## Contents

Preface .............................................................................................................................................................. xiii
   Audience ..................................................................................................................................................... xiii
   Documentation Accessibility ................................................................................................................... xiii
   Related Documents.................................................................................................................................... xiii
   Conventions................................................................................................................................................ xiii

1 Overview of the WebLogic Scripting Tool for Oracle Traffic Director
   1.1 Command-Line Interface.................................................................................................................. 1-1
   1.2 Usage Modes ..................................................................................................................................... 1-1
       1.2.1 Interactive Mode.................................................................................................................... 1-2
       1.2.2 Script Mode ............................................................................................................................ 1-2
       1.2.3 Embedded Mode ................................................................................................................... 1-2
   1.3 Changes from Release 11g.............................................................................................................. 1-3
   1.4 Offline Commands ............................................................................................................................ 1-3
       1.4.1 Offline Provisioning.............................................................................................................. 1-3
       1.4.2 Monitoring............................................................................................................................. 1-4
       1.4.3 SNMP Runtime Management.............................................................................................. 1-4
       1.4.4 Failover Runtime Management........................................................................................ 1-4
   1.5 OTD Custom WLST Command Usage.......................................................................................... 1-4
       1.5.1 Syntax...................................................................................................................................... 1-4
       1.5.2 Use with WLST ...................................................................................................................... 1-4
       1.5.3 Arguments.............................................................................................................................. 1-4
       1.5.4 Return Values......................................................................................................................... 1-4
       1.5.5 Error Messages....................................................................................................................... 1-5
       1.5.6 Unable to Unset/Disable the URI Mapping on a Route.................................................. 1-5
   1.6 List of Commands............................................................................................................................ 1-5
       1.6.1 Provisioning ........................................................................................................................... 1-5
       1.6.2 Instance Management ........................................................................................................... 1-5
       1.6.3 Configuration Deployment................................................................................................. 1-6
       1.6.4 Configuration Management ............................................................................................... 1-6
       1.6.5 Virtual Server Management.............................................................................................. 1-7
       1.6.6 TCP Load Balancer Management....................................................................................... 1-8
# Oracle Traffic Director WLST Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>activate</td>
<td>2-1</td>
</tr>
<tr>
<td>deleteKeyStoreEntry</td>
<td>2-2</td>
</tr>
<tr>
<td>displayLogs</td>
<td>2-3</td>
</tr>
<tr>
<td>displayMetricTables</td>
<td>2-5</td>
</tr>
<tr>
<td>enableOverwriteComponentChanges</td>
<td>2-7</td>
</tr>
<tr>
<td>exportKeyStoreCertificate</td>
<td>2-8</td>
</tr>
<tr>
<td>exportKeyStoreCertificateRequest</td>
<td>2-9</td>
</tr>
<tr>
<td>generateKeyPair</td>
<td>2-10</td>
</tr>
<tr>
<td>getKeyStoreCertificates</td>
<td>2-11</td>
</tr>
<tr>
<td>help</td>
<td>2-12</td>
</tr>
<tr>
<td>importKeyStoreCertificate</td>
<td>2-12</td>
</tr>
<tr>
<td>listExpiringCertificates</td>
<td>2-14</td>
</tr>
<tr>
<td>listKeyStores</td>
<td>2-15</td>
</tr>
<tr>
<td>listKeyStoreAliases</td>
<td>2-15</td>
</tr>
<tr>
<td>otd_addFailoverInstance</td>
<td>2-16</td>
</tr>
<tr>
<td>otd_blockProxyInfo</td>
<td>2-17</td>
</tr>
<tr>
<td>otd_copyConfiguration</td>
<td>2-18</td>
</tr>
<tr>
<td>otd_copyVirtualServer</td>
<td>2-19</td>
</tr>
<tr>
<td>otd_createCacheRule</td>
<td>2-19</td>
</tr>
<tr>
<td>otd_createCompressionRule</td>
<td>2-20</td>
</tr>
<tr>
<td>otd_createConfiguration</td>
<td>2-21</td>
</tr>
<tr>
<td>otd_createContentRule</td>
<td>2-22</td>
</tr>
<tr>
<td>otd_createErrorPage</td>
<td>2-23</td>
</tr>
<tr>
<td>otd_createEvent</td>
<td>2-24</td>
</tr>
<tr>
<td>otd_createFailoverGroup</td>
<td>2-25</td>
</tr>
<tr>
<td>otd_createHttpListener</td>
<td>2-27</td>
</tr>
<tr>
<td>otd_createInstance</td>
<td>2-29</td>
</tr>
<tr>
<td>otd_createMimeType</td>
<td>2-30</td>
</tr>
<tr>
<td>otd_createOriginServer</td>
<td>2-31</td>
</tr>
<tr>
<td>otd_createOriginServerPool</td>
<td>2-32</td>
</tr>
<tr>
<td>otd_createRequestLimit</td>
<td>2-33</td>
</tr>
</tbody>
</table>
1.75  otd_disableVirtualServerResponseBandwidthLimit ................................................................. 2-64
1.76  otd_enableOriginServerPoolMaintenance ............................................................................. 2-65
1.77  otd_enablePerfDump .................................................................................................................. 2-66
1.78  otd_enableRouteAuth ................................................................................................................ 2-66
1.79  otd_enableRouteBandwidthLimit ............................................................................................. 2-67
1.80  otd_enableRequestLimitEvents ............................................................................................... 2-68
1.81  otd_enableStatsXml .................................................................................................................. 2-69
1.82  otd_enableStatusListener ......................................................................................................... 2-70
1.83  otd_enableWebAppFirewall ...................................................................................................... 2-70
1.84  otd_enableVirtualServerAccessLog ......................................................................................... 2-71
1.85  otd_enableVirtualServerRequestBandwidthLimit ................................................................. 2-72
1.86  otd_enableVirtualServerResponseBandwidthLimit ............................................................... 2-73
1.87  otd_exportKeyStore ................................................................................................................. 2-74
1.88  otd_forwardProxyInfo .............................................................................................................. 2-75
1.89  otd_getAccessLogBufferProperties ......................................................................................... 2-77
1.90  otd_getCacheProperties .......................................................................................................... 2-77
1.91  otd_getCacheRuleProperties .................................................................................................... 2-78
1.92  otd_getCompressionRuleProperties ...................................................................................... 2-78
1.93  otd_getConfigFile ...................................................................................................................... 2-79
1.94  otd_getConfigurationAccessLogProperties .......................................................................... 2-80
1.95  otd_getConfigurationCrlProperties ......................................................................................... 2-81
1.96  otd_getConfigurationProperties ........................................................................................... 2-81
1.97  otd_getContentRuleProperties ............................................................................................... 2-82
1.98  otd_getDnsCacheProperties ................................................................................................. 2-82
1.99  otd_getDnsProperties ............................................................................................................ 2-83
2.100 otd_getEventProperties .......................................................................................................... 2-83
2.101 otd_getEventSubscriptionProperties .................................................................................... 2-84
2.102 otd_getFileCacheProperties .................................................................................................. 2-85
2.103 otd_getFailoverGroupProperties .......................................................................................... 2-85
2.104 otd_getHealthCheckProperties ............................................................................................. 2-86
2.105 otd_getHttpListenerProperties ............................................................................................ 2-87
2.106 otd_getHttpListenerSslProperties ......................................................................................... 2-88
2.107 otd_getHttpProperties ........................................................................................................ 2-88
2.108 otd_getHttpThreadPoolProperties ......................................................................................... 2-89
2.109 otd_getKeepaliveProperties ................................................................................................. 2-89
2.110 otd_getLogProperties ........................................................................................................... 2-90
2.111 otd_getOriginServerPoolMaintenanceProperties ............................................................. 2-90
2.112 otd_getOriginServerPoolProperties ....................................................................................... 2-91
2.113 otd_getOriginServerProperties ............................................................................................ 2-92
2.114 otd_getOriginServerPoolSslProperties ............................................................................... 2-92
2.115 otd_getPartitionAccessLogProperties .................................................................................. 2-93
2.116 otd_getPerfDump .................................................................................................................... 2-94
2.117 otd_getPerfDumpProperties .................................................................................................. 2-95
Index
<table>
<thead>
<tr>
<th>Example</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>2-17</td>
</tr>
<tr>
<td>2-2</td>
<td>2-48</td>
</tr>
<tr>
<td>2-3</td>
<td>2-60</td>
</tr>
<tr>
<td>2-4</td>
<td>2-62</td>
</tr>
<tr>
<td>2-5</td>
<td>2-69</td>
</tr>
<tr>
<td>2-6</td>
<td>2-70</td>
</tr>
<tr>
<td>2-7</td>
<td>2-84</td>
</tr>
<tr>
<td>2-8</td>
<td>2-102</td>
</tr>
<tr>
<td>2-9</td>
<td>2-102</td>
</tr>
<tr>
<td>2-10</td>
<td>2-118</td>
</tr>
<tr>
<td>2-11</td>
<td>2-129</td>
</tr>
<tr>
<td>2-12</td>
<td>2-143</td>
</tr>
<tr>
<td>2-13</td>
<td>2-144</td>
</tr>
<tr>
<td>2-14</td>
<td>2-170</td>
</tr>
</tbody>
</table>
Preface

This document provides information about custom WLST commands that can be used to manage Oracle Traffic Director.

Audience

This book is intended for administrators. This book assumes you are familiar with the following topics:

- Installing software
- Issuing commands in a terminal window
- Oracle WebLogic Server administrative tasks

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information, see the following documents, which are available on the Oracle Technology Network:

- Administering Oracle Traffic Director
- Configuration File Reference for Oracle Traffic Director
- Installing Oracle Traffic Director
- Using WebLogic Server MT

Conventions

The following text conventions are used in this document:
<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><strong>monospace</strong></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
Overview of the WebLogic Scripting Tool for Oracle Traffic Director

This chapter provides an overview of the general syntax, usage modes, WLST (WebLogic Scripting Tool) commands, common options, variables, security, and error messages that are relevant to Oracle Traffic Director.

This guide defines all the custom WLST commands supported for configuring and administering Oracle Traffic Director in Release 12.2.1.1.0 Starting with Release 12.2.1, Oracle Traffic Director administration is built on a Common Admin Model (CAM). CAM brings system components such as Oracle HTTP Server, and Oracle Traffic Director, into the WebLogic domain. Compared to Release 11g, there is no longer an Oracle Traffic Director Administration Server. CAM requires you to install Oracle WebLogic, create an Oracle WebLogic domain and use Oracle WebLogic Administration Server to manage Oracle Traffic Director.

