GREAT STORIES
START HERE

HxGN LIVE
LAS VEGAS, NV
13-16 JUNE 2017
Introduction to ERDAS APOLLO REST API

Stan Tillman

ERDAS APOLLO Product Manager
Agenda

1. Introductions and Logistics
2. What is ERDAS APOLLO?
3. ERDAS APOLLO APIs
4. Working with Swagger
5. Documentation and Support
6. Questions
Introductions

Please introduce yourself to the class by answering the following questions:

- **Name:** Stan Tillman
- **Company:** Hexagon Geospatial
- **Position:** ERDAS APOLLO Product Manager
- How long have you used ERDAS APOLLO?
- What do you hope to gain from the course?

- There will be one break at the two hour mark in the course. If you need a break before or after the scheduled break please take one, but be courteous and do not disturb others.

- Please turn off or mute all cell phones / mobile devices.

- In case of emergency, please exit the room in an orderly manner and obey all instructions from convention center staff and security.
What is ERDAS APOLLO?
What is ERDAS APOLLO?

ERDAS APOLLO is an enterprise level application for comprehensive data management, analysis, and delivery.

Organizations can use ERDAS APOLLO to:

- Describe
- Catalog
- Search
- Discover
- Process
- Securely disseminate

Massive volumes of data
What is ERDAS APOLLO?

Data Management
- Catalog
- Security
- Services
- Style

Analysis

Visualization

Feature Edits / Update

Execute online analytics

ERDAS APOLLO Catalog and Server

DATA
- Raster
  - TIFF, JPG2000, etc.
- Digital
  - PDF, WMV, DOC
- Third Party
  - OGC Services
- LiDAR
- Vector
  - Oracle, FGDB, GML

SERVICES
- OGC
- Tile Cache
- ECWP
- File

What is ERDAS APOLLO?
What is ERDAS APOLLO?

• Utilizes a geospatial enterprise data catalog (GEDC)
• Central repository of individual or integrated geospatial datasets from one or more sources within an organization
• Stores current and historical data
• Users can query to conduct simple to complex searches for geospatial enabled data to be used in analysis or creating reports

• Cataloged data can be from various groups within an organization, such as planning, mapping, operations, etc.
• Data may be freely reviewable or distributable
• Other data may have restrictions and distribution rules that must be followed

• GEDC management is equally important as any large business enterprise system
• Data must be secured but easily locatable and accessible inside the organization
• Data must be locatable, usable, and distributable outside the organization
ERDAS APOLLO SDK

ERDAS APOLLO Solution Toolkit, a Java-based SDK, has been used traditionally to extend and customize ERDAS APOLLO.

ERDAS APOLLO Solution Toolkit
• Though useful, it is no longer up to date with today’s developer needs
• Needed Java programming experience
• All operations were done via Function Calls as opposed to simple HTTP requests
• Not compatible with many existing applications
• HTML based documentation was incomplete and not always up-to-date

ERDAS APOLLO RESTful APIs
• Simpler to use
• Easier to integrate into existing software packages
• Newer and more modern way of programming and interacting with resources over the web
• Better documentation:
  • Up-to-date
  • Easy to navigate
  • Sandbox to try various requests
Out with the Old…..

The Create Aggregate functionality provides a way to create a new catalog entity of an aggregate type on the root node of the tree.

```java
/**
 * Creates a new aggregate in the catalog.
 * @param instance the instance to be persisted.
 * @param parentId the id of the parent.
 * @param permissions the permissions for the new instance.
 * @return the new instance with a primary key.
 */
Aggregate create(Aggregate instance, String parentId, Map<String, Permission> permissions);
```

In with the NEW!

The Add Dataset functionality provides a way to add new catalog entities of dataset type in the provided parent aggregate node in the catalog tree. The method starts an asynchronous process that will create new dataset nodes in the provided parent aggregate inside the crawler options object.

```java
/**
 * @param fileNames the files to be added to the catalog.
 * @param config the processing parameters for the job.
 * @return the asynchronous job (currently the jobId lets say ) that was started.
 */
String addDatasets(Collection<String> fileNames, 
CrawlConfig config, Map<String, Permission> permissions);
```
ERDAS APOLLO API – RESTful API

ERDAS APOLLO API difference
Now it is a RESTful API that uses HTTP requests to GET, PUT, POST, PATCH and DELETE data.

This means the exposed ERDAS APOLLO web services are better integrated into various applications by allowing only those modules needed for a customer’s particular needs or workflows to be integrated.

