Webserviceshell 2.0 User Documentation

8 Feb 2016

Webserviceshell 2.0 User Documentation
Introduction
- WSS core features:

WSS Concepts
Conventions and Configuration Concepts
WSS Configuration
- Built in endpoints
- Service configuration file (servicename-service.cfg)
- Global Parameters
- Setting up Endpoints
- Endpoint Parameters
- Managing HTTP headers for a client response

Param configuration file (servicename-param.cfg)
- Reserved Parameters
- param.cfg parameters

Log4j properties file (servicename-log4j.properties)
Additional CmdProcessor Features
- Command Line Process Time Limits
- CmdProcessor to Command Line Interface
- Setting HTTP Headers From a Command Line Process
- Environmental Information for Command Line Processes

Changes from version 1.x
- Converting an existing Web Service Shell application to 2.x
- Changes for endpoints provided by Web Service Shell
- Service related changes - global parameters
- Service related changes - endpoint parameters
- Param.cfg configuration changes

Parameter File Examples
- service.cfg
- param.cfg
- log4j.properties

Introduction

The Web Service Shell (WSS) is a web service that can be configured via simple properties files to utilize external resources (either command line programs or Java classes) to fulfill web service requests.

WSS core features:

- manage HTTP connection with the client
- validate request parameters
- log service requests, state, and errors
- manage execution of command line programs or Java classes
- handle, via the servlet container, HTTP authentication

WSS is written in Java and is delivered as a deployable web application in the form of a .war file. When the war file is installed into a Tomcat or other servlet container, WSS reads its respective configuration files and is ready to handle client requests. Java programming is not needed to configure and operate a web service using WSS.

WSS Concepts

One of the primary objectives of WSS is to enable internet access to data that may only be available through command-line processes. WSS takes care of the HTTP communication then executes task specific programs to retrieve the desired data, effectively separating the HTTP component of a web service from the data extraction work. While the WSS can be extended by writing Java code, two general use processors are provided with WSS:

- edu.iris.wss.endpoints.CmdProcessor
- edu.iris.wss.endpoints.ProxyResource

The CmdProcessor executes command line programs and streams requested data back to a client. ProxyResource returns data to a client and only requires a simple URL configuration parameter and the respective output media type parameter.

Both CmdProcessor and ProxyResource can be configured multiple times in one WSS installation to implement multiple endpoints and deliver a suite of data products to HTTP clients.

Conventions and Configuration Concepts

A few conventions are used to relate configuration files to their respective web service and define the URL provided to a client. For installation, the war file name should be changed according to the conventions defined below.

The following discussion is based on a URL of this form:

http://service.iris.edu/fdsnws/event/1/query?minmagnitude=8.5&format=geocsv

where the parts of the URL may be referred to by these terms:

- **protocol** - http
- **host** - service.iris.edu
- **service (or application)** - fdsnws/event/1
- **endpoint** - query
- **client interface parameters** - minmagnitude and format (i.e. the terms after the '?' character on the left side of the '=' characters)

The general concept of administering WSS services is that for each desired service, a single instance of WSS needs to be installed, and for each desired "endpoint" in a service, a respective set of parameters need to be configured.

Note: Some references external to this document define "endpoint" as the full URL up to the '?' character (e.g. http://service.iris.edu/fdsnws/event/1/query). However, in this document, "endpoint" refers only to the portion of the URL between the service (i.e. "fdsnws/event/1") and the client interface parameters (i.e. "minmagnitude=8.5&format=geocsv").

The host and protocol are usually set for a given site and are not discussed further here.

service naming convention -. The service name is derived from the installed WSS war file name. For example, the file **fdsnws#event#1.war** results in the following:

http://service.iris.edu/fdsnws/event/1/

where **fdsnws/event/1**, is determined by replacing any hash characters '#' in the WSS war file name with slash characters '/'.

endpoints - Endpoints are defined in a service.cfg file. An endpoint name may include slashes or dots. For example, endpoints named "query", "v2/swagger" and "application.wadl" result in these URLs, respectively:

http://service.iris.edu/fdsnws/event/1/query
http://service.iris.edu/fdsnws/event/1/v2/swagger
http://service.iris.edu/fdsnws/event/1/application.wadl

Interface parameter convention - Client interface parameters for a given endpoint are defined in a param.cfg file (discussed in more detail below). For example, to define the parameters "minmagnitude" and "format" for the "query" endpoint, add the following lines to the param.cfg file:

query.minmagnitude=NUMBER
query.format=TEXT

A client can now make a request using this URL:

http://service.iris.edu/fdsnws/event/1/query?minmagnitude=8.5&format=geocsv

In summary -

- A war file name may be chosen as needed, but respective configuration file names must reflect that war file name.
- Endpoint names may be chosen as needed, but the contents of WSS configuration files must reflect that endpoint name.