Oracle Traffic Director exposes all the configuration tasks via configuration mbeans which will be used to manage Oracle Traffic Director configurations and instances. The custom WLST commands are wrappers which interact with these mbeans.

1.1 Command-Line Interface

The command line interface in Oracle Traffic Director 12.2.1 is WLST (Weblogic Scripting Tool). The WLST scripting environment is based on Jython which is an implementation of the Python language for the Java platform. The tool can be used both online and offline. For more information on WLST and various online and offline commands, see Oracle WebLogic Scripting Tool. Oracle Traffic Director ships with custom WLST commands that you can run using WLST.

Note:

Oracle Traffic Director ships a wlst.sh wrapper <oracle_home>/otd/common/bin/wlst.sh which initializes the required environment and libraries for Oracle Traffic Director commands. All Oracle Traffic Director custom commands can only be executed from this wlst.sh.

1.2 Usage Modes

You can use the following techniques to invoke Oracle Traffic Director custom commands. For more information on using WLST in these modes, see Using the WebLogic Scripting Tool in Oracle WebLogic Scripting Tool.
1.2.1 Interactive Mode

In the interactive mode, the WLST scripting shell maintains a persistent connection with an instance of WebLogic Server. You can enter an Oracle Traffic Director command and view the response at the command line prompt.

# Launch wlst.sh
<oracle_home>/otd/common/bin/wlst.sh

# Connect to WLS admin server
> connect('weblogic', 'welcome1','t3://localhost:7001')

# Execute an OTD command - list existing configurations
> otd_listConfigurations()
['origin-server-1', 'test', 'origin-server-2', 'origin-server-3']

# Execute another command - get http properties of configuration 'test'
> props={'configuration': 'test'}
> ret = otd_getHttpProperties(props)
> print ret
{'ecid': 'true', 'unchunk-timeout': '60', 'discard-misquoted-cookies': 'true', 'max-request-headers': '64', 'favicon': 'true', 'request-body-timeout': '-1', 'request-header-buffer-size': '8192', 'etag': 'true', 'max-unchunk-size': '8192', 'io-timeout': '30', 'body-buffer-size': '1024', 'output-buffer-size': '8192', 'websocket-strict-upgrade': 'false', 'strict-request-headers': 'false', 'request-header-timeout': '30', 'server-header': None}

1.2.2 Script Mode

Scripts invoke a sequence of WLST commands without requiring interactive input, much like a shell script. Scripts contain WLST commands in a text file with a .py file extension.

1.2.3 Embedded Mode

In embedded mode, the WLST interpreter can be instantiated in Java code and used to run WLST commands and scripts. To run Oracle Traffic Director commands in embedded mode, you must extend the environment to include Oracle Traffic Director commands and libraries as follows:

1. Extend the Java classpath to include <oracle_home>/otd/lib/admin.jar.

2. Set the weblogic.wlstHome Java system property to point to <oracle_home>/otd/common/wlst.

   For example, -Dweblogic.wlstHome=<oracle_home>/otd/common/wlst

3. Write a Java program to invoke Oracle Traffic Director commands:

   package oracle.otd.wlst;

   import weblogic.management.scripting.utils.WLSTInterpreter;
   import org.python.util.InteractiveInterpreter;

   public class WLSTClient
   {
       public static void main(String[] args)
       {
InteractiveInterpreter interpreter = new WLSTInterpreter();
interpreter.exec("connect('weblogic', 'weblogic1', 't3://localhost:1894')");
interpreter.exec("print otd_listConfigurations()"};

1.3 Changes from Release 11g

Starting with Release 12.2.1, WLST is the equivalent of the Oracle Traffic Director tadm command line in Release 11g. Unlike in Release 11g, you can only run the commands in script mode and not in standalone mode. The commands are implemented as WLST custom command functions, they are not hyphenated and follow the pattern 'otd_MixedCaseCommandName'. For example, the create-config command in Release 11g is the otd_createConfiguration command in Release 12.2.1. There are no operands in Release 12.2.1. All the options are passed to the command in a python dict as name-value pairs.

1.4 Offline Commands

The following Oracle Traffic Director WLST commands can be executed in offline mode directly on the host where the Oracle Traffic Director instance/admin server is configured.

1.4.1 Offline Provisioning

The following commands can be used for offline provisioning where we could create and delete Oracle Traffic Director configurations and instances on the admin server after creating and extending the domain with Oracle Traffic Director domain template. These commands does not require admin server to be running and should be executed directly on the host where the admin server resides.

As these are offline commands, you need not execute activate for changes to be applied. Ensure that there is no open edit session while running these commands as these would manipulate the config-store directly and the changes will not be applied in the edit session unless the admin server is restarted.

- otd_createConfiguration
- otd_deleteConfiguration
- otd_listConfigurations
- otd_createInstance
- otd_deleteInstance
- otd_listInstances

---

**Note:**

You cannot invoke the above commands in offline mode until a domain has been read using readDomain. Make sure to update the domain using updateDomain after the command for changes to be applied.
1.4.2 Monitoring
The following commands can be used for monitoring the statistics pertaining to an instance by executing the commands directly on the host where the OTD instance resides:

- `otd_getStatsXml`
- `otd_getPerfDump`

1.4.3 SNMP Runtime Management
The following commands can be used to start/stop SNMP sub-agent by executing the commands directly on the host corresponding to the machine:

- `otd_startSnmpSubAgent`
- `otd_stopSnmpSubAgent`

1.4.4 Failover Runtime Management
The following commands can be executed to start/stop failover on the instance by executing the commands directly on the host where the OTD instance resides:

- `otd_startFailover`
- `otd_stopFailover`

1.5 OTD Custom WLST Command Usage
All OTD custom WLST commands are implemented as jython functions with options (if any) passed as function arguments.

1.5.1 Syntax

```
> <otd_custom_command>(props) or <otd_custom_command>()
```

1.5.2 Use with WLST
Unless specified otherwise, the commands can only be executed online where the connection to a running server is needed. If mentioned as Offline, the command can be executed directly on the host where the Oracle Traffic Director instances are to be configured.

While specifying the path in WLST commands in a Microsoft Windows system, ensure that the path is provided with double backslash only. For example, `C:\\newlne`.

1.5.3 Arguments
The commands either take no argument or a python dictionary as an argument. All the properties are passed to the command in python dictionary as name-value pairs with both name and the value being strings.

1.5.4 Return Values
Unless specified otherwise, all the getter commands (otd_getX) return a python dictionary with properties as name (string)-value (string) pairs, setters (otd_setX)
and create/delete do not return any value while list methods (\texttt{otd\_listX}) return a list of python dictionaries of name (string)-value (string) pairs

### 1.5.5 Error Messages

In case of an error, all the commands throw a \texttt{WLSTException} with an exception message ID in the format 'OTD-XXXXX', and a description. For example:

\texttt{WLSTException: OTD-67853 Object does not exist: oracle.otd.admin:type=Configuration,configuration=test1}

### 1.5.6 Unable to Unset/Disable the URI Mapping on a Route

To unset a value for any property, ensure that you enter "None". Leaving an empty string does not unset a property.

### 1.6 List of Commands

This section contains the functional list of WLST commands that are used in . Using this section you can look for specific commands based on the functional role of

#### 1.6.1 Provisioning

Commands for provisioning a collocated or standalone domain.

**1.6.1.1 Collocated Domain**

The following commands are for provisioning a collocated domain:

- \texttt{otd\_createConfiguration}
- \texttt{otd\_deleteConfiguration}
- \texttt{otd\_listConfigurations}
- \texttt{otd\_createInstance}
- \texttt{otd\_deleteInstance}
- \texttt{otd\_listInstances}

**1.6.1.2 Standalone Domain**

The following commands are for provisioning a standalone domain.

- \texttt{otd\_createStandaloneDomain}
- \texttt{otd\_createStandaloneInstance}
- \texttt{otd\_deleteStandaloneInstance}

#### 1.6.2 Instance Management

The following are instance management commands:

- \texttt{start}
- \texttt{stop}
- \texttt{state}
• softRestart
• otd_rotateLog

1.6.3 Configuration Deployment
The following are configuration deployment commands:
• activate
• undo
• stopEdit
• showComponentChanges
• pullComponentChanges
• resync/resyncAll
• enableOverwriteComponentChanges

1.6.4 Configuration Management
Commands for configuration management.

1.6.4.1 Configuration
The following are configuration commands:
• otd_copyConfiguration
• otd_listConfigFiles
• otd_getConfigFile
• otd_saveConfigFile
• otd_deleteConfigFile

1.6.4.2 Setting/Tuning
The following are setting/tuning commands:
• otd_setConfigurationProperties
• otd_getConfigurationProperties
• otd_setHttpProperties
• otd_getHttpProperties
• otd_setKeepaliveProperties
• otd_getKeepaliveProperties
• otd_setHttpThreadPoolProperties
• otd_getHttpThreadPoolProperties
• otd_setTcpThreadPoolProperties
1.6.5 Virtual Server Management

Commands for management of virtual server configuration and properties.

1.6.5.1 Configuration

The following are configuration commands:

- otd_createVirtualServer
- otd_deleteVirtualServer
- otd_listVirtualServers
- otd_listVirtualServers
1.6.5.2 Setting/Tuning

The following are setting/tuning commands:

- `otd_setVirtualServerProperties`
- `otd_getVirtualServerProperties`
- `otd_createErrorPage`
- `otd_deleteErrorPage`
- `otd_listErrorPages`
- `otd_createVirtualServerVariable`
- `otd_deleteVirtualServerVariable`
- `otd_listVirtualServerVariables`
- `otd_disableVirtualServerResponseBandwidthLimit`
- `otd_enableVirtualServerResponseBandwidthLimit`
- `otd_getVirtualServerRequestBandwidthLimitProperties`
- `otd_disableVirtualServerRequestBandwidthLimit`
- `otd_enableVirtualServerRequestBandwidthLimit`
- `otd_getVirtualServerRequestBandwidthLimitProperties`

1.6.6 TCP Load Balancer Management

The following are commands for TCP load balancer management:

- `otd_createTcpProxy`
- `otd_deleteTcpProxy`
- `otd_listTcpProxies`
- `otd_setTcpProxyProperties`
- `otd_getTcpProxyProperties`

1.6.7 Server Pool Management

Commands for server pool management.