ERDAS APOLLO API, like any other API, allows for interaction in the same way that a waiter acts as an interaction (API) between the customer and the kitchen.

As such, ERDAS APOLLO API allows for interaction between the server and an application.

We’ll discuss some of the various modules exposed through the ERDAS APOLLO RESTful API and how to work with them.

API – Application Program Interface
REST – Representational State Transfer
ERDAS APOLLO Catalog Service APIs – Overview

ERDAS APOLLO Catalog Service APIs Categories

- General Catalog
- Aggregates
- Images
- Point Clouds
- Vectors
- Generic Data
- Videos
- Delete Operations

- Crawl Operations
- Jobs Management
- Pyramids
- Thumbnails
- Services
- Processes
- Services-Vector
- Contacts

Planned APIs

- Move, Reparent
- WMS Context Service
- Photogrammetric Images
- Radar Data
Entire Catalog – not specific to an entity type

Search
- Advanced Search across the entire catalog
  
  POST /content/catalog/search.json

- Get all the catalog items – can be filtered by various query parameters
  
  GET /content/catalog/items.json

Actions on specific catalog items
- Delete a specific catalog item
  
  DELETE /content/catalog/items/{ID}.json

- Retrieve a specific catalog item
  
  GET /content/catalog/items/{ID}.json

- Update a specific catalog item
  
  PATCH /content/catalog/items/{ID}.json

Search allows for querying ERDAS APOLLO catalog through:
- a spatial search
- a catalog parameter search
- or both

Once items are located in the catalog, specific information can be:
- viewed
- updated
- deleted
Entire Catalog – not specific to an entity type

Actions on attachments for a specific catalog item

- Deleting an attachment for a specific catalog item
  
  DELETE /content/catalog/items/{ID}/attachment/{attachmentName}

- Retrieve an attachment for a specific catalog item

  GET /content/catalog/items/{ID}/attachment/{attachmentName}

- Adds an attachment for a specific catalog item

  POST /content/catalog/items/{ID}/attachment/{attachmentName}

Actions on specific property for a specific catalog item

- Retrieves a property from a given catalog item

  GET /content/catalog/items/{ID}/{{propertyName}}.json

- Deletes a property from a given catalog item

  DELETE /content/catalog/items/{ID}/{{propertyName}}

- Creates or updates a property from a given catalog item

  POST /content/catalog/items/{ID}/{{propertyName}}

Additional information can be attached to a catalog item, such as external metadata.

Properties on cataloged items can be reviewed, updated, or deleted from the catalog.
Entire Catalog – not specific to an entity type

Actions on Classifying Concepts for a specific catalog item

- Delete a classifying concept from a specific catalog item
  
  DELETE /content/catalog/items/{ID}/classifyingConcepts

- Retrieve a classifying concept for a specific catalog item
  
  GET /content/catalog/items/{ID}/classifyingConcepts

- Add a new concept that classifies a specific catalog item
  
  POST /content/catalog/items/{ID}/classifyingConcepts

Actions on Classified Objects for a specific catalog item

- Delete a classified object from a specific catalog item
  
  DELETE /content/catalog/concepts/{ID}/classifiedObjects

- Retrieve a classified object for a specific catalog item
  
  GET /content/catalog/concepts/{ID}/classifiedObjects

- Add a new object that classifies a specific catalog item
  
  POST /content/catalog/concepts/{ID}/classifiedObjects

When a Classification Scheme (root of a taxonomy hierarchy-ex. lasClassifications) is defined to extend a catalog record, Classifying Concepts allow for interacting with concepts associated to a given catalog item.

When a Classification Scheme (root of a taxonomy hierarchy-ex. lasClassifications) is defined to extend a catalog record, Classified Objects allow for interacting with catalog items associated with a given concept.
Roles and Rights Management

- Retrieve the rights for a specific catalog item
  
  ```python
  GET /content/catalog/items/[ID]/rights.json
  ```

- Create or modify the rights for a specific catalog item
  
  ```python
  PUT /content/catalog/items/[ID]/rights.json
  ```

- Get a summary of rights for the current user of the catalog
  
  ```python
  GET /content/catalog/items/[ID]/rights/currentuser.json
  ```

- Delete a principle’s rights to a specific catalog item
  
  ```python
  DELETE /content/catalog/items/[ID]/rights/[principleName]
  ```

Restrictions to access of cataloged items can be set and expanded to any security role defined in the ERDAS APOLLO system.
Aggregates

- Get the specific catalog item for the given entity type
  
  ```
  GET /content/catalog/aggregates.json
  ```

- Create a new instance of the given entity type
  
  ```
  POST /content/catalog/aggregates.json
  ```

- Get a specific catalog item for the defined entity type
  
  ```
  GET /content/catalog/aggregates/{ID}.json
  ```

- Update properties on a specific catalog item for the given entity type
  
  ```
  PATCH /content/catalog/aggregates/{ID}.json
  ```

Aggregates - a collection of individual datasets and records, much like a folder in Windows Explorer.

