WSS 2.4 Documentation (in progress)

Webserviceshell 2.4 User Documentation

17 May 2017

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
- route one or more endpoint queries to execution of respective command-line programs or Java classes
- handle, via the servlet container, HTTP authentication

WSS 2.x change overview, see the Issues page for more details:

- Version 2.x
  - enable queries to multiple endpoints
  - add more flexibility for service naming convention
  - add selected new configuration options, standardize naming convention
  - generalize the ability to proxy additional content
  - add more flexibility in content handling from Java endpoints

- Version 2.1
  - Add new RabbitMQ usage logging option

- Version 2.2 to 2.2.3
  - Fix issues related to initial testing and production operations
    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.

- Version 2.4
  - Bug fix 884 - correct order of fields, location and channel, in usage log output
  - Bug fix 883 - enable corsEnabled property for error handling so error handling works the same as success handling
  - Feature 875 - add relaxed-validation mode to enable operational flexibility with handler program changes
  - Feature 863 - add IP filtering property per endpoint
  - Feature 862 - add ability to complete loading of configuration file even if some parts of configuration have errors
  - Feature 861 - small updates to text on wsstatus output
  - Feature 859 - change URL check from error to warning when URL specified in the proxyURL on ProxyResource endpoint is not available
  - Bug 857 - enable CLIENTNAME parameter in CmdProcessor environment setup
  - Bug 851, 844 - update WSS to run in Glassfish 4.x server
  - Bug 839 - prevent exception in logAndThrowException handler when briefMsg is null
  - Feature 824 - add new, optional endpoint property, mediaParameter, to designate which endpoint user parameter is for media control

WSS Concepts

The primary objectives of WSS is to enable internet access to data. WSS handles the HTTP communication with clients while executing task specific programs to retrieve the desired data, thus, effectively separating the HTTP component of a web service from the data extraction work.

While the WSS can be extended by writing custom Java code, WSS is designed to add functionality by using existing command-line scripts for data access. Two built-in java code modules, called processors, are provided with WSS to accomplish this:

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

The CmdProcessor processor executes command-line programs and streams requested data back to an HTTP client. The ProxyResource processor streams data back to a client from an existing URL.

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

WSS Operation and Configuration

A WSS service (or application) is composed of WSS built-in endpoints plus endpoints defined in configuration files. Each service will have a unique set of configuration files. Multiple web services can be deployed in one container by combining sets of configuration files with copies of the WSS war file using the conventions described here.

Terms

interface parameters - these are parameters defined in a configuration file which a user can provide in a query to control the operation of a respective endpoint.
parts of a URL - as referenced in this document, are defined as follows:
given this example URL, http://service.iris.edu/fdsnws/event/1/query?minmagnitude=8.5&format=geocsv

the parts and respective values are:

- **protocol** - http
- **host** - service.iris.edu
- **service (or application)** - fdsnws/event/1
- **endpoint** - query
- **interface parameters** - minmagnitude, format (i.e. each term after the '?' or '&' character and before the '=' character)

properties - items defined in configuration files that define operation of WSS
wssConfigDir - a system property that defines a folder that contains the configuration files

Configuration Steps

The general procedure for configuring WSS services is to:

- configure where the container can find configuration files
- determine a **service** name
- create respective configuration files for each service and apply naming convention
- put the appropriate **properties** in the configuration files to get the desired **endpoint** name(s) and desired **client interface parameter(s)** names

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

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”), so in this example, the endpoint is "query".

WSS Configuration Details

Configuring configuration files location

WSS configuration files must be contained in a folder defined by system property **wssConfigDir**.

- for Tomcat - a bin/setenv.sh file can be used to set the system property **wssConfigDir**
- for Glassfish - an administration command can be used, e.g. **asadmin create-system-properties wssConfigDir=$(pwd)/glassfish/domains/domain1/wss_config**

For more details see - https://seiscode.iris.washington.edu/projects/webserviceshell/wiki/Quick_installation_instructions

Service and Configuration File Setup

In WSS, the service name is used to determine a respective set of configuration files. As the container starts the service, first the file names are determined, then WSS opens each file and acquires the configuration information.