1.6.7.1 Server Pool

The following are server pool management commands:

- `otd_createOriginServerPool`
- `otd_deleteOriginServerPool`
- `otd_listOriginServerPools`
- `otd_getOriginServerPoolProperties`
1.6.7.2 Health Check
The following are health check commands:

- otd_getHealthCheckProperties
- otd_setHealthCheckProperties

1.6.7.3 Origin Server
The following are origin server commands:

- otd_createOriginServer
- otd_deleteOriginServer
- otd_listOriginServers
- otd_setOriginServerProperties
- otd_getOriginServerProperties

1.6.7.4 Maintenance
The following are maintenance commands:

- otd_enableOriginServerPoolMaintenance
- otd_disableOriginServerPoolMaintenance
- otd_getOriginServerPoolMaintenanceProperties

1.6.8 Listener Management
Commands for managing listeners.

1.6.8.1 HTTP
The following are HTTP listener commands:

- otd_createHttpListener
- otd_deleteHttpListener
- otd_listHttpListeners
- otd_setHttpListenerProperties
- otd_getHttpListenerProperties

1.6.8.2 TCP
The following are TCP listener commands:

- otd_createTcpListener
- otd_deleteTcpListener
- otd_listTcpListeners
- otd_setTcpListenerProperties
• otd_getTcpListenerProperties

1.6.9 SSL Management
Commands for managing SSL.

1.6.9.1 Certificate Management
The following are certificate management commands:
• listKeyStores
• generateKeyPair
• listKeyStoreAliases
• getKeyStoreCertificates
• exportKeyStoreCertificateRequest
• importKeyStoreCertificate
• exportKeyStoreCertificate
• listExpandingCertificates
• deleteKeyStoreEntry
• otd_setWalletPassword
• otd_exportKeyStore
• otd_listCertificates

1.6.9.2 SSL Settings
The following are commands for SSL settings:
• otd_setVirtualServerSslProperties
• otd_getVirtualServerSslProperties
• otd_setHttpListenerSslProperties
• otd_getHttpListenerSslProperties
• otd_setTcpListenerSslProperties
• otd_getTcpListenerSslProperties
• otd_setOriginServerPoolSslProperties
• otd_getOriginServerPoolSslProperties

1.6.9.3 Ciphers
The following are the ciphers supported by the server.
• SSL_RSA_WITH_RC4_128_SHA
• SSL_RSA_WITH_3DES_EDE_CBC_SHA
• TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
• TLS_ECDHE_RSA_WITH_RC4_128_SHA
• TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
• TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
• TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
• TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
• TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
• TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
• TLS_RSA_WITH_AES_128_CBC_SHA
• TLS_RSA_WITH_AES_256_CBC_SHA
• TLS_RSA_WITH_AES_128_CBC_SHA256
• TLS_RSA_WITH_AES_256_CBC_SHA256
• TLS_RSA_WITH_AES_128_GCM_SHA256
• TLS_RSA_WITH_AES_256_GCM_SHA384
• TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
• TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
• TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
• TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
• TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
• TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
• TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
• TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

1.6.10 Rules Management

Commands for rules management.

1.6.10.1 Routes

The following are routes commands:

• otd_createRoute
• otd_deleteRoute
• otd_listRoutes
• otd_setRouteProperties
• otd_getRouteProperties
- `otd_enableRouteAuth`
- `otd_disableRouteAuth`
- `otd_listProxyInfo`
- `otd_forwardProxyInfo`
- `otd_blockProxyInfo`
- `otd_enableRouteBandwidthLimit`
- `otd_disableRouteBandwidthLimit`
- `otd_getRouteBandwidthLimitProperties`

**1.6.10.2 Proxy Cache Rules**
The following are proxy cache rules commands:

- `otd_createCacheRule`
- `otd_deleteCacheRule`
- `otd_listCacheRules`
- `otd_setCacheRuleProperties`
- `otd_getCacheRuleProperties`

**1.6.10.3 Request Limit Rules**
The following are request limit rules commands:

- `otd_createRequestLimit`
- `otd_deleteRequestLimit`
- `otd_listRequestLimits`
- `otd_setRequestLimitProperties`
- `otd_getRequestLimitProperties`

**1.6.10.4 Compression Rules**
The following are compression rules commands:

- `otd_createCompressionRule`
- `otd_deleteCompressionRule`
- `otd_listCompressionRules`
- `otd_setCompressionRuleProperties`
- `otd_getCompressionRuleProperties`

**1.6.10.5 Content Rules**
The following are content rules commands:
1.6.11 Web Application Firewall (WAF) Management
Commands for Web Application Firewall (WAF) management.

1.6.11.1 Configuration
The following are setting/tuning commands:
- otd_enableWebAppFirewall
- otd_disableWebAppFirewall
- otd_setWebappFirewallProperties
- otd_getWebappFirewallProperties

1.6.11.2 Ruleset File Management
The following are setting/tuning commands:
- otd_installConfigurationWebappFirewallRulesetFile
- otd_installVirtualServerWebappFirewallRulesetFile
- otd_deleteConfigurationWebappFirewallRulesetFile
- otd_deleteVirtualServerWebappFirewallRulesetFile
- otd_listConfigurationWebappFirewallRulesetFiles
- otd_listVirtualServerWebappFirewallRulesetFiles

1.6.12 Monitoring
Commands for monitoring.

1.6.12.1 Runtime Statistics
The following are commands for displaying runtime statistics:
- otd_getStatsXml
- otd_getPerfDump
- displayMetricTables

1.6.12.2 Setting/Tuning
The following are commands for setting/tuning monitoring settings:
- otd_enableStatsXml
• otd_enableStatsXml
• otd_getStatsXmlProperties
• otd_enablePerfDump
• otd_disablePerfDump
• otd_getPerfDumpProperties
• otd_setStatsProperties
• otd_getStatsProperties

1.6.12.3 SNMP Configuration
The following are commands for SNMP configuration:

• otd_setSnmpProperties
• otd_getSnmpProperties

1.6.12.4 SNMP Runtime Management
The following are commands for SNMP runtime management:

• otd_startSnmpSubAgent
• otd_stopSnmpSubAgent

1.6.13 Logging Configuration
The following are logging configuration commands:

• otd_setLogProperties
• otd_getLogProperties
• otd_setAccessLogBufferProperties
• otd_getAccessLogBufferProperties
• otd_setConfigurationAccessLogProperties
• otd_getConfigurationAccessLogProperties
• otd_enableVirtualServerAccessLog
• otd_disableVirtualServerAccessLog
• otd_getVirtualServerAccessLogProperties
• otd_setTcpAccessLogProperties
• otd_getTcpAccessLogProperties
• displayLogs

1.6.14 Events
The following are events commands:
• otd_createEvent
• otd_deleteEvent
• otd_listEvents
• otd_getEventProperties
• otd_setEventProperties

1.6.15 Failover Management

The following are failover management commands:

1.6.15.1 Configuration

The following are configuration commands:

• otd_createFailoverGroup
• otd_deleteFailoverGroup
• otd_getFailoverGroupProperties
• otd_toggleFailoverGroupPrimary
• otd_listFailoverGroups

1.6.15.2 Runtime Management

The following are runtime management commands:

• otd_startFailover
• otd_stopFailover

1.6.16 Multi-tenancy (with WebLogic Server MT)

Commands for use with Oracle Traffic Director in a WebLogic Server MT environment.

• otd_listPartitions
• otd_listResourceGroups
• otd_getPartitionAccessLogProperties
• otd_setPartitionAccessLogProperties
2

Oracle Traffic Director WLST Commands

This chapter lists and describes the WebLogic Scripting Tool (WLST) commands and their options for Oracle Traffic Director in alphabetical order.

2.1 activate

Description
Activates changes saved during the current editing session but not yet deployed. This command prints a message if a server restart is required for the changes that are being activated.

The activate command returns the latest ActivationTask MBean which reflects the state of changes that a user is currently making or has made recently. You can then invoke methods to get information about the latest Configuration Manager activate task in progress or just completed. In the event of an error, the command returns a WLSTException.

Use this command to deploy the configuration changes to the instances. Note that this command will deploy only the changes done after starting an edit session by executing the command startEdit. Also, the effect of this command is not limited to Oracle Traffic Director. All the changes done after starting an edit session to the various other components and managed servers will also be deployed.

Syntax
activate([timeout], [block])

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>Optional. Time (in milliseconds) that WLST waits for the activation of configuration changes to complete before canceling the operation. A value of -1 indicates that the operation will not time out. This argument defaults to 300,000 ms (or 5 minutes).</td>
</tr>
<tr>
<td>block</td>
<td>Optional. Boolean value specifying whether WLST should block user interaction until the command completes. This argument defaults to false, indicating that user interaction is not blocked. In this case, WLST returns control to the user after issuing the command and assigns the task MBean associated with the current task to a variable that you can use to check its status. If you are importing WLST as a Jython module, as described in Importing WLST as a Jython Module in Understanding the WebLogic Scripting Tool, block is always set to true.</td>
</tr>
</tbody>
</table>
Example
The following example activates the changes made during the current edit session that have been saved to disk, but that have not yet been activated. WLST waits for 100,000 ms for the activation to complete, and 200,000 ms before the activation is stopped.

```wls:/mydomain/edit !> activate(200000, block='true')
Activating all your changes, this may take a while ...
The edit lock associated with this edit session is released once the activation is completed.
Action completed.
wls:/mydomain/edit>
```

See Also
help, otd_createConfiguration, otd_listConfigurations, otd_deleteConfiguration, otd_copyConfiguration, otd_listConfigFiles, otd_getConfigFile, otd_saveConfigFile

### 2.2 deleteKeyStoreEntry

**Description**
Deletes a certificate or trusted certificate from the keystore using its alias. Refer to *Security Custom WLST Commands* for more information.

**Syntax**
```deleteKeyStoreEntry(appStripe='stripe', name='keystore', password='password', alias='alias', keypassword='keypassword')```  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>svc</code></td>
<td>Specifies the service command object obtained through a call to getOpssService().</td>
</tr>
<tr>
<td><code>appStripe</code></td>
<td>Specifies the name of the stripe where the keystore resides.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>Specifies the name of the keystore.</td>
</tr>
<tr>
<td><code>password</code></td>
<td>Specifies the keystore password.</td>
</tr>
<tr>
<td><code>alias</code></td>
<td>Specifies the alias of the entry to be deleted.</td>
</tr>
<tr>
<td><code>keypassword</code></td>
<td>Specifies the key password of the entry to be deleted.</td>
</tr>
</tbody>
</table>

**Example**
This example deletes a keystore entry denoted by alias *mycert*.

