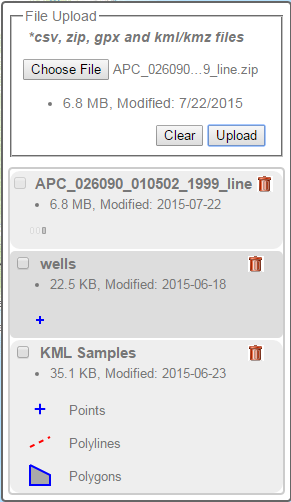
FileUploader Widget

Jared Lang 2015/8/26

# Overview

This document describes the file uploading capabilities implemented by the FileUploader widget and the supported widgets and web services. The FileUploader widget is designed and implement to upload a single file of KML/KMZ, zipped shapefile, csv and gpx, enable users to manage their own upload files, and render the uploaded data in an Esri-compatible styles. It consists of two web services and the HTML5 widget-based web client.



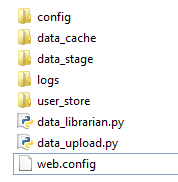
This widget is currently running on <http://myserver/chen_test/fileUploader/index_demo.html>. The physical files are located at C:\inetpub\wwwroot\chen\_test.

# Web Services

Two web services, Data\_Librarian and Data\_Upload, are developed in Python 2.7 and Esri ArcPy 10.2.1. Both are fully tested as standalone web services in IIS 7.5 on Windows Server 2012 R2.

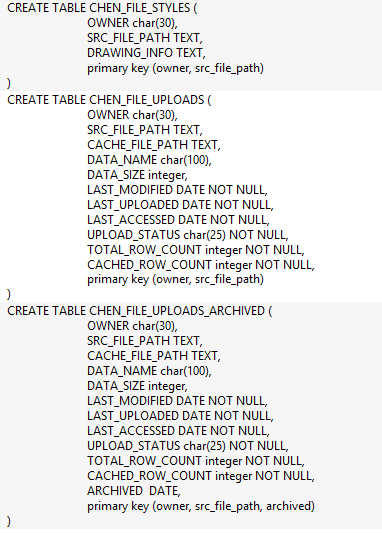
* Data\_Upload.py implements the web service of receiving the file upload via Web.
* Data\_Librarian.py implements the web service of processing data in the uploaded files and managing the uploaded files, and also exposing the management functionalities.

Below is the file structure of the web services.



* The web.config file contains key properties to configure both services.
* The data\_cache folder is to store the generated json files. One subfolder is created for each user and named by its username.
* The data\_stage folder is used as a staging workspace for intermediate or temporary data. One subfolder is created for each user and named by its username.
* The user\_store folder is to save files uploaded by individual users. One subfolder is created for each user and named by its username.
* The logs folder is for the widget logging files.
* The config folder contains a sqlite database file (imaps\_config.db) specifically managed by the FileUploader widget. Inside the database, there are 3 tables to support the widget to operate properly:
  + CHEN\_FILE\_STYLES stores user-defined Esri symbols in json for uploaded files.
  + CHEN\_FILE\_UPLOADS stores the properties of uploaded files.
  + CHEN\_FILE\_UPLOADS\_ARCHIVED stores the properties of uploaded but archived files.

Below are the SQL Table Create statements for sqlite.



# Web Client

The web client is developed with HTML5, DOJO and Esri ArcGIS Server Javascript API. The web client consists of three DOJO widgets: FileUploader, StyleEditor and SvgSymbol, and a sample web page (index\_demo.html) demonstrating how to use and interact with widgets in a common web map application.

* FileUploader is a template-based widget made of a form of file uploading and a list of uploaded file. Its corresponding css style is inside css/FileUploader.css.
* SvgSymbol is a utility widget to transform an Esri symbol to a legend icon.
* StyleEditor is a sample widget to demonstrate the interaction with the FileUploader widget and the style change. Its corresponding css style is inside css/StyleEditor.css.

In addition, FileUploader supports the following user operations in the uploaded list:

* Checkbox allows showing or hiding data on the map instance.
* Single-click the data name when data is on display to make the map zoom to the data extent.
* Double-click the data name to allow changing its name
* Click the trash bin icon to delete (aka., archive) the data
* Double-click the graphic icons under the data name to allow changing its style

Also In support of interaction it exposes the following events:

* load: fired when the widget is loaded
* dataReady: fired when the data is processed and ready to use
* dataOnMap: fired when the data is displayed on the given map instance
* dataArchive: fired when the data is archived successfully
* dataRename: fired when the data is renamed successfully
* dataStyle: fired when the data style is updated successfully. This will automatically trigger the change of its corresponding SVG icon in the uploaded list
* style-start: fired when a SVG icon is doubled clicked.
* clear: fired when the clean button is clicked, which removes all uploaded data from the map instance.