WSS Configuration

WSS uses configuration files to define endpoints and interface parameters, as well as manage operation of WSS. The following naming convention is used so that multiple web services can be deployed in the same container.

configuration file name convention - WSS tries to open these configuration files when starting. These file names are derived from the WSS .war file name and are determined by replacing any hash character '#' with a dot character '.' and appending the respective file type. For the fdsnws#event#1.war file, the required configuration files are:

- **fdsnws.event.1-service.cfg** - contains operational and endpoint parameters
- **fdsnws.event.1-param.cfg** - contains client interface parameters for each endpoint
- **fdsnws.event.1-log4j.properties** - contains logging configuration

Built in endpoints

These endpoints are built in to WSS and are available to a client without configuration. The wssstatus endpoint is limited to clients on the local network only as the status may contain site sensitive information.

Endpoint	Default Value	Comments
/	Internally generated information page	Use parameter rootServiceDoc in service.cfg to specify a URL with this service's specific documentation.
wssstatus	status information about this WSS and its configuration	Blocked to external request, it is only available on the local network.
wssversion	version of the Web Service Shell code	The wssversion value is set in code when WSS is released.
whoami	remote address of requestor	The address returned may be affected by network topology.

version	"notversioned"	Use parameter version in service.cfg to set a version number or identifier as needed for this service.

Service configuration file (servicename-service.cfg)

The service.cfg file is composed of two types of parameters: global and endpoint parameters. Global parameters apply to the application as a whole while endpoint parameters can be configured multiple times.

Global Parameters

Parameter Name	Default Value	Description
appName	"notnamed"	It can be different from the war file name - it can indicate this service or data application name - Note, this value is also used in output to log files as field "application".
version	"notversioned"	Indicates the version of this web service. The value here is returned to a client from a "version" endpoint request.
corsEnabled	true	when true, the HTTP response header includes Access-Control-Allow-Origin: *
rootServiceDoc	null	a URL providing information about the application, documentation on usage, etc., if a value is not provided, a generic html page will be generated.
loggingMethod	"LOG4J"	choices are LOG4J or JMS.
sigkillDelay	30	time in seconds before a handler process is sent a SIGKILL, see section "Command Line Process Time Limits", time starts after handlerTimeout (see Endpoint Parameter table)
singletonClassName	null	optional - A user provided Java class that will be instantiated once when the service starts. A service can use this class to provide capability that may be needed by individual endpoints in the application.

Setting up Endpoints

The flexibility to set multiple arbitrary endpoints within the service.cfg file is a primary benefit to the 2.x version of the Web Service Shell. An endpoint is defined and configured within the service.cfg file by prepending a string in front of the parameters described in this section.

For example, to specify a "query" endpoint to execute programABC:

query.handlerProgram=/usr/folder1/programABC
query.handlerTimeout=45

plus other endpoint parameters as needed.

Likewise, an endpoint "answer" could execute programDEF and be specified as:

answer.handlerProgram=/usr/folder1/programDEF
answer.handlerTimeout=160

plus other endpoint parameters as needed.

Endpoint Parameters

The term "formatType" refers to the name of a media type provided by an endpoint. It is a case insensitive string defined in the formatTypes parameter. The formatType name is used when constructing the formatDispositions parameter and for the file type on downloaded files.

The term "${UTC}" can be used where noted to insert a UTC time string in ISO 8601 format.

The term "${appName}" can be used to insert the appName parameter string.