```java
svc = getOpssService("KeyStoreService")
svc.deleteKeyStoreEntry(appStripe='OTD', name='myconfig', password='', alias='mycert', keypassword='')
```
See Also
help, exportKeyStoreCertificateRequest, otd_listCertificates, importKeyStoreCertificate, getKeyStoreCertificates, generateKeyPair

2.3 displayLogs

Description
Use this command to view the contents of Oracle Traffic Director log files, the access log, tcp access log and error log. The access log records information about requests to and responses from the server.

The command returns a value only when the returnData option is set to true. By default it will not return any data. The return value depends on the option used.

Syntax

displayLogs([searchString,] [options])

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchString</td>
<td>An optional search string. Only messages that contain the given string (case-insensitive) will be returned. Note that the displayLogs command can read logs in multiple formats and it converts the messages to ODL format. The search will be performed in the native format, if possible. Otherwise, it may be performed in the message contents, and it may exclude mark-up. Therefore you should avoid using mark-up characters in the search string.</td>
</tr>
<tr>
<td>target</td>
<td>Optional. The name of a WebLogic Server instance, or a system component. For a system component, the syntax for the target is: sc:component-name In connected mode, the default target is the WebLogic domain. In disconnected mode, there is no default; the target option is required.</td>
</tr>
<tr>
<td>oracleInstance</td>
<td>Optional. Defines the path to the ORACLE_INSTANCE or WebLogic domain home. The command is executed in disconnected mode when you use this parameter.</td>
</tr>
<tr>
<td>log</td>
<td>Optional. A log file path. The command will read messages from the given log file. If the log file path is not given, the command will read all logs associated with the given target.</td>
</tr>
<tr>
<td>last</td>
<td>Optional. An integer value. Restricts the search to messages logged within the last minutes. The value can have a suffix s (second), m (minute), h (hour), or d (day) to specify a different time unit. (For example, last='2h' will be interpreted as the last 2 hours).</td>
</tr>
<tr>
<td>tail</td>
<td>Optional. An integer value. Restrict the search to the last n messages from each log file and limits the number of messages displayed to n.</td>
</tr>
<tr>
<td>Argument</td>
<td>Definition</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td>pattern</td>
<td>Optional. A regular expression pattern. Only messages that contain the given pattern are returned. Using the pattern option is similar to using the searchString argument, except that you can use a regular expression. The regular expression pattern search is case sensitive (unless you explicitly turn on case-insensitive flags in the pattern). The pattern must follow java.util.regex syntax.</td>
</tr>
<tr>
<td>ecid</td>
<td>Optional. A string or string sequence containing one or more Execution Context ID (ECID) values to be used as a filter for log messages.</td>
</tr>
<tr>
<td>component</td>
<td>Optional. A string or string sequence containing one or more component ID values to be used as a filter for log messages.</td>
</tr>
<tr>
<td>module</td>
<td>Optional. A string or string sequence containing one or more module ID values to be used as a filter for log messages.</td>
</tr>
<tr>
<td>type</td>
<td>Optional. A string or string sequence containing one or more message type values to be used as a filter for log messages.</td>
</tr>
<tr>
<td>app</td>
<td>Optional. A string or string sequence containing one or more application values to be used as a filter for log messages.</td>
</tr>
</tbody>
</table>
| query    | Optional. A string that specifies an expression used to filter the contents of log messages. A simple expression has the form: 

```plaintext
field-name operator value
```

where `field-name` is a log record field name and `operator` is an appropriate operator for the field type (for example, you can specify equals, startsWith, contains or matches for string fields).

A field name is either one of the standard ODL attribute names (such as COMPONENT_ID, MSG_TYPE, MSG_TEXT, and SUPPL_DETAIL), or the name of a supplemental attribute (application specific), prefixed by SUPPL_ATTR. (For example, SUPPL_ATTR.myAttribute).

A few common supplemental attributes can be used without the prefix. For example, you can use APP to filter by application name.

You can combine multiple simple expressions using the boolean operators and, or and not to create complex expressions, and you can use parenthesis for grouping expressions.

See Administering Oracle Fusion Middleware for a detailed description of the query syntax.

<p>| groupBy  | Optional. A string list. When the groupBy option is used, the output is a count of log messages, grouped by the attributes defined in the string list. |</p>
<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>orderBy</td>
<td>Optional. A string list that defines the sort order for the result. The values are log message attribute names. The name may be extended with an optional suffix :asc or :desc to specify ascending or descending sorting. The default sort order is ascending. By default, the result is sorted by time.</td>
</tr>
<tr>
<td>returnData</td>
<td>Optional. A Jython boolean value (0 or 1). If the value is true the command will return data (for example, to be used in a script). The default value is false, which means that the command only displays the data but does not return any data.</td>
</tr>
<tr>
<td>format</td>
<td>Optional. A string defined the output format. Valid values are ODL-Text, ODL-XML, ODL-complete and simple. The default format is ODL-Text.</td>
</tr>
<tr>
<td>exportFile</td>
<td>Optional. The name of a file to where the command output is written. By default, the output is written to standard output.</td>
</tr>
<tr>
<td>follow (f)</td>
<td>Optional. Puts the command in “follow” mode so that it continues to read the logs and display messages as new messages are added to the logs (similar to the UNIX tail -f command). The command will not return when the f option is used. This option is currently not supported with system components.</td>
</tr>
</tbody>
</table>

**Example**

displayLogs(target="sc:otd_test_varunam.in.example.com")

**See Also**


## 2.4 displayMetricTables

**Description**

This WLST command can be used to display runtime statistics about a server instance.

**Syntax**

displayMetricTables([metricTable_1], [metricTable_2], [...], [servers], [variables])
**displayMetricTables**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>metricTable_n</td>
<td>Optional. Specifies a list of metric tables. By default, this argument displays all available metrics. The metric table name can contain special characters for simple pattern matching. The character '?' matches any single character. The character '*' matches zero or more characters. You specify the metric table name. You can specify multiple metric table names in a comma-separated list. These are the same names output by the WLST command displayMetricTableNames.</td>
</tr>
<tr>
<td>servers</td>
<td>Optional. Specifies the servers from which to retrieve metrics. Valid values are a list of WebLogic Server instance names and system component names. To specify one server, use the following syntax: servers='servername' To specify multiple servers, use one of the following syntax options: servers=['servername1', 'servername2', ...] servers=(servername1, servername2, ...) If this argument is not specified, the command returns the list of metric tables for all WebLogic servers and system components. For system components, such as Oracle HTTP Server, use the following format: servers=['component_name', servertype='component_type'])</td>
</tr>
<tr>
<td>variables</td>
<td>Optional. Defines the metric aggregation parameters. Valid values are a set of name-value pairs. It uses the following syntax: variables={name1:value1, name2:value2, ...} The specific name-value pairs depend on the aggregated metric tables. Each aggregated metric table has its specific set of variable names.</td>
</tr>
</tbody>
</table>

**Example**

Note that at least a single Oracle Traffic Director instance needs to be running for the following examples to work correctly.

```bash
# View metrics for all OTD instances
displayMetricTables('OTD_*')

# View origin server metrics for all instances
displayMetricTables('OTD_OriginServer')

# Get list of metric tables for a specific instance
displayMetricTableNames(servers='/OTD/otd_test_myserver.example.com')

# View all metrics for a specific instance
displayMetricTables(servers='/OTD/otd_test_myserver.example.com')
```
# View instance metrics for a specific instance
displayMetricTables('OTD_Instance', servers='/OTD/otd_test_myserver.example.com')

See Also
help, otd_createOriginServer, otd_deleteOriginServer, otd_listOriginServers, otd_setOriginServerProperties

2.5 enableOverwriteComponentChanges

Description
Executing this command before activate lets the activate call overwrite the local configuration file modifications on instances with their corresponding server versions.

An activate call would fail if there are any local configuration file modifications on the instance. In such a case, you would want to either discard the changes on the instance or pull the changes from the instance to the config store by executing pullComponentChanges. In either case, you should execute the command enableOverwriteComponentChanges before activate such that the activate call would not fail because of the local modifications on the instance.

Note:
This command can only be executed from an open edit session. See resync/resyncAll for overriding instance changes outside of an open edit session.

Syntax
enableOverwriteComponentChanges()

Example
props={'configuration': 'test', 'name': 'var_foo', 'value': 'bar'}
otd_createVariable(props)
activate()

weblogic.management.provider.UpdateException: [Management:141191]The prepare phase of the configuration update failed with an exception.
Caused by: weblogic.nodemanager.NMException: Received error message from Node Manager Server:
[ChangeList validation failed for transaction '3033897627106602' with cause:
OTD-67807 Validation failed for instance 'otd_test.example.com':
The instance configuration has been locally modified. The following changes can either be discarded on the next activate using 'enableOverwriteComponentChanges' or pulled into the current configuration using 'pullComponentChanges'.
Modified files: config/server.xml,config/test-obj.conf,config/obj.conf

# Scenario 1: Pull the changes on instance to config store and call enableOverwriteComponentChanges and activate.

showComponentChanges("otd_test.example.com")
component otd_test.example.com changes on machine example.com:
edit OTD/test/config/obj.conf 2014.12.01-16:20:50 1970.01.01-05:29:59
edit OTD/test/config/test-obj.conf 2014.12.01-16:20:50 1970.01.01-05:29:59
pullComponentChanges("otd_test.example.com")
pull component otd_test.example.com changes on machine in.example.com:
edit OTD/test/config/obj.conf
edit OTD/test/config/test-obj.conf
edit OTD/test/config/server.xml
enableOverwriteComponentChanges()
activate()
Activating all your changes, this may take a while ...
The edit lock associated with this edit session is released once the activation is completed.
Activation completed

# Scenario 2: Discard the changes on the instance and override them with changes from the current edit session

showComponentChanges("otd_test.example.com")
component otd_test.example.com changes on machine example.com:
edit OTD/test/config/obj.conf 2014.12.01-16:55:29 1970.01.01-05:29:59
enableOverwriteComponentChanges()
activate()
Activating all your changes, this may take a while ...
The edit lock associated with this edit session is released once the activation is completed.
Activation completed

See Also
help, pullComponentChanges, resync/resyncAll, showComponentChanges, stopEdit, undo

2.6 exportKeyStoreCertificate

Description
Exports a certificate, trusted certificate or certificate chain. Refer to Security Custom WLST Commands for more information.

Syntax
exportKeyStoreCertificate(appStripe='stripe', name='keystore', password='password', alias='alias', keypassword='keypassword', type='entrytype', filepath='absolute_file_path')

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>svc</strong></td>
<td>Specifies the service command object obtained through a call to getOpssService().</td>
</tr>
<tr>
<td><strong>appStripe</strong></td>
<td>Specifies the name of the stripe where the keystore resides.</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>Specifies the name of the keystore.</td>
</tr>
</tbody>
</table>
### Argument Definition

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>Specifies the keystore password.</td>
</tr>
<tr>
<td>alias</td>
<td>Specifies the alias of the entry to be exported.</td>
</tr>
<tr>
<td>keypassword</td>
<td>Specifies the key password.</td>
</tr>
<tr>
<td>type</td>
<td>Specifies the type of keystore entry to be exported. Valid values are 'Certificate', 'TrustedCertificate' or 'CertificateChain'.</td>
</tr>
<tr>
<td>filepath</td>
<td>Specifies the absolute path of the file where certificate, trusted certificate or certificate chain is exported.</td>
</tr>
</tbody>
</table>

#### Example

