SALT Command Reference

The SALT Command Reference describes system processes and commands delivered with the SALT software.

Table 1 lists SALT commands and functions.

Table 1 SALT Commands and Functions

Name	Description
GWWS(5)	Web service gateway server.
jsoncvt(1)	JSON object to service metadata converter.
tmscd(1)	Command line utility used to activate and deactivate service contract discovery.
tmwsdlgen	WSDL document generator.
wsadmin	SALT administration command interpreter.
wscobolcvt	Generates SALT artifacts from COBOL copybook for exposing COBOL service as a web service.
wsdlcvt	WSDL document converter.

Table 1 SALT Commands and Functions

Name	Description
wsloadcf	Reads SALT Deployment file and other referenced artifacts. Loads a binary SALTCONFIG file.
wsunloadcf	Reads a binary SALTCONFIG file, creates a SALT deployment file and other referenced files (WSDF files, WS-Policy files).

GWWS(5)

Name

GWWS – Web service gateway server.

Synopsis

```
GWWS SRVGRP="identifier" SRVID=number [other_parms]
CLOPT="-A -- -i InstanceID [-a <scheme>://<host>:<port>]"
```

Description

The GWWS server is the Web service gateway for Tuxedo applications, the core component of SALT. The GWWS gateway server provides communication with Web service programs via SOAP 1.1/1.2 protocols. The GWWS server has bi-directional (inbound/outbound) capability. It can accept SOAP requests from Web service applications and passes Tuxedo native calls to Tuxedo services (inbound). It also accepts Tuxedo ATMI requests and passes SOAP calls to Web service applications (outbound). GWWS servers are used as Tuxedo system processes and are described in the *SERVERS section of the UBBCONFIG file.

The CLOPT option is a string of command-line options passed to the GWWS server when it is booted. The GWWS server accepts the following CLOPT options:

-i InstanceID

Specifies the GWWS instance unique ID. It is used to distinguish multiple GWWS instances provided in the same Tuxedo domain. This value *must* be unique among multiple GWWS items within the UBBCONFIG file.

Note: The InstanceID value must be pre-defined in the <WSGateway> section of the SALT Deployment File.

`-a <scheme>://<host>:<port>`

Web administration is disabled by default. In order to enable admin capabilities, the GWWS server must be configured in the `UBBCONFIG` file using the `-a` option as follows: `-a <scheme>://<host>:<port>`.

Note: Use the following URL to access the Web Admin Console:
`<scheme>://<host>:<port>/admin`

`<scheme>`

Can be 'http' or 'https'. If using 'https', GWWS must be configured for SSL capabilities by adding private-key and certificates in the same manner as securing application Web Services with SSL.

`<host>`

The name or IP address of the admin URL listening endpoint.

`<port>`

The port value of the admin URL listening endpoint.

Environment Variables

The `SALTCONFIG` environment variable must be set before the GWWS server is booted. `Accesslog(5)` can be enabled by setting environment variable `TMENABLEALOG=y`.

Note: Windows platforms: Add `%TUXDIR%\bin\ssllibs to PATH`.

Deprecation

The following SALT 1.1 `GWWS` parameter is deprecated in the current release.

`-c Config_file`

Specifies the SALT 1.1 configuration file.

Note: Starting with the SALT 2.0 release, the `GWWS` server loads the SALT configuration from the binary `SALTCONFIG` file instead of the XML-based configuration file. The configuration file is no longer a GWWS server input parameter. The `SALTCONFIG` file must be generated using `wsloadcf` before booting `GWWS` servers.

Diagnostics

For inbound call, if an error occurs during SOAP message processing, the error is logged. The error is also translated into appropriate SOAP fault and/or HTTP error status code and returned to the Web service client.

For outbound call, if an error occurs during processing, the error is logged. The error is also translated into appropriate Tuxedo system error code (`tperrno`) and returned to the Tuxedo client.

Example(s)

Listing 1 GWWS Description in the UBBCONFIG File

```
*SERVERS

GWWS SRVGRP=GROUP1 SRVID=10
      CLOPT="-A -- -i GW1"
GWWS SRVGRP=GROUP1 SRVID=11
      CLOPT="-A -- -i GW2"
GWWS SRVGRP=GROUP2 SRVID=20
      CLOPT="-A -- -i GW3"
```

See Also

UBBCONFIG(5)

tmwsdlgen

SALT Deployment File Reference

SALT Web Service Definition File Reference

jsoncvt(1)

Name

`jsoncvt` – JSON object to service metadata converter.

Synopsis

```
jsconcvt [-f inout.json [-f inout2.json ...]] [-i input.json [-i input2.json
...]] [-o output.json [-o output2.json ...]] -s servicename -m(POST / GET /
PUT / DELETE) -a serviceaddres
```

Description

The `jsconcvt` command generates service metadata from JSON content, which can be used to construct service interfaces for easier application development. The command also generates fml32 tables and SALT deployment service definitions.

It is possible to mention more than 1 of each input and output payload samples by either separating the file names with spaces and enclosing the list in double-quotes, or specifying "`-i`" or "`-o`" multiple times.The corresponding definitions are concatenated in the metadata and fml32. This is to accommodate services that may return or accept data in different formats.