- For **Tomcat**, these file names are derived from the deployed .war file name.
  - Tomcat create a context name, then WSS uses the context name to create a filename
  - Tomcat creates a context name by replacing any hash character, i.e. ‘#’ in the war file name with a slash character ‘/’, the context name also determining the **service** part of the URL.
  - WSS uses the context name and replaces any slash character ‘/’ in the file name with a dot character ‘.’ then generates a complete file name by appending -service.cfg, -param.cfg, or -log4j.properties. For example, a war file with name **fdsnws#event#1.war**, determines:
  - **fdsnws/event/1** - for a **service** name
  - **fdsnws.event.1-service.cfg** - for the name of file which contains operational and endpoint parameters
  - **fdsnws.event.1-param.cfg** - for the name of file which contains client interface parameters for each defined endpoints
  - **fdsnws.event.1-log4j.properties** - for the name of file which contains log4j information

- For **Glassfish**, the same pattern applies, the context name may be set with an administrative command,
  - something like **asadmin deploy --contextroot fdsnws/event/1 /path/to/webserviceshell-2.4.war**

Note: For the initial deployment of any service, the respective log files (as defined in the log4j.properties file) should be reviewed to see that there are any file protection or location problems, and if all of the parameters are processes successfully.

Endpoint and Client Interface Parameter Setup

Endpoints are defined by setting WSS properties in a **service.cfg** file. Client interface parameters are added as needed per endpoint by adding properties in a **param.cfg** file. For example, to add an endpoint called "query" which will run a script called "getMyData.sh" and allow two interface parameters, "minmagnitude" and "format", to be specified in a request, do the following:

add these lines to a respective service.cfg file

query.endpointClassName=edu.iris.wss.endpoints.CmdProcessor
query.handlerProgram="/some/path/getMyData.sh"

and add these lines to a param.cfg file

query.minmagnitude=NUMBER
query.format=TEXT

Note: - endpoint names 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

In summary:

- setup a folder to hold all configuration files
- In Tomcat, a war file name may be chosen as needed to get a desired context name, in Glassfish, the contextroot must be set
- Respective configuration file names must correspond to context name.
- Endpoint names may be chosen as needed, then configured in service.cfg.
- Parameter names may be chosen as needed, then configured in param.cfg.

WSS Configuration Reference

Built in endpoints

In addition to endpoints that are added by configuration, there are a few endpoints built in to WSS. These endpoints are listed in the following table.

Endpoint	Default Value	Comments

/	WSS generated information page	Use property "rootServiceDoc" in service.cfg to specify a URL with this service's specific documentation.
wssstatus	status information about this WSS and its configuration	It should be blocked for external IPs, see the property "allowedIPs" for controlling this.
wssversion	current version of Web Service Shell	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 property "version" in service.cfg to set a number or other identifier as desired for this service.

Note: The **wssstatus** endpoint may contain site sensitive information and therefor should not be generally available all client request. To address this concern, a new property, allowedIPs, has been added. See Endpoint Properties below.

Service configuration file (*servicename*-service.cfg)

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

Global Properties

Property Name	Default Value	Description
appName	"notnamed"	This may be defined as desired, it should be different for each service configured - Note, this value is also used in output to "usage" log files as field "application".
version	"notversioned"	This may be defined as desired, it indicates the version of this service. This value is returned by a request to built-in endpoint, "version".
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 or RABBIT_ASYNC.
loggingConfig	null	a file name or URL referencing an IRIS RabbitMQ configuration file. Only required if loggingMethod is set to RABBIT_ASYNC
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, which is the endpoint name, in front of the properties 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 properties as needed (defined in the following table).

to add another endpoint "answer" which is to execute programDEF, do the following:

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

etc.