Aggregates can created, searched, reviewed, updated, and deleted like any other record in the catalog.

Deleting an Aggregate must be done through the Delete job or using the generic Items API. Deletes will also delete any sub-aggregate or records within that aggregate.
Images, Point Clouds, Vectors, Generic Data, and Videos

- Get the specific catalog item for the given entity type
  - Get /content/catalog/{entity type}.json
    ex. GET /content/catalog/datasets/images.json
- Create a new instance of the given entity type
  - Post /content/catalog/{entity type}.json
    ex. POST /content/catalog/datasets/images.json
- Get a specific catalog item for the defined entity type
  - Get /content/catalog/{entity type}/{ID}.json
    ex. GET /content/catalog/datasets/images/{ID}.json
- Update properties on a specific catalog item for the given entity type
  - Patch /content/catalog/{entity type}/{ID}.json
    ex. PATCH /content/catalog/datasets/images/{ID}.json

These APIs work with various records based on certain data types.

This is an indirect filter where the only applicable records are restricted to the given type.

The items are added and manipulated as records that still allow users to search for information on the item.

Deleting a specific type must be done through the Delete job or using the generic Items API.
## Operations and Jobs

### Delete Operation
- Delete one or more aggregates or catalog items from the catalog

```
POST /delete
```

### Crawl Operation
- Finds recognized data types on a file system or database and adds them to the catalog

```
POST /crawl
```

### Job Management
- Get the job ids for all jobs
  
  ```
  GET /jobs
  ```

- Delete a specific job
  
  ```
  DELETE /jobs/{jobId}
  ```

- Get the status of a specific job
  
  ```
  GET /jobs/{jobId}
  ```

- Cancel a specific job
  
  ```
  PUT /jobs/{jobId}/cancel
  ```

Major operations such as crawling data, updating data, and deletions are done through a jobs mechanism.

Delete operation - deletes the catalog record and any file created during the crawl.

Crawl operation - decodes the file, parses metadata, creates thumbnails, and creates pyramids.

Once an operation or job is running one can:
- retrieve status
- delete a job
- cancel a job
Auxiliary File Association

Pyramids

- Generates a pyramid file and updates one or more catalog items with the pyramid information
  
  POST /pyramids

- Deletes pyramid information from a specific catalog item
  
  DELETE /pyramids/{itemId}

- Add an existing pyramid file to the specific catalog item
  
  POST /pyramids/{itemId}

Pyramid layers of geospatial image datasets can be created, added, or deleted using Pyramids API.

Thumbnails

- Generates a thumbnail file and updates one or more catalog items with the thumbnail information
  
  POST /thumbnails

Thumbnails can be removed using the Delete Attachments method on the generic catalog item APIs.
Services and Processes

**Services**

- Get a list of all OGC services listed in the catalog
  
  ```
  GET /content/catalog/services.json
  ```

- Delete a service from the catalog
  
  ```
  DELETE /content/catalog/services/{ID}.json
  ```

- Get a specific OGC service from the catalog
  
  ```
  GET /content/catalog/services/{ID}.json
  ```

- Harvest properties of an external OGC service into the catalog
  
  ```
  POST /content/catalog/services.json
  ```

**Processes**

- Return a list of processes in the catalog
  
  ```
  GET /content/catalog/processes.json
  ```

OGC services such as WMS, WCS, WFS, etc.:  
- can be turned on and off  
- removed from defined datasets in the catalog

Processes API reviews all Spatial Models that have been published to ERDAS APOLLO server and catalog.
Vector Services

Services-Vector

- Get all legacy vector service names
  
  ![GET icon] /services/vector

- Add a legacy vector service
  
  ![POST icon] /services/vector

- Delete a legacy vector service
  
  ![DELETE icon] /services/vector/{providerName}

- Get a legacy vector service
  
  ![GET icon] /services/vector/{providerName}

- Update a legacy vector service
  
  ![PATCH icon] /services/vector/{providerName}

These endpoints represent the APIs specific to managing legacy vector services.