```java
svc = getOpssService("KeyStoreService")
svc.exportKeyStoreCertificate(appStripe='OTD', name='myconfig', password='',
alias='mycert', keypassword='', type='Certificate', filepath='/scratch/cert.txt')
```

#### See Also

help, importKeyStoreCertificate, otd_listCertificates, deleteKeyStoreEntry, getKeyStoreCertificates

### 2.7 exportKeyStoreCertificateRequest

#### Description

Generate a certificate signing request for a key pair and saves it to a file. Refer to Security Custom WLST Commands for more information. This Base64-encoded certificate request can be submitted to a third-party Certificate Authority (CA) which will verify the sender, sign and return the signed certificate.

#### Syntax

```java
exportKeyStoreCertificateRequest(appStripe='stripe', name='keystore',
password='password', alias='alias', keypassword='keypassword',
filepath='absolute_file_path')
```

#### Argument Definition

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>svc</td>
<td>Specifies the service command object obtained through a call to getOpssService().</td>
</tr>
<tr>
<td>appStripe</td>
<td>Specifies the name of the stripe where the keystore resides.</td>
</tr>
<tr>
<td>name</td>
<td>Specifies the name of the keystore.</td>
</tr>
</tbody>
</table>
### Argument Definition

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>password</code></td>
<td>Specifies the keystore password.</td>
</tr>
<tr>
<td><code>alias</code></td>
<td>Specifies the entry’s alias name.</td>
</tr>
<tr>
<td><code>keypassword</code></td>
<td>Specifies the key password.</td>
</tr>
<tr>
<td><code>filepath</code></td>
<td>Specifies the absolute path of the file where certificate request is exported.</td>
</tr>
</tbody>
</table>

### Example

```python
svc = getOpssService("KeyStoreService")

# generate a key pair with the proper DN
svc.generateKeyPair(appStripe='OTD', name='myconfig', password='©©', alias='mycert',
keypassword='©©', dn='CN=test.example.com, OU=Webtier, O=\'Company Name\',
ST=California, C=US', keysize='1024')

# generate the CSR and put it in to a text file
svc.exportKeyStoreCertificateRequest(appStripe='OTD', name='myconfig', password='©©',
alias='mycert', keypassword='©©', filepath='/scratch/certreq.crt')
```

### See Also

- `help`
- `importKeyStoreCertificate`
- `otd_listCertificates`
- `deleteKeyStoreEntry`
- `getKeyStoreCertificates`

## 2.8 generateKeyPair

### Description

Use this command to generate a key pair in a keystore and wrap it in a demo CA-signed certificate. Refer to Security Custom WLST Commands for more information. This command is the equivalent of creating a self-signed certificate in Release 11g. You can use this key pair to generate a certificate signing request (CSR) using `exportKeyStoreCertificateRequest` which you can submit to a third-party Certificate Authority (CA) for signing.

### Syntax

```python
generateKeyPair(appStripe='stripe', name='keystore', password='password',
dn='distinguishedname', keysize='keysize', alias='alias',
keypassword='keypassword')
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
</table>
| `svc` | Specifies the service command object obtained through a call to `getOpssService()`.

---

The generated text is a detailed documentation of `generateKeyPair` command, including its arguments, definitions, and examples.
**appStripe**
Specifies the name of the stripe where the keystore resides.

**name**
Specifies the name of the keystore.

**password**
Specifies the keystore password.

**dn**
Specifies the distinguished name of the certificate wrapping the key pair.

**keysize**
Specifies the key size.

**alias**
Specifies the alias of the key pair entry.

**keypassword**
Specifies the key password.

### Example

```
svc = getOpssService("KeyStoreService")
svc.generateKeyPair(appStripe='OTD', name='myconfig', password='', alias='mycert',
keypassword='', dn='CN=test.example.com, OU=Webtier, O=\"Company Name\", ST=California, C=US',
keysize='1024')
```

### See Also

help, importKeyStoreCertificate, otd_listCertificates, deleteKeyStoreEntry,
getKeyStoreCertificates, exportKeyStoreCertificateRequest

## 2.9 getKeyStoreCertificates

### Description

Use this command to view the certificate properties including subject, issuer, issue date, and expiry date. Refer to *Security Custom WLST Commands* for more information.

### Syntax

```
getKeyStoreCertificates(appStripe='stripe', name='keystore',
password='password', alias='alias', keypassword='keypassword')
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>svc</strong></td>
<td>Specifies the service command object obtained through a call to getOpssService().</td>
</tr>
</tbody>
</table>
### Argument Definition

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>appStripe</td>
<td>Specifies the name of the stripe where the keystore resides.</td>
</tr>
<tr>
<td>name</td>
<td>Specifies the name of the keystore.</td>
</tr>
<tr>
<td>password</td>
<td>Specifies the keystore password.</td>
</tr>
<tr>
<td>alias</td>
<td>Specifies the alias of the certificate, trusted certificate or certificate chain to be displayed.</td>
</tr>
<tr>
<td>keypassword</td>
<td>Specifies the key password.</td>
</tr>
</tbody>
</table>

#### Example

```python
csvc = getOpssService("KeyStoreService")
csvc.getKeyStoreCertificates(appStripe='OTD', name='myconfig', password=''
alias='mycert')
```

#### See Also

help, importKeyStoreCertificate, deleteKeyStoreEntry, otd_listCertificates, generateKeyPair, exportKeyStoreCertificateRequest

### 2.10 help

#### Description

Lists all available Oracle Traffic Director custom WLST commands, or lists help for a particular command.

#### Syntax

To list all available Oracle Traffic Director custom WLST commands:

```python
help('otd')
```

To list help for a particular command:

```python
help('<otd_custom_command>')
```

#### Example

```python
help('otd_createConfiguration')
```

### 2.11 importKeyStoreCertificate

#### Description

Imports a CA signed or trusted certificate into the keystore. Refer to Security Custom WLST Commands for more information.
Once a CSR is submitted to a CA for signing, the CA signs the request and typically sends the certificate as a Base-64 encoded string. You should import this certificate as type CertificateChain along with any Intermediate and Root CA certificates using the same alias as that of the key pair that was used to generate the certificate request.

Once you have downloaded your certificate from your CA, you can download any Intermediate and Root certificates from your CA’s website, open a text editor and paste the entire body of each certificate into one text file in the following order: Primary Certificate > Intermediate Certificate > Root Certificate.

The file should appear as follows when finished:

```
-----BEGIN CERTIFICATE-----
(Server SSL certificate)
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
(Intermediate certificate)
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
(Root certificate)
-----END CERTIFICATE-----
```

**Syntax**

```python
importKeyStoreCertificate(appStripe='stripe', name='keystore', password='password', alias='alias', keypassword='keypassword', type='entrytype', filepath='absolute_file_path')
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>svc</code></td>
<td>Specifies the service command object obtained through a call to getOpssService().</td>
</tr>
<tr>
<td><code>appStripe</code></td>
<td>Specifies the name of the stripe where the keystore resides.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>Specifies the name of the keystore.</td>
</tr>
<tr>
<td><code>password</code></td>
<td>Specifies the keystore password.</td>
</tr>
<tr>
<td><code>alias</code></td>
<td>Specifies the alias of the entry to be imported.</td>
</tr>
<tr>
<td><code>keypassword</code></td>
<td>Specifies the key password of the newly imported entry.</td>
</tr>
<tr>
<td><code>type</code></td>
<td>Specifies the type of keystore entry to be imported. Valid values are 'Certificate', 'TrustedCertificate' or 'CertificateChain'.</td>
</tr>
<tr>
<td><code>filepath</code></td>
<td>Specifies the absolute path of the file from where certificate, trusted certificate or certificate chain is imported.</td>
</tr>
</tbody>
</table>
Example

svc = getOpssService("KeyStoreService")

# generate a key pair with the proper DN
svc.generateKeyPair(appStripe='OTD', name='myconfig', password='', alias='mycert',
keypassword='', dn='CN=test.example.com, OU=Webtier, O=\'Company Name\',
ST=California, C=US', keysize='1024')

# generate the CSR and put it in a text file
svc.exportKeyStoreCertificateRequest(appStripe='OTD', name='myconfig', password='',
alias='mycert', keypassword='', filepath='/scratch/certreq.crt')

# Submit the CSR to a CA who can sign the certificate and import signed cert into
# the keystore using the same alias as the key pair. Note that the file being imported
# should contain the CA cert along with the server cert and should be
# imported as type 'CertificateChain'
svc.importKeyStoreCertificate(appStripe='OTD', name='myconfig', password='',
alias='mycert', keypassword='', type='CertificateChain', filepath='/scratch/
certsign.pem')

# Any CA cert can be imported into the keystore as a trusted cert
svc.importKeyStoreCertificate(appStripe='OTD', name='myconfig', password='',
alias='ca-cert', keypassword='', type='TrustedCertificate', filepath='/scratch/cacert.crt')

See Also
help, exportKeyStoreCertificateRequest, otd_listCertificates, deleteKeyStoreEntry,
getKeyStoreCertificates, generateKeyPair

2.12 listExpiringCertificates

Description
List certificates expiring in a specified period. Refer to Security Custom WLST Commands for more information.