Endpoint Properties

The term "formatType" refers to the name of a media type that may be provided by an endpoint. Media type is a case insensitive string and one or more may be defined in the formatTypes property.

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 property string, as defined in the Global Properties section.

Property Name	Default Value	Description
		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 WSS, **edu.iris.wss.endpoints.CmdProcessor** and

endpointClassName	edu.iris.wss.endpoints.CmdProcessor	**edu.iris.wss.endpoints.ProxyResource**. CmdProcessor executes a handlerProgram and ProxyResource delivers the contents specified in its proxyURL property.
handlerProgram	"nonespecified"	a fully qualified executable file name. It must be specified when endpointClassName is set to edu.iris.wss.endpoints.CmdProcessor
handlerTimeout	30	inactivity time in seconds before WSS sends a handlerProgram process a SIGTERM. see section "Command-Line Process Time Limits".
handlerWorkingDirectory	/tmp	if specified, it must be a valid folder with write access. It is required to start a handlerProgram process and the handlerProgram may use the folder to write content into.
usageLog	true	when true, log information is written to the usageLogger logger defined in the log4j.properties file, or JMS, or RabbitMQ, depending on the value of loggingMethod
formatTypes	BINARY: application/octet-stream	a list of (formatType, media type) pairs. formatType defines a term which a client can use to select the respective return data media type. When provided, the first pair in the list becomes the default. The BINARY formatType pair is retained in the list and is alway available for selection. Note: A client selects a media type by using the "format" interface parameter and specifying one of the defined formatTypes. The "format" interface parameter may be redefined to another name by using the mediaParameter property.
mediaParameter	format	is available to allow an override to the builtin default parameter "format"
formatDispositions	empty string	a list of (formatType, HTTP Content-Disposition value) pairs that will override the Content-Disposition header set by WSS default or addHeaders property. A Content-Disposition which includes a filename value may have ${UTC} and/or ${appName} appended/inserted so as to avoid name conflicts when files are downloaded by a client.
addHeaders	empty string	a list of HTTP headers that will be included in a response. They are added to the default headers, or they will override any default header with the same name. ${UTC} and/or ${appName} may be used in the filename value.
postEnabled	false	when false, POST request are ignored, when true, the POST body is passed to the respective endpoint processor.
logMiniseedExtents	false	when true, additional Miniseed channel information is collected and written to the usage log, only applies to edu.iris.wss.endpoints.CmdProcessor
use404For204	false	when true and a response has no data, return an HTTP code 404 instead of 204.
relaxedValidation	false	when true, client parameters not defined in param.cfg for this endpoint will not be type checked, but will be passed through, as is, to the handler program.
allowedIPs	empty list (i.e. all IPs allowed)	a list of one or more subnets that may access this endpoint, in CIDR notation. e.g. 192.168.0.1/24 for local IPv4 subnet or 127.0.0.1/32,::1/128 for localhost.
proxyURL	"noproxyURL"	It is only used when endpointClassName is set to edu.iris.wss.endpoints.ProxyResource. When used, it must point to a valid URL.

Managing HTTP headers for a client response

HTTP headers can be set both by endpoint properties 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. apply **addHeaders** property - adds new headers, or overrides default headers.
3. apply **formatDispositions** property- overrides content-disposition header, but only for the respective formatType.
4. apply **command-line handler** provided headers - 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. WSS provides an immediate error response to a client query for parameter type mismatch or parameters not defined in the param.cfg file, thus preventing potentially needless access to a handler. However, if the endpoint property "relaxedValidation" is used, the misspelled names, or undefined names will be pass to the endpoint processor as is and no message is returned to the client.

Reserved Parameters

A few parameter names are reserved for WSS and can only be used as defined here. Reserved names are:

- format
- aliases
- username
- stdin
- nodata

WSS uses the "format" parameter to determine the response media type. When needed, the format parameter must be specified the param.cfg file per endpoint. In a client query, it is used to select one of the formatType names defined in formatTypes property. When format parameter is not specified in for a given endpoint, or the value is not set in the client request, the client response will have the default 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 the usage example below.