Input and output are optional, although specifying neither is not accepted. The metadata and fml32 files are generated using the service name as the base name. For example, a switch of "`-s service1`" generates files with names "`service1.mif`" and "`service1.fml32`".

Note: `jsoncvt` does not support long double value.

Parameters/Options

`-f`

Sample input and output json content. `-jsoncvt` uses this to generate a metadata service structure for the data received and returned by this service.

`-i`

Sample input json content. `-jsoncvt` uses this to generate a metadata service structure for the data received by this service.

`-o`

Sample output json content - `jsoncvt` uses this to generate a metadata service structure for the data returned by this service.

`-s`

Name of the service generated. Specified in metadata and SALT deploy file.

`-m`

Method for the service. Can be one of the following: `POST`, `GET`, `PUT`, or `DELETE`.

`-a`

Address of the external service.

Diagnostics

Error, warning or information messages are output to standard output.

Environment Variables

The `TUXDIR` and `LANG` environment variables must be set correctly.

Examples

Given the following JSON examples.

- Input/Output

- Running the Command
- Results

Input/Output

Input and output file examples are shown in Listing 2 and Listing 3.

Listing 2 Input

```
$ cat balance.json
{
       "account":5563909,
       "location":"US"
}
```

Listing 3 Output

```
$ cat result.json
{
       "account":5563909,
       "location":"US",
       "accounts": [
              {
                     "type":"savings",
                     "currency":"US Dollars",
                     "balance":35000.34
              },
              {
                     "type":"checking",
                     "currency":"US Dollars",
                     "balance":500.15
              }
       ]
}
```

Running the Command

An example of running the comand is shown in Listing 4.

Listing 4 Running the command

```
$ jsoncvt -i balance.json -o result.json -s balance -m POST -a
http://bank.com:4434/online_banking
Files balance.mif, balance.fml32 and balance.dep generated.
Please add the generated service definition – balance.dep – to the SALT
deploy configuration file.
```

Results

The end results are shown in Listing 5, Listing 6, and Listing 7.

Listing 5 MIF

```
service=balance
tuxservice=balance
export=y
servicetype=service
servicemode=webservice
inbuf=FML32
outbuf=FML32

param=account
access=in
type=long

param=location
access=in
type=string

param=account
access=out
```

```
type=long
param=location
access=out
type=string
param=accounts
access=out
type=fml32
(
        param=type
        access=out
        type=string

        param=currency
        access=out
        type=string

        param=balance
        access=out
        type=double
```

Listing 6 fml32 Table

```
# FML32 JSON Mapping Generated by jsoncvt

*base 30000 # Customize base number if necessary.

#name      rel-number  type     flags  comment
#----      ----------  ----     -----  -------
account       1        long     -      param
location      2        string   -      param
options       3        string   -      param
accounts      4        fml32    -      structured parameter
type          5        string   -      param
currency      6        string   -      param
```

```
balance       7       long     -       param
```

Listing 7 SALT Deploy Definition

```
      <service name=balance
             content-type="JSON" output-buffer="FML32"
             address="http://bank.com:4434/online_banking"/>
```

See Also

Creating the Oracle Tuxedo Service Metadata Repository

`field_tables(5)`

SALT Web Service Definition File Reference

tmscd(1)

Name

`tmscd(1)` – Activates and deactivates service contract discovery.

Synopsis

```
tmscd start|stop|status [-e] [-f <file>][id1 [ id2 [ ...]]]
```

Description

The `tmscd` command line utility is used to activate and deactivate service contract discovery.

Parameters and Options

`tmscd` accepts following parameters and options:

`start|stop|status`
Required. Starts, stops, or displays service contract dictionary settings for specific services, or all services if none are specified. A `start` or `stop` request for a service that has already activated or deactivated contract discovery is ignored. Effective service information is displayed when handling the requests.

Note: `start|stop|status` must occur after `-e` and `-f` , if either of those options are specified.

`[-e]`

Specifies the service scope as a regular expression.

`[-f <file>]`

The service scope is defined in the given `<file>`. The file may contain sections to group related definitions together. All entries for a section must be written together line-by-line.

Empty lines or lines starting with '#' are ignored. Lines starting with '*' are section lines. Other lines are `"id=content"` definitions.

`id1 id2 ...`

Indicates one or more services. If `-e` is specified, a regular expression is used to match the service name. If `-e` is not specified, the service name is matched exactly.

Example(s)

Example 1 - start discovery for `TOUPPER`, `TOLOWER`:

```
tmscd start TOUPPER TOLOWER
```

Example 2 - start discovery for services started with `TO` and `BR`:

```
tmscd -e start TO.* BR.*
```

Example 3 - same request as example 1 but via file:

```
tmscd -f svcfile start id1 id2
```

Note: The first found definition is used if section is not provided:

Example 4 - same request as example 2 but via file:

```
tmscd -e -f svcfile start case4.svcs
```

Listing 8 shows content of the file named "`svcfile`".

Listing 8 svcfile Content

```
# file: svcfile
*case3
id1     = TOUPPER
id2     = TOLOWER
```