Parameter Name	Default Value	Description
endpointClassName	edu.iris.wss.endpoints.CmdProcessor	a Java class that extends the IrisProcessor class. The class will be instantiated for each request on this endpoint. Two IrisProcessor classes are provided with Web Service shell, edu.iris.wss.endpoints.CmdProcessor and edu.iris.wss.endpoints.ProxyResource. ProxyResource provides the capability to deliver the contents specified in the proxyURL parameter.
handlerProgram	"nonespecified"	a fully qualified executable file name. It must be specified when endpointClassName is set to edu.iris.wss.endpoints.CmdProcess
handlerTimeout	30	time in seconds before a handler process is sent a SIGTERM. see section "Command Line Process Time Limits".
handlerWorkingDirectory	/tmp	a valid folder with write access. It is required to start a process and if any command line process needs to write to the file system.
usageLog	true	when true, log information is written to the usageLogger logger defined in the log4j.properties file or JMS, depends on the value of loggingMethod
		a list of (formatType, media type) pairs that are provided by this endpoint.

formatTypes	BINARY: application/octet-stream	When provided, the first pair in the list becomes the new default. Note: User selection of a formatType is accomplished with the "format" interface parameter.
formatDispositions	empty string	a list of (formatType, HTTP Content-Disposition value) pairs that will override the Content-Disposition header set by WSS default or addHeader parameter. ${UTC} and ${appName} may be used in the value field.
addHeaders	empty string	a list of HTTP headers that will add to or override any default headers. ${UTC} and ${appName} may be used in the value field.
postEnabled	false	when false, POST request are ignored, when true, the POST body is passed to the endpoint Java class or handler program.
logMiniseedExtents	false	when true, additional Miniseed channel information is collected and logged, only applies to edu.iris.wss.endpoints.CmdProcess
user404For204	false	when true and a response has no data, return an HTTP code 404 instead of 204.
proxyURL	"noproxyURL"	when used, it must point to a valid URL. It is required when the endpointClassName is set to edu.iris.wss.endpoints.ProxyResource

Managing HTTP headers for a client response

HTTP headers can be set both by endpoint parameter and by a command line handler. WSS sets headers in the following order:

1. WSS default headers
   - When corsEnabled is true, return this header
     access-control-allow-origin: *
   - When formatType is MSEED, MINISEED, or BINARY, return this header
     content-disposition: attachment; filename=service${UTC}.${formatType} (note: no file type for BINARY)
     otherwise, return this header
     content-disposition: inline; filename=service${UTC}.${formatType}
2. **addHeaders** - adds new headers, or overrides default headers.
3. **formatDispositions** - overrides content-disposition header, but only for the respective formatType.
4. **command line handler** - can add to, or override previous headers - see section "Additional CmdProcessor Features".

Param configuration file (servicename-param.cfg)

The contents of the param.cfg file specify the names and types of client interface parameters that a given endpoint supports. Web Service Shell will perform type checking of parameters from client requests before executing respective handlers or Java classes. This enables a quick response to a client for simple parameter format errors in a request URL.

The data types recognized by Web Service Shell are NUMBER, TEXT, BOOLEAN, DATE, and NONE. If the type "NONE" is used for a parameter type, WSS will return a "No value permitted" exception for that parameter.

Reserved Parameters

A few interface parameters have special meaning in Web Service Shell and can only be used as defined here. The reserved interface parameters are:

- format
- aliases
- output

WSS uses the "format" parameter to determine the response media type. The value of format must be one of the formatType names defined in formatTypes parameter. By default, the format parameter does not need to be defined and the response will be the default media type. Add the format parameter to allow a client to explictly select media type.

The "aliases" parameter can be used to specify one or more short names for interface parameters on a given endpoint. The aliases parameter itself is not accessible by a client, see a usage example below.

"output" is a deprecated parameter and should not be used.

param.cfg parameters

Client interface parameter names and types are defined as needed for respective endpoints in the param.cfg file. Each parameter must be prepended with an endpoint name. The endpoint name must be defined in the service.cfg file or the parameter will be ignored.

For example, to specify interface parameters "format", "minlongitude", "maxlongitude" for endpoint "query" enter the following lines in a param.cfg file:

```
query.format=TEXT
query.minlongitude=NUMBER
query.maxlongitude=NUMBER
```

The "aliases" parameter can be used to define shorter names for a respective interface parameter name. Each endpoint may have an "aliases" list that relates a defined parameter to one or more alternate parameter names. For example, to define mnlg and minlong as short names for minlongitude; and mxlong for maxlongitude, add the following:

```
query.aliases = \
```

**minlongitude: (mnlg, minlong), \
maxlongitude: mxlong**

Log4j properties file (servicename-log4j.properties)

WSS uses log4j to log two types of messages: operation messages and data usage messages. Use the log4j properties file to specify the name and location of the respective log files. Note that if parameter loggingMethod is set to JMS, usage messages will go to a UsageStatsTopicDestination binding.

If a respective log4j properties files is not found when WSS starts, it will try to write log messages to files wss.log and wss_usage.log in the container log folder.