Syntax
listExpiringCertificates(days='days', autorenew=true|false)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>svc</td>
<td>Specifies the service command object obtained through a call to getOpssService().</td>
</tr>
<tr>
<td>days</td>
<td>Specifies that the list should only include certificates within this many days from expiration.</td>
</tr>
<tr>
<td>autorenew</td>
<td>Specifies true for automatically renewing expiring certificates, false for only listing them.</td>
</tr>
</tbody>
</table>

Example
svc = getOpssService("KeyStoreService")
svc.listExpiringCertificates(days='365', autorenew=false)
2.13 listKeyStores

Description
List all the keystores in a stripe. Refer to Security Custom WLST Commands for more information. In the case of Oracle Traffic Director, a permission-protected keystore is created at the same time as the configuration and also has the same name as the configuration. Hence the keystore names returned by listKeyStores will typically match the configuration names.

Syntax
listKeyStores(appStripe='stripe')

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>svc</td>
<td>Specifies the service command object obtained through a call to getOpssService().</td>
</tr>
<tr>
<td>appStripe</td>
<td>Specifies the name of the stripe whose keystores are listed.</td>
</tr>
</tbody>
</table>

Example
svc = getOpssService("KeyStoreService")
svc.listKeyStores(appStripe='OTD')

See Also
help, importKeyStoreCertificate, otd_listCertificates, deleteKeyStoreEntry, getKeyStoreCertificates, exportKeyStoreCertificateRequest

2.14 listKeyStoreAliases

Description
List aliases in a keystore. Refer to Security Custom WLST Commands for more information. Any certificate that is generated or imported into the keystore will be listed by its alias.

Syntax
listKeyStoreAliases(appStripe='stripe', name='keystore', password='password', type='entrytype')

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>svc</td>
<td>Specifies the service command object obtained through a call to getOpssService().</td>
</tr>
</tbody>
</table>

Oracle Traffic Director WLST Commands 2-15
## Argument Definition

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>appStripe</strong></td>
<td>Specifies the name of the stripe where the keystore resides.</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>Specifies the name of the keystore.</td>
</tr>
<tr>
<td><strong>password</strong></td>
<td>Specifies the keystore password.</td>
</tr>
<tr>
<td><strong>type</strong></td>
<td>Specifies the type of entry for which aliases are listed. Valid values are 'Certificate', 'TrustedCertificate', 'SecretKey' or '*'.</td>
</tr>
</tbody>
</table>

## Example

```python
tsvc = getOpssService("KeyStoreService")

# List all certificates
tsvc.listKeyStoreAliases(appStripe='OTD', name='myconfig', password='', type='*')

# List all user certificates (both SSL server and client)
tsvc.listKeyStoreAliases(appStripe='OTD', name='myconfig', password='', type='Certificate')

# List only Trusted CA certificates
tsvc.listKeyStoreAliases(appStripe='OTD', name='myconfig', password='', type='TrustedCertificate')
```

## See Also

help, importKeyStoreCertificate, otd_listCertificates, deleteKeyStoreEntry, getKeyStoreCertificates, exportKeyStoreCertificateRequest

## 2.15 otd_addFailoverInstance

### Description

Use this command to add a failover instance. This command is valid only for active-active failover type. You can add a maximum of 254 instances in a failover group.

### Syntax

```python
otd_addFailoverInstance(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-ip</td>
<td>Virtual IP that uniquely identifies the failure group.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Comments</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>instance</td>
<td>Name of the instance.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>nic</td>
<td>A network interface, upon which the VIP must be managed.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example 2-1  Example**

```python
props = {
    'configuration': 'ha',
    'virtual-ip': '10.128.67.44',
    'instance': '1.example.com',
    'nic': 'eth0'
}

otd_addFailoverInstance(props)
```

**See Also**

2.16 otd_blockProxyInfo

**Description**

Use this command to block the generation and forwarding of a particular proxy parameter to the origin server. The information about the proxy parameters and headers is described in otd_forwardProxyInfo.

**Note:**

If the incoming request contains any of the headers corresponding to the proxy parameters, Oracle Traffic Director will pass-through the incoming request containing this header to the origin server.

**Syntax**

```python
otd_blockProxyInfo(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>route</td>
<td>Name of the route.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
### otd_blockProxyInfo

#### Description
Name of the proxy parameter to be blocked.

Range of values: jroute, via, ip, cipher, proxy-agent, keysize, secret-keysize, ssl-id, issuer-dn, user-dn, auth-cert, xforwarded-for, cache-info, ssl.

#### Comments
Mandatory.

#### Example
```python
def otd_blockProxyInfo(props):
    pass
```

#### See Also
- help, otd_listProxyInfo, otd_forwardProxyInfo

### 2.17 otd_copyConfiguration

#### Description
Use this command to create a copy of an existing configuration.

#### Syntax
```python
don't_copyConfiguration(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>source-configuration</td>
<td>Name of the configuration to be copied.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>dest-configuration</td>
<td>Name of the new configuration. Name should not contain spaces, invalid characters or non-ASCII characters.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

#### Example
```python
def otd_copyConfiguration(props):
    pass
```

#### See Also
- help, otd_createConfiguration, otd_deleteConfiguration, otd_listConfigurations, activate
2.18 otd_copyVirtualServer

Description
Use this command to create a copy of an existing virtual server.

Syntax
otd_copyVirtualServer(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>source-virtual-server</td>
<td>Name of the virtual server to be copied.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>dest-virtual-server</td>
<td>Name of the new virtual server.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

Example
props = {}  
props['configuration'] = 'foo'  
props['source-virtual-server'] = 'bar'  
props['dest-virtual-server'] = 'baz'  
otd_copyVirtualServer(props)

See Also
help, otd_createVirtualServer, otd_setVirtualServerProperties, otd_deleteVirtualServer, otd_getVirtualServerProperties, otd_listVirtualServers

2.19 otd_createCacheRule

Description
Use this command to create a cache rule with a set of initial values.

Syntax
otd_createCacheRule(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
## otd_createCompressionRule

### Description
Use this command to create a compression rule with an initial set of values.

### Syntax
```
otd_createCompressionRule(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>compression-rule</td>
<td>Name of the compression rule to be created.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>condition</td>
<td>A condition is an expression which if evaluates to true, will result in the rule being executed. Conditions are constructed from literals, variables, functions and operators.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

### Example
```
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['compression-rule'] = 'cache-rule-1'
otd_createCacheRule(props)
```

### See Also
`help, otd_deleteCompressionRule, otd_getCacheProperties, otd_getCacheRuleProperties, otd_listCacheRules`
Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['compression-rule'] = 'compression-rule-1'
otd_createCompressionRule(props)
```

See Also

`help`, `otd_deleteCompressionRule`, `otd_setCompressionRuleProperties`, `otd_listCompressionRules`

### 2.21 `otd_createConfiguration`

**Description**

Use this command to create a new configuration that listens to HTTP and TCP traffic on a given port and front-ends a set of HTTP and TCP origin servers. The command creates a default virtual server that handles HTTP traffic and a default TCP proxy that handles TCP traffic. In addition, it creates a default route and forwards all traffic to the origin server.

**Syntax**

```
otd_createConfiguration(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the configuration to be created. Name should not contain spaces, invalid characters or non-ASCII characters.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>listener-port</td>
<td>Listener port through which the server accepts requests.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>server-name</td>
<td>Valid only if <code>origin-server-type</code> is <code>http</code> or <code>https</code>. The server name is used in any URLs that are generated automatically by the server and sent to the client. This server name should be the virtual host name or alias name if your server uses an alias. If a colon and port number are appended to the server name then that port is used in the generated URLs.</td>
<td></td>
</tr>
<tr>
<td>ip</td>
<td>The server will bind to this Internet Protocol (IP) address for the default listener. Only traffic sent to this IP address will be serviced. * indicates that the server will listen on all IP addresses.</td>
<td></td>
</tr>
<tr>
<td>origin-server-type</td>
<td>Type of requests handled by the origin servers.</td>
<td></td>
</tr>
</tbody>
</table>

Range of values: 1 and 65535, both inclusive.

Range of values: *, a hostname, or an IPV4/IPV6 address

Range of values: http/https/tcp

Default: http
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>origin-server</td>
<td>A back-end server to which Oracle Traffic Director forwards requests that it receives from clients, and from which it receives responses to client requests. The origin servers could, for example, be application servers like Oracle WebLogic Server, web servers, LDAP servers, and so on. This should be specified as a comma separated list of origin servers of the format host:port.</td>
<td>Multi-valued.</td>
</tr>
</tbody>
</table>

**Note:**

You cannot invoke this command in offline mode until you have read a domain using `readDomain`. Make sure to update the domain using `updateDomain` after the command to apply the changes.

**Example**

**Online:**

```bash
# Online
props = {}
props['name'] = 'foo'
props['listener-port'] = '12345'
props['server-name'] = 'foo'
props['origin-server'] = 'vault.example.com:80'
otd_createConfiguration(props)
```

**Offline:**

```bash
# Offline
readDomain('/export/domains/otd_domain')
props = {}
props['name'] = 'foo'
props['listener-port'] = '12345'
props['server-name'] = 'foo'
props['origin-server'] = 'vault.example.com:80'
otd_createConfiguration(props)
updateDomain()
closeDomain()
```

**See Also**

`help`, `otd_listConfigurations`, `otd_deleteConfiguration`, `otd_copyConfiguration`, `otd_listConfigFiles`, `otd_getConfigFile`, `otd_saveConfigFile`, `activate`

### 2.22 otd_createContentRule

**Description**

Use this command to create a content rule.
Syntax

```python
otd_createContentRule(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server for which the content rule is to be created.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>content-rule</td>
<td>Name of the content rule to be created.</td>
<td>Mandatory. Name should be unique.</td>
</tr>
<tr>
<td>uri-prefix</td>
<td>URI prefix that has to be mapped to a directory.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>directory-path</td>
<td>Absolute server path and a valid directory for storing documents.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['uri-prefix'] = '/baz'
props['directory-path'] = '/qux'
props['content-rule'] = 'content-rule-1'
otd_createContentRule(props)
```

See Also

```
help, otd_getContentRuleProperties, otd_listContentRules, otd_deleteContentRule, otd_setContentRuleProperties
```

**2.23 otd_createErrorPage**

Description
Use this command to create an error page corresponding to the specified error code.

