EPF REST Web Services v1.02.x

Description:

The REST web services described within this document are the second generation web services developed for EPF using the REST web service style with JSON inputs and outputs instead of the current SOAP style xml based web services. Detailed information on REST web services and JSON objects can be found on the Internet and many libraries exist to help in creating and extracting the information from JSON objects in all popular computer programming languages. The function calls are made through a standard https URL style connection with a JSON object contained within a POST (form style) to the corresponding URL for security reasons.

The "logonkey" and "tokenkey" should be SAVED from each reply as they are refreshed on every request/response call made to the web server. These web services must be called one at a time in serial as the security token refresh process will fail if calls are threaded and you do not pass the most recently refreshed security token in the next function call. Only one active session per user login is supported; multiple users cannot use the same login and expect to succeed as only one user will have the most currently refreshed security token. The security token is returned on each call within the JSON object and also in the response header. From a programming perspective, it may be better to program using the headers from the response object, as when the actual file is downloaded, no JSON object is returned. To make the request calls, a JSON object will need to be created and passed as a POST form parameter **obj={jsonObject}** for security reasons using content-type "**application/x-www-form-urlencoded**". The actual call to retrieve the file will be a different than all the other web service calls, as the file is be delivered from an edge server in another location and information must be presented within the request headers to be successful.

The version function call allows users to test service availability through their firewalls using a browser or in code using a simple GET method with no parameters or JSON object and database connectivity is not required for a response to be successful.

NOTE: All items are considered case sensitive.

Functions:

1. https://epfws.usps.gov/ws/resources/epf/**version** **GET**
2. https://epfws.usps.gov/ws/resources/epf/**login** **POST**
3. https://epfws.usps.gov/ws/resources/epf/**logout** **POST**
4. https://epfws.usps.gov/ws/resources/download/**list** **POST**
5. https://epfws.usps.gov/ws/resources/download/**file** **POST**
6. https://epfws.usps.gov/ws/resources/download/**epf** **POST**
7. https://epfws.usps.gov/ws/resources/download/**status** **POST**

NOTE: the form parameter name must be "**obj**", as in **obj=**{json object to string}

The web services look for a parameter called obj and takes the string value and converts it back to a JSON object to retrieve the internal values.

Function Call Details:

Version:

The ***epf/version*** function call is a simple GET style call that can be used as a test call in a web browser. By entering the URL, a response object containing a JSON object should be returned immediately. This can also be used to test connectivity through firewalls and if this simple call does not work it could indicate that the web server is hanging up or slow to respond.

INPUT: Call the URL (GET) without any additional parameters and a response JSON object will be return or a server error if the services are not running.

OUTPUT: JSON object similar to example below.

```
{
    "response":"success",
    "messages":"Web service version and build date.",
    "version":"v1.02.1",
    "build":"2013-05-20"
}
```

Login:

The ***epf/login*** function call is used to log into EPF system and retrieve the required keys for all future web service calls. As with most systems, one should program for sessions timing out and passwords expiring.

INPUT: Call the URL via POST method with JSON object serialized and set parameter.

```
obj={
        "login":"[epf login required]",
        "pword":"[epf password required]"
}
```

OUTPUT: JSON object similar to example below.

```
{
    "response":"[success] or [failed]",
    "messages":"Login validation succeeded.",
    "logonkey":"[logon key]",
    "tokenkey":"[security token]"
}
```

Logout:

The ***epf/logout*** function call is used to log out of EPF system and remove all security tokens from being active. This is not a required function call but is considered good practice and will deactivate your security token.

INPUT: Call the URL via POST method with JSON object serialized and set parameter.

```
obj={
        "logonkey":"[epf logon key required]",
        "tokenkey":"[epf security token required]"
}
```

OUTPUT: JSON object similar to example below.

```
{
    "response":"[success] or [failed]",
    "messages":" Logout process succeeded."
}
```

List:

The *download/list* function call is used to retrieve a list of actively available files for the supplied user login, product code and product id. Additional parameters can be optionally supplied to filter the list even more.

