
Download Web Service Users Guide

USGS / EROS

February 26, 2010

Revision Sheet

Release No.	Date	Revision Description
	06/08/2009	Original
Rev. 1	11/04/2009	Elaborated on the getData method call.
Rev. 2	02/26/2010	1. Added Python and Java code examples. 2. New formatting for this document.

Download Web Service Users Guide

TABLE OF CONTENTS

1 GENERAL INFORMATION.....	4
1.1 Introduction.....	4
1.2 Organization of the Manual.....	4
1.3 Acronyms and Abbreviations.....	4
2 DOWNLOAD WEB SERVICE.....	5
2.1 Overview.....	5
2.2 initiateDownload() Method.....	5
2.3 getDownloadStatus() Method.....	6
2.4 getData() Method.....	6
2.5 setDownloadComplete() Method.....	7
3 CODE EXAMPLES FOR CALLING THE DOWNLOAD WEB SERVICE.....	8
3.1 Sample Python Code.....	8
3.2 Sample Java Code.....	10

1 GENERAL INFORMATION

1.1 Introduction

The USGS Seamless and Tiled Server functionality has recently been expanded to provide a set of webservices to the developer that can be incorporated into custom applications. This document discusses how to use the Inventory web service to obtain dataset information the USGS Seamless and Tiled Server data holdings. There are two additional web services that can be used to obtain the full resolution data.

Inventory Service. This is a data discovery service. This service will provide information about what data is available over a particular area of interest. The addition of new datasets and demotion of older datasets from on-line systems to near-line systems is an ongoing process. Updates to the Inventory Service occur on a monthly basis. Efforts are currently underway to include in the Inventory Service all of the datasets currently available through the USGS Seamless Server.

Request Validation Service. Utilizing dataset information obtained from the previous call to the Inventory Service and a user-defined area of interest, this service verifies and validates the information, and then returns to the user fully parameterized URL(s) that can be used by the Download Service.

Download Service. This service initiates a request for data, queries the system to obtain a job status, and returns the requested data to the user.

Orthoimagery data that is no longer considered the “best available” is removed from on-line systems and map services on a periodic basis. When this occurs, the dataset is clipped into pre-packaged zipfiles and stored on near-line systems so that it can still be obtained by the public. These datasets currently show up in the Inventory Service with a STATUS = Tiled.

1.2 Organization of the Manual

Section 1 provides a general description of the system.
Section 2 provides a description of the Request Validation Web Service.

1.3 Acronyms and Abbreviations

Acronym	Definition
WMS	Web Map Service
SDDS	Seamless Data Distribution Server
TDDS	Tiled Data Distribution Server
WSDL	Web Service Description Language

2 DOWNLOAD WEB SERVICE

2.1 Overview

The Download Web Service is used to make a request for full resolution data from either of two USGS data distribution systems: The Seamless Data Distribution System, which performs real-time clipping from national data layers, and The Tiled Data Distribution System, which retrieves pre-packaged data tiles.

The URL to the Download Service WSDL page is:

<http://extract.cr.usgs.gov/axis2/services/DownloadService?wsdl>

There are four methods that must be called in the correct sequence:

- initiateDownload() – used to submit a request
- getDownloadStatus() – used to check the current status of the request
- getData() – used to obtain the finished download bundle from USGS webservers
- setDownloadComplete() – used to force immediate cleanup of old requests

2.2 initiateDownload() Method

As you can see in the wsdl file, there are over 40 potential parameters that can be passed to this method. This is why it's best to use the Request Validation Service to compose this URL for you. The URL obtained from the Request Validation Service is guaranteed to correctly pass the initiateDownload requirements. This method call will submit the request to the job queue.

The response returned from initiateDownload method call will include a unique id that will be used as input for all successive calls to the Download Service. If there is an error when using this method, a string containing the word "error" will be returned. The response will look something like this if the call was successful:

```
<ns:initiateDownloadResponse>  
<ns:return>20090608.150009516.152061160045</ns:return>  
</ns:initiateDownloadResponse>
```

Your download request is assigned a priority value based on the number of requests you currently have in the job queue. If you only have one request in the queue, it will always be assigned the highest or "best" priority. All jobs of the same priority are worked on via a First In, First Out basis. If you already have one unfinished job in the queue, then your second job is assigned a priority of one less. Your third unfinished job is assigned a priority of two less than the "best" priority. This is so that one single user cannot use all of the resources when others are also contending for those resources. Therefore, you might want to write your application so that only a certain number of your jobs are in the queue at any time or you might want to monitor the time it takes to work off your jobs and submit new jobs accordingly.

2.3 getDownloadStatus() Method

This method is used to obtain the current status of your request. The USGS data is stored on a variety of systems in a variety of different formats. It will take a short amount of time to find your requested data, reformat the data if necessary, package it up with metadata, and move that finished download bundle to a USGS webserver where it is then available to you for one hour. After one hour, that download bundle is automatically deleted from the web server. You must use the job identifier that was returned from initiateDownload method call.