Syntax

```python
otd_createErrorPage(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
### Property Description Comments

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>code</td>
<td>Error code for which you want to create an error page.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>error-page</td>
<td>Absolute path for which an error page is to be created.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

#### Example

```python
props = {
    'configuration': 'foo',
    'virtual-server': 'bar',
    'code': '408',
    'error-page': '/documents/otd'
}
```  

#### See Also

`help, otd_deleteErrorPage, otd_listErrorPages`

### 2.24 otd_createEvent

#### Description

Use this command to create an event.

#### Syntax

```python
otd_createEvent(props)
```  

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration for which the event is to be created.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>event</td>
<td>Name that uniquely identifies the event. Name can consist of one or more characters. Whitespace is not permitted.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>command</td>
<td>The command that the event executes. Range of values: the value can be restart, reconfig, rotate-log, rotate-access-log, and update-crl, or any executable command.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>time</td>
<td>Time, for example, 12:30, when this event is to be started. Range of values: the format of the time is <code>hh:mm</code>.</td>
<td></td>
</tr>
</tbody>
</table>
### Property Description Comments

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>month</td>
<td>Month at which this event should occur. Range of values: 1-12.</td>
<td></td>
</tr>
<tr>
<td>day-of-month</td>
<td>Day of the month at which this event should occur. Range of values: 1-31.</td>
<td></td>
</tr>
<tr>
<td>day-of-week</td>
<td>Day of the week at which this event should occur. Range of values: Sun, Mon, Tue, Wed, Thu, Fri, or Sat.</td>
<td></td>
</tr>
<tr>
<td>interval</td>
<td>Time interval at which this event should occur. Range of values: an interval in seconds between 60 (1 minute) and 86400 (1 day), inclusive.</td>
<td></td>
</tr>
<tr>
<td>enabled</td>
<td>Whether the event is enabled at runtime. Range of values: true or false.</td>
<td>Default: true.</td>
</tr>
</tbody>
</table>

### Example

```python
props = {}
props['configuration'] = 'foo'
props['event'] = 'event-1'
props['command'] = 'rotate-log'
props['time'] = '12:00'
otd_createEvent(props)
```

### See Also

help, otd_deleteEvent, otd_listEvents, otd_getEventProperties, otd_setEventProperties

### 2.25 otd_createFailoverGroup

#### Description

Use this command to create a failover group consisting of two instances grouped by a unique virtual IP address (VIP), to provide high availability in active-passive mode. Requests are received at the specified VIP address and routed to the instance that is designated as the primary instance. If the primary instance is not reachable, requests are routed to the backup instance.

For active-active failover, you should create two failover groups, both consisting of the same two nodes, but with the primary and backup roles reversed. One of the nodes will be master node, and the rest of the nodes are backup nodes. The incoming requests to VIP are distributed among the OTD instances. If the master node fails, the backup node having the highest priority is chosen as the next master node. After creating the failover group, add the failover instance using the otd_addFailoverInstance command.

There can be a maximum of 255 failover groups, across configurations.

When creating a failover group, if the administration node process is running as non-root on the node where the instances are located, then you must run
**otd_startFailover** on those nodes as a **root** user. This is to manually start the failover. If this command is not executed, failover will not start and there will be no high availability.

For more information about how failover works in Oracle Traffic Director, see *Administering Oracle Traffic Director*.

**Syntax**

```python
otd_createFailoverGroup(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration for which the failover group is to be created.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-ip</td>
<td>The VIP for which we are creating a failover for. The VIP should belong to</td>
<td>Mandatory.</td>
</tr>
<tr>
<td></td>
<td>the same subnet as that of the nodes in the failover group, and must be</td>
<td></td>
</tr>
<tr>
<td></td>
<td>accessible to clients. Range of values: host name or an IPv4/IPv6 address.</td>
<td></td>
</tr>
<tr>
<td>primary-instance</td>
<td>An existing instance which is designated as the primary.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>backup-instance</td>
<td>An existing instance which is designated as the backup.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>primary-nic</td>
<td>A network interface, on the node where primary-instance is running, upon</td>
<td>Mandatory.</td>
</tr>
<tr>
<td></td>
<td>which the VIP must be managed.</td>
<td></td>
</tr>
<tr>
<td>backup-nic</td>
<td>A network interface, on the node where backup-instance is running, upon</td>
<td>Mandatory.</td>
</tr>
<tr>
<td></td>
<td>which the VIP must be managed.</td>
<td></td>
</tr>
<tr>
<td>router-id</td>
<td>A VRRP necessity, identifies the VRRP router group that are participating</td>
<td></td>
</tr>
<tr>
<td></td>
<td>in failover for a VIP. The value should be unique across failover groups. If</td>
<td></td>
</tr>
<tr>
<td></td>
<td>not specified, the default router-id will be a random number between 1 -</td>
<td></td>
</tr>
<tr>
<td></td>
<td>255. Range of values: positive integer, valid range is 1-255. Default:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>random number between 1 - 255.</td>
<td></td>
</tr>
<tr>
<td>failover type</td>
<td>Failover group type. Values: Active-passive, Active-active. Default value:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Active-passive. Note: This property is valid on a Linux platform only.</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

Creating an active-passive failover group

```python
props = {}
props['configuration'] = 'ha'
props['virtual-ip'] = '192.0.2.1'
props['primary-instance'] = '1.example.com'
```
Creating an active-active failover group

```python
props = {
    'configuration': 'ha',
    'virtual-ip': '192.0.2.1',
    'failover-type': 'active-active'
}
otd_createFailoverGroup(props)
```

See Also
help, otd_deleteFailoverGroup, otd_getFailoverGroupProperties, otd_toggleFailoverGroupPrimary, otd_startFailover, otd_stopFailover

### 2.26 otd_createHttpListener

**Description**

Use this command to create a new HTTP listener socket with a set of initial values. All virtual servers have an HTTP listener specified. When a new request comes in, determines which virtual server to send it to, based on the configured HTTP listener.

**Syntax**

```python
otd_createHttpListener(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>http-listener</td>
<td>Name that uniquely identifies the HTTP listener. Name can consist of one or more characters. Whitespace is not permitted.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>port</td>
<td>Port on which to listen.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>server-name</td>
<td>Default server name. May include a scheme (for example, http://) prefix and port (for example, :80) suffix. Can be a hostname, fully qualified domain name, IP address, or a URL prefix that contains one. The URL prefix must not specify a path.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>default-virtual-server-name</td>
<td>Name of the virtual server that processes requests that did not match a host.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Comments</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>enabled</td>
<td>Whether the listener is enabled at runtime.</td>
<td>Range of values: true or false. Default: true.</td>
</tr>
<tr>
<td>ip</td>
<td>IP address on which to listen.</td>
<td>Range of values: *, a hostname, or an IP address.</td>
</tr>
<tr>
<td>acceptor-threads</td>
<td>Number of threads dedicated to accepting connections received by this listener.</td>
<td>Range of values: 1 - 128.     Default: auto-tuned.</td>
</tr>
<tr>
<td>blocking-io</td>
<td>Whether the server uses blocking IO.</td>
<td>Range of values: true or false. Default: false.</td>
</tr>
<tr>
<td>blocking-accept</td>
<td>Enables/Disables blocking of the server Listen Socket while retaining client end points as non blocking (useful when MaxProcs &gt; 1).</td>
<td>Range of values: true or false. Default: false.</td>
</tr>
<tr>
<td>handle-protocol-mismatch</td>
<td></td>
<td>Range of values: true or false. Default: true.</td>
</tr>
<tr>
<td>family</td>
<td>The socket family used to connect to the origin server.</td>
<td>Range of values: default, inet, inet6, or inet-sdp Default: auto-tuned.</td>
</tr>
<tr>
<td>receive-buffer-size</td>
<td>Size (in bytes) of the operating system socket receive buffer.</td>
<td>Range of values: size in bytes between 0 and 2147483647, inclusive.</td>
</tr>
<tr>
<td>send-buffer-size</td>
<td>Size (in bytes) of the operating system socket send buffer.</td>
<td>Range of values: size in bytes between 0 and 2147483647, inclusive.</td>
</tr>
<tr>
<td>max-requests-per-connection</td>
<td>Maximum number of keep-alive requests that will be handled per HTTP connection after which the keep-alive connection will be closed. 0 indicates no limit.</td>
<td>Range of values: any positive Integer Default: 0.</td>
</tr>
</tbody>
</table>
### Example

```python
def createHttpListener(props):
    # Create an HTTP listener for the administrator's reference.
    # Example props:
    props = {}
    props['configuration'] = 'foo'
    props['http-listener'] = 'http-listener-1'
    props['port'] = '23456'
    props['server-name'] = 'example.com'
    props['default-virtual-server-name'] = 'bar'
    createHttpListener(props)
```

### See Also

- help, otd_setHttpListenerProperties, otd_listHttpListeners, otd_deleteHttpListener

### 2.27 otd_createInstance

#### Description

Use this command to create an instance of this configuration on the specified machine. Instance refers to the environment of a daemon, including its configuration, log files, and other runtime artifacts such as lock databases, caches, and temporary files.

#### Syntax

```python
otd_createInstance(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>machine-name</td>
<td>Name specified while creating the machine in the Oracle WebLogic Server console corresponding to the host name of the machine on which the Oracle Traffic Director instance is running.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

#### Note:

When this command is executed in offline mode, the instance file artifacts are created only if the machine specified is on the same host as that of the admin server. Otherwise, the instance file artifacts will get created after the start of both admin server and node manager.
Example

# Online
props = {}
props['configuration'] = 'foo'
props['machine-name'] = 'machine1'
otd_createInstance(props)

# Offline
readDomain('/export/domains/otd_domain')
props = {}
props['configuration'] = 'foo'
props['machine-name'] = 'machine1'
otd_createInstance(props)
updateDomain()
closeDomain()

See Also
help, otd_deleteInstance, otd_listInstances, start, stop, softRestart

2.28 otd_createMimeType

Description
Use this command to create a MIME type.

Syntax
otd_createMimeType(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>content-type</td>
<td>The content type of the MIME types.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>extensions</td>
<td>The file extension for the MIME value.</td>
<td>Mandatory. To define multiple file extensions, separate them by a comma (,)</td>
</tr>
</tbody>
</table>

Example

props = {}
props['configuration'] = 'foo'
props['content-type'] = 'bar'
props['extensions'] = 'baz'
otd_createMimeType(props)
2.29 otd_createOriginServer

Description

Use this command to create a origin pool server with a set of initial values to the existing origin server pool. The origin server defines a member of a server pool.