NOTE: Added the ability to filter by more than one status code … you can now pass a string of status codes. "SNX" would bring back all 3 statuses.

INPUT: Call the URL via POST method with JSON object serialized and set parameter.

```
obj={
        "logonkey":"[epf logon key required]",
        "tokenkey":"[epf security token required]",
        "productcode":"[epf product code required]",
        "productid":"[epf product id required]",
        "status":"[filter by download status - optional]",
        "fulfilled":"[filter by fulfilled date - optional]",
}
```

OUTPUT: JSON object similar to example below.

```
{
        "response":"[success] or [failed]",
        "messages":"Process succeeded.",
        "logonkey":"[logon key]",
        "tokenkey":"[security token]",
        "reccount":"1",
        "fileList":[
                {
                    "fileid":"12345",
                    "status":"N",
                    "filepath":"/epfdata/ncam/20120815/rdi.tar",
                    "fulfilled":"2012-08-15"
                },
                {
                    "fileid":"23456",
                    "status":"N",
                    "filepath":"/epfdata/ncam/20120915/rdi.tar",
                    "fulfilled":"2012-09-15"
                }
        ]
}
```

Status:

The download/status function call is used to change the file status; either before being downloaded (set to "S" for started) and / or after a successful download (set to "C" for completed).

INPUT: Call the URL via POST method with JSON object serialized and set parameter.

```
obj={
        "logonkey":"[epf logon key required]",
        "tokenkey":"[epf security token required]",
        "newstatus":"[required - [N] or [S] or [X] or [C] or NCSX for ALL",
        "fileid":"123456"
}
```

OUTPUT: JSON object similar to example below.

```
{
        "response":"[success] or [failed]",
        "messages":"Update process succeeded.",
        "logonkey":"[logon key]",
        "tokenkey":"[security token]"
}
```

File:

The download/file is used to get the actual file from the AKAMAI edge server. This call is more complicated than the other calls as it requires information to be placed in the request headers so that AKAMAI can identify the file requiring download. Akamai only allows the file download after the information sent has been validated is our database.

INPUT: Call the URL via POST method with JSON object serialized and set parameter.

```
obj={
        "logonkey":"[epf logon key required]",
        "tokenkey":"[epf security token required]",
        "fileid":"[fileid from file list required]"
}
```