"username" is added by WSS to a handlerProgram argument list when a user has been successfully authenticated.

"stdin" is added by WSS to a handlerProgram argument list when a post request has been made. It indicates to the handler that the handler should read stdin to get the post data.

"nodata" is predefined in WSS and can be used on any query. It causes WSS to return an HTTP code 404 instead of 204 when no data is available. The accepted values are "204" or "404". When this parameter is used in a query, it overrides the setting for the endpoint property "use404For204"

param.cfg parameters

Client interface parameter names and respective types are defined per endpoint 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.

The 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.

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

relaxedValidation - Adding Operational Flexibility when calling Handlers

In regular operations, changes to handler parameters requires a change to the respective information in param.cfg file, and therefor a redeployment of the respective web service is needed. To allow testing and more operational flexibility, a new endpoint property, relaxedValidation, is available in this release. When relaxedValidation is set, only parameters define in the param.cfg (if any) will be checked, any other parameters (including potential misspellings or incorrect types) will be passed from the client query to the handler with out validation. Thus, the parameter processing for a handler may be updated without a redeployment of the respective service, or a handler may take undefined parameters without a generating a client query error.

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: the usage log file is only used when loggingMethod is set to LOG4J, otherwise usage log messages are sent to their respective JMS or RabbitMQ destination.

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.

Usage logging

WSS usage logging provides information concerning the amount of data delivered per request. The fields in the log record are described with these headings and this line is written out when a web services is started.

```
# Application|Host Name|Access Date|Client Name|Client IP|Data Length|Processing Time (ms)|Error Type|User Agent|HTTP
```

Notes on data values

Application - is the value given to property appName in service.cfg

"Host Name" and "Client Name" values are not always the names expected because they can be affected by network configurations external to WSS, such as load balancers, firewalls, etc., also DNS lookup performance may be too slow, so IPs are not necessarily resolved to names

"Data Length" is bytes

"Processing Time (ms)" is in milliseconds

"Extra", is for experimental uses, right now, it shows the name of the endpoint providing the log entry

"Message Type" is routing information for miniseed related information

- "usage" is a summary of all the channels of miniseed data for one request
- "wfstat" is information for one channel of miniseed data

Miniseed information

Some of the fields in a log entry only apply when the data provided is miniseed. This information is only provided by the endpoint processor edu.iris.wss.endpoints.CmdProcessor and only when the output media type is "mseed" or "miniseed".

- the CmdProcessor uses edu.sc.seis.seisFile Java code to scan and decode the byte stream, then reports the respective miniseed information
- additionally, the total bytes of miniseed data delivered is shown in the "Data Length" field on records with "Message Type" "usage"

Additional CmdProcessor Features

The edu.iris.wss.endpoints.CmdProcessor provides additional features that may be useful when developing services.

Command-Line Process Time Limits

Once a command-line process starts, the CmdProcessor will wait for handlerTimeout seconds for a process to return data or error messages. When the handlerTimeout time is exceeded, the CmdProcessor sends the process a SIGTERM. A process may catch the SIGTERM and do cleanup as needed such as close databases, write messages to standard err, etc., and terminate. The CmdProcessor will then wait sigkillDelay seconds for the process to end, if the process does not end, a SIGKILL will be sent to the process.

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 handler process. The handler program is expected to test for the argument STDIN, and if found, read the POST data from standard in.

Under normal operation, with no errors, when the CmdProcessor detects any data from the process on standard output, an HTTP 200 is returned the WSS container (e.g. Tomcat or GlassFish), streams the data to the client.

If an error occurs before data streaming starts, the CmdProcessor reads standard error and returns this information in an error response to the client. If an error occurs after streaming starts, error messages are logged.