<http://extract.cr.usgs.gov/axis2/services/DownloadService/getDownloadStatus?downloadID=20090608.150009516.152061160045>

We recommend calling this method no more often than every 30 seconds so that unnecessary status queries don't slow down the servers. As the request passes through different stages of processing, the status code and text message that is returned will indicate the current progress if you want to pass this on to the requestor.

`<ns:return>210,Extracting Data</ns:return>`

If an error has occurred during one of the processing stages, you will receive a string containing the text "error".

This method is simply checking the status field from the job record that was submitted in the initiateDownload method call. One hour after the job record was first submitted, it is automatically deleted from the database, even if your job has not yet finished processing. Therefore, you may want to stagger your requests and/or submit your jobs during off-hours. If your job has returned an error, try it again. Repeated errors to the same product could indicate a problem with the dataset at EROS and USGS Customer Services should be notified so that someone at EROS can investigate the issue. As demand for the services increase and resource contention becomes an issue, our architecture is designed such that we can easily and quickly add or reapportion hardware (if available) to process more requests.

2.4 getData() Method

Once you receive back a status of 400 or greater, the finished download bundle has been placed on a USGS web server and is ready to be downloaded by the user. The download bundle can be retrieved using the getData method call and the job identifier:

<http://extract.cr.usgs.gov/axis2/services/DownloadService/getData?downloadID=20090608.150009516.152061160045>

Note that the finished download bundle will be available on the web server for one hour after the download bundle has been placed there. At that time, an automated process will delete the download bundle to make room for other users. If your application misses the one hour window, then the request must be re-submitted starting at the initiateDownload step.

Depending on what programming language you are using, your application will receive the data as a series of bytes through some kind of connection object. If you have asked for a zip

file in your request, you will write the bytes to a file called xxxxxxxx.zip. If you have requested a tar-gzip file as the bundle format, then you will write the bytes to a file called xxxxxxxx.tgz. The data and all support files are contained within the .zip file or the .tgz file. You are then responsible for extracting the information you want from the download bundle.

2.5 setDownloadComplete() Method

Once the download bundle has finished downloading to your computer, please send a download complete message back to the Download Service so that the job is properly logged and the cleanup process on our web servers can begin. You can do this by using the setDownloadComplete method call and the job identifier:

<http://extract.cr.usgs.gov/axis2/services/DownloadService/setDownloadComplete?downloadID=20090608.150009516.152061160045>

A successful response returned will look like:

<ns:return>Successfully updated 20090608.150009516.152061160045 to Complete</ns:return>

This method is simply a courtesy call to allow our system to begin immediate cleanup of this job. If you fail to call this method, your request will be automatically cleaned up one hour after the initiateDownload method call for the request.