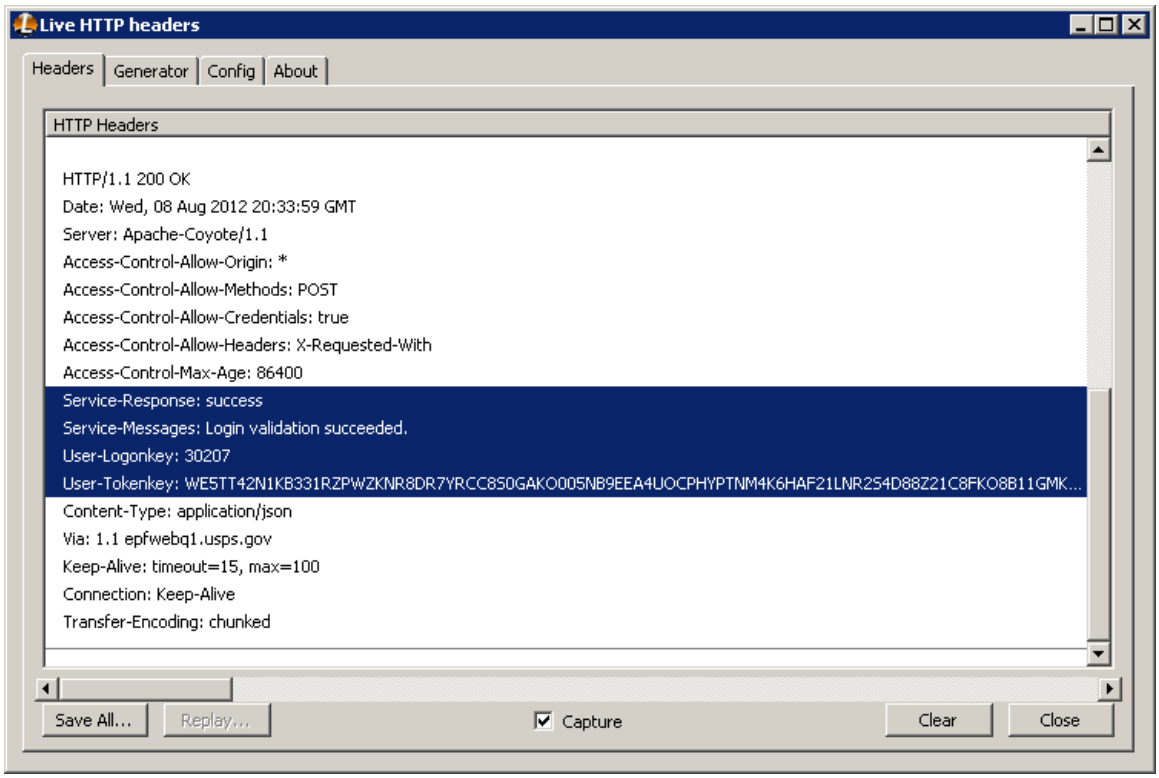
Four headers are required to be added to the REQUEST object.

(All of these required items are returned in the LIST response)

```
request.Headers.Add("Akamai-File-Request", "/epfdata/ncam/rdi/20120715/rdi.tar");
request.Headers.Add("logonkey", [epf logon key required]);
request.Headers.Add("tokenkey", [epf token key required]);
request.Headers.Add("fileid", "589765");
```

OUTPUT: File is returned as the response object. Logonkey and tokenkey have to be extracted from the headers to be used in the next web service call.

```
Header User-Logonkey = [logonkey]
Header User-Tokenkey = [tokenkey]
```



epf:

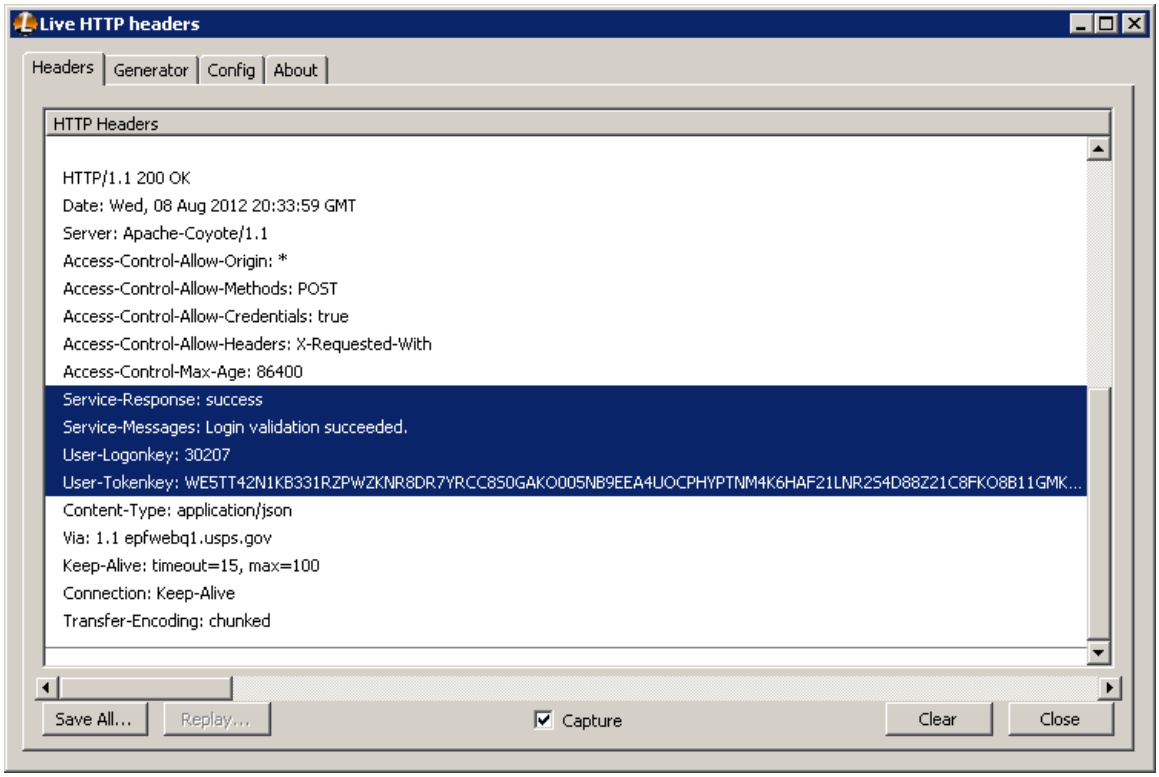
The download/epf function call is used to download smaller EPF files from the web servers instead of the AKAMAI edge servers (used only for large files). This call and response is similar to the call to download the large files, but does not require the header information as AKAMAI will pass this request through to EPF. For programming reasons, if you include the header information it will just be ignored as AKAMAI response to the URL pattern /download/file and not download/epf.