Note: Expect to see other types of data (such as images) represented here in the future.
Contacts

Contacts

- Return a list of contacts from the catalog
  
  **GET** /content/catalog/contacts.json

- Delete a specific contact from the catalog
  
  **DELETE** /content/catalog/contacts/[ID].json

- Return a specific contact from the catalog
  
  **GET** /content/catalog/contacts/[ID].json

The response to a GetCapabilities request to an OGC service contains several metadata properties including contact information.

Contacts APIs manage the contacts used in the ERDAS APOLLO created services.
Working with Swagger
Swagger and ERDAS APOLLO API

ERDAS APOLLO API Framework is implemented through an Open API RESTful Documentation Specification known as Swagger.

Using Swagger, the developer can input JSON requests that communicate with the ERDAS APOLLO server and receive the requested information back as JSON.

Open the Swagger documentation one of these ways:

- From ERDAS APOLLO main page http://<YourServer>/erdas-apollo. Click the API link / graphic.
- Type the URL in a browser - http://<YourServer>/erdas-apollo/api-docs
Swagger and ERDAS APOLLO API

You will see the following HTTP verbs (or methods) identifying how the content of the request should be organized:

- POST
- GET
- DELETE
- PATCH
- PUT

POST, PATCH, and PUT have similar formats

- Difference between PATCH and PUT involves what is sent in the body:
  - PATCH can send only a portion of the object
  - PUT must contain an update of the entire object
Swagger and ERDAS APOLLO API

Most methods share the following sections in the Swagger documentation:

- Implementation Notes (if notes are needed)
- Successful Status Code
- See the response in the form of either:
  - A Model with descriptions
  - Example values
- Response Type (Primarily “application/json”)
Swagger and ERDAS APOLLO API

All methods also contain a section for parameters. However, POST, PUT, PATCH typically show a json body whereas a GET describes each parameter separately.

For the JSON body, you can see either the Model (with descriptions) or Example Value – and you can autopopulate the Payload box by clicking in the Example Value box.

POST, PUT, PATCH
Swagger and ERDAS APOLLO API

Finally, all methods also contain the following documentation:

- Response message for all error codes – again, note that the format can be seen as a model or as an example value.

- **Try it out!** button -- Once either the JSON body or individual parameter fields have been populated, click this button to submit a request to the server.

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Reason</th>
<th>Response Model</th>
<th>Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>401</td>
<td>Unauthorized</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>error: &quot;string&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>error_description: &quot;string&quot;</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>Unexpected error</td>
<td>{</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>error: &quot;string&quot;,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>error_description: &quot;string&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Swagger and ERDAS APOLLO API

Once you submit a request by clicking **Try it out!** button, you see the following information in the response:

- Curl and HTTP request formats
- Response Body
- Response Code
- Response Headers
Additional Documentation

• ERDAS APOLLO REST API reference guide provides the expected inputs and outputs for the endpoints exposed by the ERDAS APOLLO REST API, as well as examples of properly formatted JSON used in the calls, explanations of the URI parameters, and ranges of expected values.

• The guide is a living document, updated as further endpoints are added and additional clarification is requested or needed.

• Location -- Hexagon Developer Network (HGDN)
  http://www.hexagongeospatial.com/support/hgdn

ERDAS APOLLO REST API REFERENCE

The purpose of this document is to provide reference material relating to the Apollo implementation of the Swagger API. It is not a full Swagger or Apollo tutorial and is not a substitute for the Apollo help files and documentation. This document does summarize and provide templates and examples for Apollo Swagger implementation inputs and outputs. Both the document and the implementation of the API are works in progress.

Reference Contents

Entire Catalog ........................................................................................................................................... 2
Aggregates .................................................................................................................................................. 11
Images ...................................................................................................................................................... 13
Point Clouds .............................................................................................................................................. 14
Vectors .................................................................................................................................................... 15
Generic Data ............................................................................................................................................. 17
Photogrammetric Images ....................................................................................................................... 18
Videos ..................................................................................................................................................... 20
Crawling .................................................................................................................................................. 21
Delete .................................................................................................................................................... 26
Jobs Management ................................................................................................................................... 27
Pyramids .................................................................................................................................................. 28
Thumbnails ............................................................................................................................................... 30
Additional Documentation

These outputs are essentially comprised of job information; the individual fields should be self-explanatory for the most part, with Apollo mostly returning the job ID until the state is COMPLETE. The "context" fields are from the search endpoint. More information is available in Apollo's help files.