Additional CmdProcessor Features

The Java class, edu.iris.wss.endpoints.CmdProcessor, provided with WSS has additional features that may be useful when developing services.

Command Line Process Time Limits

Once a command line process starts, CmdProcessor will wait for handlerTimeout seconds for a process to return data or error messages. Once handlerTimeout is exceeded, CmdProcessor sends the process a SIGTERM. A process may catch the SIGTERM and do cleanup as needed, write messages to standard err, and terminate. If the process does not handle a SIGTERM, the process will be sent a SIGKILL sigkillDelay seconds later.

CmdProcessor to Command Line Interface

CmdProcessor sends request parameters to the command line process's standard input. The interface parameters and values are written to the process in this form:

--interfaceParameter value

If the client request is a POST, a parameter, --STDIN (with no value) along with the POST body is written to the process.

If an error occurs, CmdProcessor checks standard error and includes any information read from standard error in an error response to the client.

Once CmdProcessor detects any data from the process on standard output, it returns an HTTP 200 to the client and hands off the data stream to the web service container, which then streams the data to the client.

Setting HTTP Headers From a Command Line Process

CmdProcessor has the ability to set response HTTP headers. The CmdProcessor checks for header values when a process starts writing and then returns any headers found to WSS. WSS adds or overrides any previously set headers before returning an HTTP response, see section "Managing HTTP headers for a client response".

The following restrictions apply:

- A command line process that needs to set headers must write the header information before writing any other output
- The header information must be text in the following form:
    - starts with "HTTP_HEADERS_START"
    - followed by each header in regular HTTP format, terminated with linefeed or carriage return linefeed
    - ends with "HTTP_HEADERS_END"
- The marker strings must not be followed by linefeed or carriage return linefeed.

For example, to provide headers "Content-Disposition", "Access-Control-Allow-Origin", and "Test-Header", the output looks like this (\n indicates linefeed):

```
HTTP_HEADERS_STARTContent-Disposition : inline\nAccess-Control-Allow-Origin : http://host.example\nTest-Header : value
```

Environmental Information for Command Line Processes

When CmdProcessor starts a command line process, the following environmental values are set.

REQUESTURL - the client request URL
USERAGENT - value from request HTTP header User-Agent
IPADDRESS - client IP, may be affected by network topology
APPNAME - value from appName configuration parameter
VERSION - value from version configuration parameter
CLIENTNAME - dummy string, not currently in use
HOSTNAME - host name of WSS server
If an authenticated user is present, then also AUTHENTICATEDUSERNAME - authenticated user name

Changes from version 1.x

Web Service Shell has been upgraded to:

- enable more flexibility in naming a WSS application

- support more than one user defined endpoint per WSS application
- generalize proxying capability
- enable setting HTTP headers from a command line process
- add detailed error description as needed

Additionally, unused parameters have been removed and a few parameter names have been changed.

Converting an existing Web Service Shell application to 2.x

The general steps needed to change a Web Service Shell 1.x configuration to 2.x are:

1. service.cfg file
   - change web service part of name as needed
   - create global parameters
   - create endpoint for "query"
   - create endpoints for "application.wadl" or "v2/swagger" as needed and set respective proxyURL
2. param.cfg file
   - change web service part of name as needed
   - change parameter names by prepending endpoint name
3. log4j.properties
   - change web service part of name as needed
   - change output log file names as needed

Changes for endpoints provided by Web Service Shell

Previous Parameter	New Parameter	Comment
/	same as before	
status	wssstatus	endpoint rename
application.wadl	builtin implementation removed	A general solution is now provided. Add an endpoint configuration with endpointClassName set to edu.iris.wss.endpoints.ProxyResource, add proxyURL and set the proxy content, and add formatTypes and set the default media type.
wssversion	same as before	
whoami	same as before	
version	same as before	
catalogs	removed	Removed from WSS code and will be re-implemented as endpoint in Event service
contributors	removed	same as catalogs
counts	removed	same as catalogs

Service related changes - global parameters

Previous Parameter	New Parameter	Comment
appName	same as before	
version	same as before	
corsEnabled	same as before	
rootServiceDoc	same as before	
loggingMethod	same as before	
sigkillDelay	same as before	
singletonClassName	same as before	
rootServicePath	removed	not needed
jndiUrl	removed	not needed
connectionFactory	removed	not needed
topicDestination	removed	not needed
argPreProcessorClassName	removed	not needed