3 CODE EXAMPLES FOR CALLING THE DOWNLOAD WEB SERVICE

3.1 Sample Python Code

```
# submit a job to the Download Service
chunkUrl = "http://extract.cr.usgs.gov/axis2/services/DownloadService/initiateDownload?
PL=HOIT&MSU=http://ims.cr.usgs.gov/servlet/com.esri.esrimap.Esrimap&MSS=USGS_EDC_Ortho_Urban
&MSL=Historical_Orthoimagery_Tiles&MSEA=WEBMAP.ortho_historical_tiles.OBJECTID&DLS=http://edclxs
77.cr.usgs.gov/ortho&FID=ZI&ARC=ZI&DLA=WEBMAP.ortho_historical_tiles.FILENAME&EIDL=13428&siz
=57&ft=-105.023371493578&bot=39.8129764994474&rgt=-
105.005841493561&top=39.82649349946&ORIG=null"

try:
    page = urllib2.urlopen(chunkUrl)
except IOError, e:
    if hasattr(e, 'reason'):
        print 'We failed to reach a server.'
        print 'Reason: ', e.reason
    elif hasattr(e, 'code'):
        print 'The server couldn\'t fulfill the request.'
        print 'Error code: ', e.code
    else:
        result = page.read()
        print result
        # parse response for request id
        if result.find("VALID>>false") > -1:
            # problem with initiateDownload request string
            # handle that here
        else: # downloadRequest successfully entered into queue
            startPos = result.find("<ns:return>") + 11
            endPos = result.find("</ns:return>")
            requestID = result[startPos:endPos]
            print requestID

# call Download service with request id to get status
downloadStatusUrl =
http://extract.cr.usgs.gov/axis2/services/DownloadService/getStatus?
downloadID=20090608.150009516.152061160045"
try:
    page2 = urllib2.urlopen(downloadStatusUrl)
except IOError, e:
    if hasattr(e, 'reason'):
        print 'We failed to reach a server.'
        print 'Reason: ', e.reason
    elif hasattr(e, 'code'):
        print 'The server couldn\'t fulfill the request.'
        print 'Error code: ', e.code
    else:
        result = page2.read()

        # remove carriage returns
```

```

    result = result.replace("&#xd;\n", " ")

    # parse out status code and status text
    startPos = result.find("<ns:return>") + 11
    endPos = result.find("</ns:return>")
    status = result[startPos:endPos]
    print status

# once a status of 400 has been received this code will retrieve
the file and store it locally
    getFileUrl =
"http://extract.cr.usgs.gov/axis2/services/DownloadService/getData?
downloadID=20090608.150009516.152061160045"
try:
    page = urllib2.urlopen(getFileUrl)
except IOError, e:
    if hasattr(e, 'reason'):
        print 'We failed to reach a server.'
        print 'Reason: ', e.reason
    elif hasattr(e, 'code'):
        print 'The server couldn\'t fulfill the request.'
        print 'Error code: ', e.code
    else:
        # store file in a local folder
        downloadFile =
open('D:/localdownloads/20090608.150009516.152061160045.zip', 'wb')
        while True:
            data = page.read(8192)
            if data == "":
                break
            downloadFile.write(data)
            downloadFile.close()
        finally:
            page.close()

# send complete message back to server so it can cleanup the job
setStatusUrl =
"http://extract.cr.usgs.gov/axis2/services/DownloadService/setDownloadC
omplete?downloadID=20090608.150009516.152061160045"

try:
    page = urllib2.urlopen(setStatusUrl)
except IOError, e:
    if hasattr(e, 'reason'):
        print 'We failed to reach a server.'
        print 'Reason: ', e.reason
    elif hasattr(e, 'code'):
        print 'The server couldn\'t fulfill the request.'
        print 'Error code: ', e.code
    else:
        result = page.read()

        # remove carriage returns
        result = result.replace("&#xd;\n", " ")

```

```

    # parse out status code and status text
    startPos = result.find("<ns:return>") + 11
    endPos = result.find("</ns:return>")
    status = result[startPos:endPos]
    print status

finally:
    page.close()

```

3.2 Sample Java Code

```

# use this sample block to call initiateDownload, getStatus or
setDownloadComplete

import java.net.*;

String completeURL =
http://extract.cr.usgs.gov/axis2/services/DownloadService/initiateDownload?
PL=HROT&MSU=http://imsref.cr.usgs.gov/servlet/com.esri.esrimap.Esrimap&MSS=USGS_ED
C_Seamless_Inventory&MSL=TDDS_INDEX&MSEA=FILE_ID&DLS=http://gisdata.usgs.net/
TDDS/DownloadFile.php?TYPE=ortho
%26FNAME=&FID=ZI&ARC=ZI&DLA=FILE_ID&EIDL=14RNP895800_200606_0x5000m_
CL&siz=430&lft=-98.1054426942589&bot=26.0361112377698&rgt=-
98.0903471056137&top=26.0497478879939&ORIG=null";

try {
    URL extURL = new URL(completeURL);
    URLConnection connection = extURL.openConnection();
    connection.setDoOutput(false);
    connection.setDoInput(true);
    connection.setConnectTimeout(10000);
    connection.setReadTimeout(10000);
    connection.connect();

    // Save the XML response as a string.
    InputStream in = connection.getInputStream();
    BufferedReader bin = new BufferedReader(new InputStreamReader(in));
    String ln;
    String tempString="";
    while ((ln = bin.readLine()) != null) {
        if (tempString == null) {
            tempString = ln;
        } else {
            tempString = tempString + ln;
        }
    }
    bin.close();

    // Parse response here
    System.out.println(tempString);

}
catch (SocketTimeoutException ste) {

```

```
        System.err.println(ste.getMessage());
    }
    catch (MalformedURLException mue) {
        System.err.println(mue.getMessage());
    }
    catch (IOException ioe) {
        System.err.println(ioe.getMessage());
    }
}
```

// use this sample block to retrieve the requested file and store it locally.

```
import javax.activation.DataHandler;
import java.net.*;
```

```
String completeURL = "http://extract.cr.usgs.gov/axis2/services/DownloadService/getData?
downloadID=20100222.132733625.152061165003";
```

```
try {
    URL extURL = new URL(completeURL);

    DataHandler dhSource = new DataHandler(extURL);
    InputStream is = dhSource.getInputStream();

    dhSource.writeTo(new
java.io.FileOutputStream("d:/download_folder/20100222.132733625.152061165003.zip));
    is.close();

}
catch (SocketTimeoutException ste) {
    System.err.println(ste.getMessage());
}
catch (MalformedURLException mue) {
    System.err.println(mue.getMessage());
}
catch (IOException ioe) {
    System.err.println(ioe.getMessage());
}
}
```