bytesRead != buffer.Length)
    {
        var newBuffer = new byte[bytesRead];
        Array.Copy(buffer, newBuffer, bytesRead);
        buffer = newBuffer;
    }

    // Submit over to the sdk uploadApi for upload
    var response = uploadApi.PostUploads(new Upload
    {
        id = fileId,
        size = bytesRead,
        offset = offset,
        fileBytes = buffer
    });

    offset += bytesRead;

    if (response.StatusCode != HttpStatusCode.Created)
    {
        this.Logger.DebugFormat("Error uploading file {0}.", uploadFile.
FilePath);
        break;
    }

    this.Logger.DebugFormat("{0} bytes uploaded.", offset);
}
```

Step 3. Commit the Upload

For each file, after finishing the upload, take the `fileId` and commit the upload.

POST to `/uploads/fileId/commit` where `fileId` is the same fileId that was uploaded to. The expected response is a `202 Accepted` with no body.

Step 4. Check Status

At this point, the bulk operation is completed, and will be processed on the server asynchronously. Once the commit command is received, the operation is pushed to a queue that will trigger the actual processing. Status can be checked at any time by performing a GET to `/bulkoperations/bulkOperationId`, where `bulkOperationId` is the `id` sent back from the original creation of the operation.

On a happy path, after committing all the files, the `status` should be Started, such as this example:

```
{
  "id": "1b8e6786-53ef-4ec0-9ee1-2d1194e1374c",
  "uploadFiles": [
    {
      "id": "1b9e9c6d-56b1-4489-b485-454528b18602",
      "size": 699,
      "format": "text/xml",
      "interchangeType": "student",
      "status": "Started"
    }
  ],
  "status": "Started"
}
```

Once the operation is done processing, the `status` should be Completed, such as this example:

```
{
  "id": "1b8e6786-53ef-4ec0-9ee1-2d1194e1374c",
  "uploadFiles": [
    {
      "id": "1b9e9c6d-56b1-4489-b485-454528b18602",
      "size": 699,
      "format": "text/xml",
      "interchangeType": "student",
      "status": "Completed"
    }
  ],
  "status": "Completed"
}
```

If any of the data elements don't load correctly, the `status` will come back as Error such as this example:

```
{
  "id": "d3b18de4-1f1b-482c-802a-0ba9b71bbf8f",
  "uploadFiles": [
    {
      "id": "83bafffe-377a-4962-844e-88d0d3fcf5e9",
      "size": 699,
      "format": "text/xml",
      "interchangeType": "student",
      "status": "Error"
    }
  ],
  "status": "Error"
}
```

An Error status doesn't necessarily mean every record failed to load. To see which parts failed to load, you can perform a GET against `/bulkoperations/operationId/exceptions/fileId?offset=0&limit=50` to get 50 exceptions per file at a time. You can adjust the offset and limit to page through all the exceptions until you've received them all.

Further Information

This section contains a few additional resources related to bulk loading through the API:

- **Bulk Operation Endpoint Documentation.** The endpoints used in the example above are documented on the Ed-Fi API sandbox instance at https://apidocs.ed-fi.org/. See the "Other" API section.
- **API Client Boot Camp Documents.** The Ed-Fi Alliance hosted a "boot camp" training session for API Client developers that included a walkthrough of bulk loading. An instructional overview and training materials are available online here.
- **Ed-Fi Tracker / JIRA.** The Ed-Fi Alliance's issue tracking system is a good resource for fine points and troubleshooting specific implementation issues. See, e.g., the discussion on Tracker ticket ODS-820.
- **Deployment Documentation.** The Deployment section in the Platform Developers' Guide has additional information that platform hosts and operations teams may find useful.