Service related changes - endpoint parameters

Previous Parameter	New Parameter	Comment
streamingOutputClassName	endpointClassName	parameter renamed

handlerProgram	same as before	
handlerTimeout	same as before	
handlerWorkingDirectory	same as before	
usageLog	same as before	
outputTypes	formatTypes	parameter renamed
-	formatDispositions	new parameter
-	addHeaders	new parameter
postEnabled	same as before	
-	logMiniseedExtents	new parameter - adds ability to **not do** extra processing and logging for Miniseed channel information
use404For204	same as before	
-	proxyURL	new parameter, used when endpointClassName is set to edu.iris.wss.endpoints.ProxyResource

Param.cfg configuration changes

Configuring interface parameters has not changed except to add an endpoint name. Each parameter must now be prepended with an endpoint name. The endpoint name must be defined in the service.cfg file or the parameter will be ignored.

For example, to change parameters for endpoint "query":

original form	is now
format=TEXT	query.format=TEXT
minlongitude=NUMBER	query.minlongitude=NUMBER
maxlongitude=NUMBER	query.maxlongitude=NUMBER

The "aliases" parameter is handled like other parameters, i.e. prepend the endpoint name in front of "aliases".

```
answer.aliases = \
minlongitude: (mnlg, minlong), \
maxlongitude: mxlong
```

Additional endpoints are specified in the same way, for example, to add parameters for an endpoint "answer":

```
answer.format=TEXT
answer.minlongitude=NUMBER
answer.maxlongitude=NUMBER
```

Parameter File Examples

service.cfg

```
# Service Documentation and context path baselines
# ---------------- globals
appName=my-service-app-name
version=1.0.0

# CORS is enabled by default, set to false to disable CORS processing
##corsEnabled=false

# a URL providing information about the application, documentation on usage, etc.
rootServiceDoc=http://service/fdsnws/dataselect/docs/1/root/

# LOG4J or JMS
loggingMethod=LOG4J

# time is seconds
sigkillDelay=30

# -------- endpoint --------
query.endpointClassName=edu.iris.wss.endpoints.CmdProcessor
query.handlerProgram=/user/home/tomcat/bin/extractdata-handler

# time in seconds for command line processes
query.handlerTimeout=300

query.handlerWorkingDirectory=/tmp

# usageLog is true by default, set this to false to disable usage logging
#query.usageLog=false
```

```
# formatTypes - specifies a list of "formatType: mediaType" pairs
query.formatTypes = \
    mseed:application/vnd.fdsn.mseed, \
    miniseed:application/vnd.fdsn.mseed, \
    text: text/plain,\
    csv: text/csv,\
    json: application/json, \
    xml: application/xml

# Disable or remove this to disable POST processing
query.postEnabled=true

# false by default, enables additional miniseed processing and logging
##query.logMiniseedExtents=true

# Enable this to return HTTP 404 in lieu of 204, NO CONTENT
query.use404For204=true

# -------- endpoint --------

application.wadl.endpointClassName=edu.iris.wss.endpoints.ProxyResource

# required when endpointClassName is set to edu.iris.wss.endpoints.ProxyResource
application.wadl.proxyURL=http://service/fdsnws/dataselect/docs/1/wadl
application.wadl.formatTypes = \
    xml: application/xml
```

param.cfg

```
query.format=TEXT
query.type=TEXT

query.minlongitude=NUMBER
query.maxlongitude=NUMBER
query.minlatitude=NUMBER
query.maxlatitude=NUMBER
```

log4j.properties

```
log4j.rootLogger=INFO, ShellAppender

log4j.appender.ShellAppender=org.apache.log4j.RollingFileAppender
log4j.appender.ShellAppender.File=${catalina.home}/logs/my_app_1.log
log4j.appender.ShellAppender.MaxFileSize=10MB
log4j.appender.ShellAppender.MaxBackupIndex=5
log4j.appender.ShellAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.ShellAppender.layout.ConversionPattern=%d %-5p [%t]: %c{1} %x - %m%n

log4j.category.UsageLogger=INFO, UsageAppender
log4j.additivity.UsageLogger=false

log4j.appender.UsageAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appender.UsageAppender.File=${catalina.home}/logs/my_app_usage_1.log
log4j.appender.UsageAppender.DatePattern='_'yyyy-MM-dd
log4j.appender.UsageAppender.layout=edu.iris.wss.utils.WssLog4JLayout
log4j.appender.UsageAppender.layout.ConversionPattern=%m%n
```