Setting HTTP Headers From a Command-Line Process

The CmdProcessor has the ability to set response HTTP headers if provided by the command-line process. The CmdProcessor checks for header values when a handler process first writes to STDOUT, if any headers text is found, WSS includes these headers in the response. 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 between marker strings 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 the 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 a user is authenticated, then this value is included: AUTHENTICATEDUSERNAME - authenticated user name

Changes from version 2.x previous to 2.4

WSS has changes to:

- enable relaxedValidation for more flexible handler interaction without redeploying service
- continue processing configuration files even if non critical errors occurred
- continue processing configuration files if proxyURL is not immediately available
- add IP filtering ability
- enable operation in Glassfish container

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 1.x 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 config file 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 config file name as needed
    - change parameter names by prepending endpoint name

3. log4j.properties
    - change web service part of config file name as needed
    - change output log file names as needed

Changes for built in endpoints

Previous Endpoint	New Endpoint	Comment
/	same as 1.x	
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, also set property proxyURL to the URL of the content to proxy, and add property formatTypes to set the default media type.
wssversion	same as 1.x	
whoami	same as 1.x	
version	same as 1.x	
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.cfg related changes - global parameters

Previous Parameter	New Parameter	Comment
appName	same as 1.x	
version	same as 1.x	
corsEnabled	same as 1.x	
rootServiceDoc	same as 1.x	

loggingMethod	same as 1.x	with addition of RABBIT_ASYNC option
-	loggingConfig	new with 2.1, only applies to loggingMethod RABBIT_ASYNC
sigkillDelay	same as 1.x	
singletonClassName	same as 1.x	
rootServicePath	removed	not needed
jndiUrl	removed	not needed
connectionFactory	removed	not needed
topicDestination	removed	not needed
argPreProcessorClassName	removed	not needed

service.cfg related changes - endpoint parameters

Previous Parameter	New Parameter	Comment
streamingOutputClassName	endpointClassName	parameter renamed
handlerProgram	same as 1.x	
handlerTimeout	same as 1.x	
handlerWorkingDirectory	same as 1.x	
usageLog	same as 1.x	
outputTypes	formatTypes	parameter renamed
-	mediaParameter	new parameter
-	formatDispositions	new parameter
-	addHeaders	new parameter
postEnabled	same as 1.x	
-	logMiniseedExtents	new parameter - extra processing and usage logging for Miniseed channel information is **no longer** performed by CmdProcessor unless this parameter is added
use404For204	same as 1.x	
-	relaxedValidation	new parameter
-	allowedIPs	new parameter
-	proxyURL	new parameter, must be used when endpointClassName is set to edu.iris.wss.endpoints.ProxyResource

param.cfg configuration changes

Interface parameter types has not changed. 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. When using the new endpoint property relaxedValidation, interface parameters need not be specified.

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 or RABBIT_ASYNC
loggingMethod=LOG4J
##loggingConfig=local_file_system or URL for  rabbitconfig-publisher.properties file

# time is seconds
sigkillDelay=30

# restrict access to wssstatus
wssstatus.allowedIPs=127.0.0.1/32,::1/128,172.17.0.0/24

# -------- 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

# Content-Disposition overrides for respective media types "text" and "miniseed"
query.formatDispositions= \
  text: inline; filename="mypart_${appName}_${UTC}.txt", \
  miniseed: attachment; filename="a_miniseed_file.mseed"

# 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.base}/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.base}/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
```

rabbitconfig-publisher.properties

```
# Host that's the broker. This will normally be the load balancer
broker=broker1,broker2

# The virtual host within the broker
virtualhost=test

# Internal buffer size for the async publishers
buffersize=10000

# Persistent or not
default_persistence=true

# The exchange name that recieves these messages
exchange=ws_logging

# Credentials
user=hellorabbit
password=hellopw

# Probably never normnally reconnect
reconnect_interval=-1

# How often to wait between failed connection attempts in msec
retry_interval=4000
```