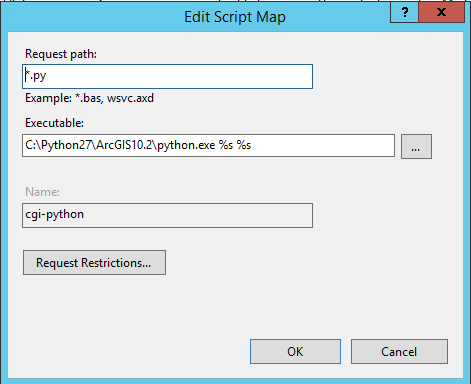
# Deployment Instruction

This instruction is tested successfully on IIS 7.5 on Windows Server 2012 R2.

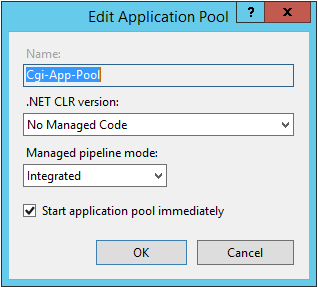
## Web Services

### Configure IIS for FileUploader

1. Copy the services folder into the IIS web folder.
2. Open the IIS Manager and convert the service folder into an application
3. Open Handler Mappings and add Python to handle a py file



1. Add Application Pool



### Configure FileUploader Web Services

1. Open web.config for editing
2. Specify the full paths of the working folders:
   * <add key="store" value="<user\_store\_folder>" />
   * <add key="stage" value="<data\_stage\_folder>" />
   * <add key="cache" value="<data\_cache\_folder>" />
3. Grant IIS with the full-control permission on those working folders
4. Specify the full path of the sqlite database file:

* <add key="db\_conn" value="<some\_folder\imaps\_config.db" />

1. Grant IIS with the read and write permissions on the database file
2. Specifying the full paths of the default style files in web.config.

* <add key="default\_style\_point" value="<deploy\_folder\config\default\_symbol\_point.json" />
* <add key="default\_style\_line" value="<deploy\_folder\config\default\_symbol\_line.json" />
* <add key="default\_style\_polygon" value="<deploy\_folder\config\default\_symbol\_polygon.json" />

1. Grant IIS with the read-only permission on those style files
2. Open data\_upload.py for editing
3. Specify the DEPLOY\_ROOT folder: e.g., "C:\inetpub\wwwroot\chen\_test\services"
4. Open data\_librarian.py for editing
5. Specify the DEPLOY\_ROOT folder: e.g., "C:\inetpub\wwwroot\chen\_test\services"
6. Based on the server memory size, assign an appropriate number to limit the number of features returned by the Data\_Librarian web service. The default value is 10,000.

* <add key="max\_num\_of\_rows" value="10000" />

### Web Service Test

1. Here are some URL to test the deployed web services
   * http://<Web-Deploy-path>/data\_librarian.py?username=<username>&action=list

The expected response is

[]

* + http://<Web-Deploy\_path>/data\_librarian.py?username=<username>&action=data&filename=xxxx

The expected response is

{"error": "no such data file [xxxx]", "scope":"env"}

## Web Client

The index\_demo.html web page provides a working example of utilizing the FileUploader widget.

Below are the required parameters:

var fileUploader = new FileUploader({

map: <esri map object>,

username: <username>,

renderingStyles: {

"point": <default\_point\_symbol>,

"line": <default\_line\_symbol>,

"polygon": <default\_polygon\_symbol>

},

uploadServiceUrl: <data\_upload service url>,

dataServiceUrl: <data\_librarian service url>

}, "<html tag for widget>");

The widget also defines some optional parameters that could be changed at the creation time.

uploadTimeout: 60000 /\* 1 minutes \*/

dataTimeout: 180000 /\* 3 minutes \*/

maxRefreshCount: 5 /\* number of attempts to check the data status \*/

refreshInterval: 30 /\* interval of data status checks in seconds \*/

If the style needs to be changed, the css files are located under the CSS folder. All images should be of 16x16 pixels and are located under the image folder.