**Entire Catalog**

**Aggregates**

**Images**

**Point Clouds**

**Vectors**

**Generic Data**

**Videos**

**Crawling**

The current implementation of the Swagger API in Apollo supports three types of crawling, which version is used depends on the required JSON inputs provided.

**POST**

`.../crawl`

The POST request's basic URI is simple, with the POST body containing the necessary parameters to configure the crawl operation.

**Directory or File Crawling Required Inputs:**

```
{ "crawlType": "file",
  "fileExt": "*
```
Additional Documentation

```
"devscopeAccessories": "devscopeAccessories",
"accessoriesIndex": "accessoriesIndex",
"formatList": [
  "[List1", "List2"
],
"accessoriesIndex": () ** see below
```

Explanations of Properties:

- `replicateDirectory` - A Boolean value that determines whether or not the crawl job will replicate the directory of the data the user is crawling.
- `dropOnDirectory` - A string value that
- `targetDirectory` - A string value that
- `formatList` - A string array value that

**CrawlProperties Section Explanation of Properties:

- `formatListAExtension` - The file format will be generated as an extension when this property is set to true; otherwise, the file format will be generated as active area when set to false. `formatListAsText` - True property will generate footprints identical to the spatial extent of the aggregate or dataset. The `activeArea` option will generate footprints that contain only the valid data found within the aggregate or dataset.
- `immediate` - The "immediate" property is part of the `scheduleSettings` group. The property values are either true or false.
- `recurringJob` - The "recurringJob" property is part of the `scheduleSettings` group. This property provides the user with properties to schedule jobs for future processing if not using recurring property. Leave empty brackets, i.e. "recurringJob": []
- `occurrence` - The available occurrence options are "once", (include new times?) "hourly", "daily", "weekly", or "monthly".
- `startDateTime` - The startDateTime is a string with ISO8601 dateTime format.
- `metadataParsers` - If the () are empty(), the default is that no metadata parsers are enabled. List of parsers from metadata.yaml file "ENIGMA", "RussianSatellite", "CrownRuler", ... The "parsers" property contains an ordered list of metadata parsers. The names must be the same as the names from the metadata.yaml file.
- `executionStrategy` - The "executionStrategy" property options are "firstSuccess" or "executeAll".

Basic Crawl Example

This block of json forms the body of a post request over the Swagger API that will trigger a crawl job. The three fields that must be set with the user's configurations are directory (the full path to the file on the machine where is configured to run on), `parentID` (the ID of the parent folder), and `subfolder` (the name of the aggregate that will be created to house the crawled data).

```json
{
  "crawlJobs": {
    "json": {
      "replicateDirectory": true,
      "directory": "INSERT PATH TO FILES HERE",
      "fileExtensions": ["jpg", "png"]
    },
    "crawlProperties": {
      "parentID": "INSERT PARENT ID HERE",
      "subfolder": "INSERT NAME OF SUBFOLDER HERE",
      "scheduleSettings": {
        "immediate": true
      },
      "generateServices": {
        "aggregate": {
          "generatePathway": false,
          "generateThumbnail": false
        },
        "dataset": {
          "generatePathway": false,
          "generateThumbnail": false
        }
      },
      "viewServices": {
        "aggregate": {
          "listEnabled": true,
          "listEnabled": true
        },
        "dataset": {
          "listEnabled": true,
          "listEnabled": true,
          "geocodingEnabled": true,
          "indexEnabled": true,
          "spellEnabled": true
        }
      }
    }
  }
}
```
Additional Documentation

 Dropbox Crawling Required Outputs:

```
{
  "jobId": "<jobId>",
  "crawType": "dropbox",
  "datasetId": [{"datasetId1", "datasetId2"},
  "parentId": <parentId>,
  "statusMessage": <statusMessage>
}
```

Explanation of Output Parameters:

**jobId** - The **jobId** property is the job id created that is associated with the immediate or scheduled job for the input payload.

**datasetId** - The **datasetId** property represents an array of new catalog ids that were created after crawling the dataset specified in the input payload.

**crawType** - The **crawType** property represents the type of crawl job created. For example, files or database.

**parentId** - The **parentId** property represents an array of new folder ids that were created after crawling the dataset specified in the input payload.

**statusMessage** - The **statusMessage** property represents the returned status code of the current job.

**statusMessage** - The **statusMessage** property is a verbose message regarding the job. This could also be an error message.

- Delete
- Jobs Management
- Pyramids
TAKE THE SESSION SURVEY

Drop your completed comment card in a nearby drop box or complete a full session survey on the HxGN LIVE Mobile App.
THANK YOU