INPUT: Call the URL via POST method with JSON object serialized and set parameter.

```
obj={
        "logonkey":"[epf logon key required]",
        "tokenkey":"[epf security token required]",
        "fileid":"[fileid from file list required]"
}
```

OUTPUT: File is returned as the response object. Logonkey and tokenkey have to be extracted from the headers to be used in the next web service call.

```
Header User-Logonkey = [logonkey]
Header User-Tokenkey = [tokenkey]
```



JSON Object Field Descriptions (may be case sensitive):

- **logonkey** – user login key returned from the login call. This key will not change, but must be used in every call for security validation.
- **tokenkey** – security token returned from the login call. This key must be used in every function call with a newly refreshed security token being return to use in your next call. Threading should not be used, as the security token could get out of synch, only serial calls are supported.
- **productcode** – This is the product code assigned to products. You will need to know your product codes and only one product code can be requested per list call.
- **productid** – The is a sub-key of the product code. You will need to know your product ID and only one product code + product ID can be requested per list call. This basically acts as a filter to limit the size of the list being returned.
- **status** – (optional) The current download status of the listed file. This can be used as an additional filter with most requesters being interested in only newly released files. After the file is downloaded, status should be set to completed.
- **fulfilled** – (optional) The fulfillment date can be used as an additional filter. Date format is 'CCYY-MM-DD' and will be ignored if passed in an incorrect format.

File Statuses: (leave out for ALL to be returned)

- "N" = new file available
- "S" = download started
- "X" = download canceled
- "C" = download completed successfully

Product Codes for AKAMAI Edge Server Files - /download/file

NCAW = weekly files
NCAM = monthly files

Product CODE	Product ID	Description
AISVR	AISVIEWER	AIS Viewer (Monthly Build)
AMS	AMS	AMS Data
AMS	DPV	DPV® Data
AMS	DSF	DSF2® Data
AMS	ELOT	eLOT® Data
AMS	IBIP	IBIP Data
AMS	LLK	LACSLink® Data
AMS	SLK	SUITELink® Data
AMS	AMSDK	AMS Developer Kit
AMS	DPVDK	DPV® Developer Kit
AMS	DVD_COMM	AMS Commerical DVD
AMS	DVD_POST	AMS Postal DVD
AMS	DVD_IBIP	AMS IBIP DVD
[weekly]		
NCAW	NCL18H	NCOALink® 18 MONTH HASH
NCAW	NCL18F	NCOALink® 18 MONTH FLAT
NCAW	NCL48H	NCOALink® 48 MONTH HASH
NCAW	NCL48F	NCOALink® 48 MONTH FLAT
NCAW	NCL18ANKH	NCOALink® WITH ANKLink® HASH
NCAW	NCL18ANKF	NCOALink® WITH ANKLink® FLAT
[monthly]		
NCAWM	NCL18H	NCOALink® 18 MONTH HASH
NCAWM	NCL18F	NCOALink® 18 MONTH FLAT
NCAWM	NCL48H	NCOALink® 48 MONTH HASH
NCAWM	NCL48F	NCOALink® 48 MONTH FLAT
NCAWM	NCL18ANKH	NCOALink® WITH ANKLink® HASH
NCAWM	NCL18ANKF	NCOALink® WITH ANKLink® FLAT

[monthly]

NCAM	DPVFL	DPV® FULL
NCAM	DPVFLT	DPV® FLAT
NCAM	DPVSP	DPV® SPLIT
NCAM	DSFFL	DSF FULL
NCAM	DSFFLT	DSF FLAT
NCAM	DSFSP	DSF SPLIT
NCAM	LACLNK	LACSLink®
NCAM	RDI	RDI™
NCAM	STELNK	SUITELink®

Product Codes for EPF Web Server Files - /download/epf

Product CODE	Product ID	Description
ACS	INVOICE	ACS Invoices
ACS	PARENT	ACS Products
ACS	RECONCILE	ACS Reconciliation Report
AEC	AECDATA	AEC Data
AEC	AECDOCS	AEC Documents
AEC	STATRPT	AEC STAT RPT File
AEC	BLANKZIP	AEC ZIP File
AEC	AECIIDATA	AEC II Data
AEC	AECIIDOCS	AEC II Documents
AEC	AECIIZIP	AECII ZIP File
AEC	SUMMARYRPT	AECII Summary RPT File
AEC	SUMMARYTXT	AECII Summary Text File
AEC	STMNTRPT	AECII Statement RPT File
AEC	STMNTTXT	AECII Statement Text File
AEC	FORM3553RPT	AECII Form 3353 RPT File
AEC	FORM3553TXT	AECII Form 3353 Text File
AIS	CR215N	CARRIER ROUTE NATIONAL
AIS	CR215S	CARRIER ROUTE BY STATE
AIS	CS215N	CITY STATE NATIONAL
AIS	DS215N	DELIVERY STATISTICS
AIS	FD215N	FIVE-DIGIT ZIP
AIS	LT215N	eLOT® NATIONAL
AIS	LT215S	eLOT® BY STATE
AIS	ZC215N	Z4 CHANGE
AIS	ZM215N	ZIPMOVE
AIS	ZP215N	ZIP+4® NATIONAL
AIS	ZP215S	ZIP+4® BY STATE
AISW	LT215N	eLOT® NATIONAL
CASS	CASSERROR	CASS Error Report
CASS	CASSFILE	CASS Stage File
CASS2	CASS2ALPHA	Trailing Alpha File
CASS2	CASS2INFO	Z4 Info File
CASS2	CASS2WARN	Early Warning File
CDSW	WEEKLY	CDS Weekly Product
CHART	ZONECHARTS	Zone Charts Matrix
COA	CENSUS_AIS_MTH	Census Fulfillment (Monthly)
COA	CENSUS_AIS_SEMI	Census Fulfillment (Semi-Annual)
COA	CENSUS_TRANSACTIONS	Census Transactions
HUD	NOSTAT	HUD NOSTAT Counts
HUD	VACANT	HUD Vacant Counts
LLIST	LABELINGLISTS	Labeling Lists
NCAD	MAINFRAME	NCOALink® Daily Delete [EBCDIC]
NCAD	TEXTFILE	NCOALink® Daily Delete [TEXT]
PAVE	PAVEFILE	PAVE and MAC Batch Product