Syntax

otd_createOriginServer(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>host</td>
<td>IP address/Host name of this origin server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>port</td>
<td>Port number of this origin server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>weight</td>
<td>Load distribution weight for this origin server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>backup</td>
<td>The parameter specifies if the origin server is a backup server.</td>
<td></td>
</tr>
<tr>
<td>max-connections</td>
<td>The maximum number of concurrent connections to a server.</td>
<td></td>
</tr>
<tr>
<td>ramp-up-time</td>
<td>The time in seconds to ramp the sending rate up to the capacity of a newly up origin server. If the parameter is not specified, request rate accelerating will not be activated for the server.</td>
<td></td>
</tr>
</tbody>
</table>

Range of values: hostname or IP address.
Range of values: port number between 1 and 65535, inclusive.
Range of values: 1 - 1000.
Default: 1
Range of values: true or false.
Default: false.
Range of values: 0 - 20480.
Default: 0.
Range of values: an interval in seconds between 0.001 and 3600 (1 hour), inclusive.
Default: 0.001.
Example

```python
props = {}
props['configuration'] = 'foo'
props['origin-server-pool'] = 'origin-server-pool-1'
props['host'] = 'www.example.com'
props['port'] = '12345'
otd_createOriginServer(props)
```

See Also

`help`, `otd_deleteOriginServer`, `otd_listOriginServers`, `otd_getOriginServerProperties`, `otd_setOriginServerProperties`

2.30 `otd_createOriginServerPool`

Description

Use this command to create an origin-server pool. The origin-server pool configures a pool of origin servers that are used for load balancing requests.

Syntax

```python
otd_createOriginServerPool(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name by which this server pool is referenced. Name can consist of one or more characters, whitespace is not permitted.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server</td>
<td>Represents an origin server that belongs to this server pool. Multiple comma separated values can be specified.</td>
<td>Multi-valued.</td>
</tr>
<tr>
<td>type</td>
<td>Specifies the type of (requests handled by) every server in this server pool. Range of values: tcp, http, or https. Default: http.</td>
<td></td>
</tr>
<tr>
<td>family</td>
<td>The socket family used to connect to servers in this pool. Range of values: default, inet, inet6, or inet-sdp. Default: auto-tuned.</td>
<td></td>
</tr>
<tr>
<td>load-distribution</td>
<td>Algorithm that is used for load distribution of this server pool. Range of values: round-robin, least-connection-count, or least-response-time. Default: least-connection-count.</td>
<td></td>
</tr>
</tbody>
</table>
**Property** | **Description** | **Comments**
--- | --- | ---
proxy-server | Name of the proxy-server in the form of host:port. |

**Example**

```python
global = {}
global['configuration'] = 'foo'
global['origin-server-pool'] = 'origin-server-pool-1'
global['origin-server'] = 'www.example.com:12345'
otd_createOriginServerPool(global)
```

**See Also**


---

### 2.31 otd_createRequestLimit

**Description**

Use this command to create a request limit rule with a set of initial values.

**Syntax**

```python
otd_createRequestLimit(props)
```

The argument `props` is a dictionary that can contain the following properties:

**Property** | **Description** | **Comments**
--- | --- | ---
configuration | Name of the configuration. | Mandatory. |
virtual-server | Name of the virtual server. | Mandatory. |
request-limit | Name of the request limit rule. | Mandatory. |
condition | A condition is an expression which if evaluates to true, will result in the rule being executed. Conditions are constructed from literals, variables, functions and operators. |
max-rps | Maximum number of requests that the virtual server can receive per second. |
max-connections | Maximum number of concurrent matching connections. |
monitor-attribute | Request attribute to monitor. |
Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['request-limit'] = 'request-limit-1'
props['max-connections'] = '2048'

otd_createRequestLimit(props)
```

See Also

help, otd_deleteRequestLimit, otd_listRequestLimits, otd_getRequestLimitProperties, otd_setRequestLimitProperties

2.32 otd_createRoute

Description

Use this command to create a route with a set of initial values. Based on the condition specified while creating a route, the load balancing requests are routed to the specified origin-server pool. A default route is created when a virtual-server is created.

Syntax

```python
otd_createRoute(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>route</td>
<td>Name of the route to be created.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool to which the load balancing requests must be routed.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>condition</td>
<td>A condition is an expression which if evaluates to true, will result in the rule being executed. Conditions are constructed from literals, variables, functions and operators.</td>
<td>condition cannot be specified if uri-prefix is specified.</td>
</tr>
<tr>
<td>uri-prefix</td>
<td>A uri-prefix is a URI path with wildcard patterns. If a request URI matches with the uri-prefix then the rule will be executed.</td>
<td>uri-prefix can not be specified if condition is specified.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['route'] = 'route-1'
```

2-34 WebLogic Scripting Tool Command Reference for Oracle Traffic Director
props['origin-server-pool'] = 'origin-server-pool-1'
otd_createRoute(props)

See Also
help, otd_deleteRoute, otd_listRoutes, otd_getRouteProperties, otd_setRouteProperties

2.33 otd_createStandaloneDomain

Description
Use this command to create an Oracle Traffic Director standalone domain at the given location.
This command can only be run in offline mode.

Syntax
otd_createStandaloneDomain(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance</td>
<td>Name of the instance to be created.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>domain-home</td>
<td>Path to the domain directory which should contain the Oracle Traffic Director standalone domain.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>listener-port</td>
<td>Listener port through which the server accepts requests.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>server-name</td>
<td>Valid only if origin-server-type is http or https. The server name is used in any URLs that are generated automatically by the server and sent to the client. This server name should be the virtual host name or alias name if your server uses an alias. If a colon and port number are appended to the server name then that port is used in the generated URLs.</td>
<td></td>
</tr>
<tr>
<td>ip</td>
<td>The server will bind to this Internet Protocol (IP) address for the default listener. Only traffic sent to this IP address will be serviced. * indicates that the server will listen on all IP addresses.</td>
<td></td>
</tr>
<tr>
<td>origin-server-type</td>
<td>Type of requests handled by the origin servers.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Range of values:</td>
<td>port number should be an integer between 1 and 65535, both inclusive.</td>
</tr>
<tr>
<td>Range of values:</td>
<td>*, a hostname, or an IPV4/IPV6 address.</td>
</tr>
</tbody>
</table>
## Property | Description | Comments
--- | --- | ---
origin-server | A back-end server to which Oracle Traffic Director forwards requests that it receives from clients, and from which it receives responses to client requests. The origin servers could, for example, be application servers like Oracle WebLogic Server, web servers, LDAP servers, and so on. Specified as a comma separated list of origin servers of the format host:port. |  

### Example

```python
def create_standalone_domain(props):
    # Create standalone domain
    otd_createStandaloneDomain(props)
```

### See Also

`help, otd_createStandaloneInstance, otd_deleteStandaloneInstance`

## 2.34 otd_createStandaloneInstance

### Description

Use this command to create an Oracle Traffic Director instance in an Oracle Traffic Director standalone domain.

This command can only be run in offline mode.

### Syntax

```python
domain_home = '/export/domains/otd_standalone'
server_name = 'foo.com'

# Create standalone domain
otd_createStandaloneDomain({'domain-home': domain_home, 'server-name': server_name})
```

The argument `props` is a dictionary that can contain the following properties:

### Property | Description | Comments
--- | --- | ---
domain-home | Path to the domain directory which should contain the Oracle Traffic Director standalone domain. | Mandatory.

### Example

```python
def create_standalone_instance(props):
    # Create standalone instance
    otd_createStandaloneInstance(props)
```

```python
def create_standalone_instance(props):
    # Create standalone instance
    otd_createStandaloneInstance(props)
```
See Also
help, otd_createStandaloneDomain, otd_deleteStandaloneInstance

2.35 otd_createTcpListener

Description
Use this command to create a new TCP listener with a set of initial values. When a new request comes in, Oracle Traffic Director determines which TCP proxy to send it to, based on the configured TCP listener.

Syntax

```
otd_createTcpListener(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>tcp-listener</td>
<td>Name that uniquely identifies the listener. Name can consist of one or more characters. Whitespace is not permitted.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>port</td>
<td>Port on which to listen. Range of values: port number between 1 and 65535, inclusive.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>tcp-proxy-name</td>
<td>Name that identifies the exposed TCP service.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>enabled</td>
<td>Whether the instance is enabled. Range of values: true or false. Default: true.</td>
<td></td>
</tr>
<tr>
<td>ip</td>
<td>IP address on which to listen. Range of values: hostname, or an IP address.</td>
<td></td>
</tr>
<tr>
<td>acceptor-threads</td>
<td>Acceptor threads for this listening end point. Range of values: 1 - 128. Default: auto-tuned.</td>
<td></td>
</tr>
<tr>
<td>blocking-accept</td>
<td>Enables/Disables blocking of the server Listen Socket while retaining client end points as non blocking (useful when MaxProcs &gt; 1). Range of values: true or false. Default: false.</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>Description of the TCP listener for the administrator’s reference.</td>
<td></td>
</tr>
</tbody>
</table>
## otd_createTcpProxy

### Description

Use this command to create a new TCP proxy with a set of initial values. A TCP proxy handles TCP requests through TCP listeners for traffic tunnelling to the listed origin servers. A TCP proxy can have several TCP listeners associated with it.

You can enable FTP support for a TCP proxy. This will enable the TCP proxy along with the TCP listeners referring to it to be used to front-end an FTP server.

### Syntax

```python
otd_createTcpProxy(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

### Example

```python
props = {}  
props['configuration'] = 'foo' 
props['tcp-listener'] = 'tcp-listener-1' 
props['port'] = '34567' 
props['tcp-proxy-name'] = 'tcp-proxy-1' 
otted_createTcpListener(props)
```

### See Also

- help, otd_deleteTcpListener, otd_listTcpListeners, otd_getTcpListenerProperties, otd_setTcpListenerProperties

---

### Table: Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>family</td>
<td>Protocol family. Range of values: default, inet, inet6, or inet-sdp.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: auto-tuned.</td>
<td></td>
</tr>
<tr>
<td>listen-queue-size</td>
<td>Maximum size of the operating system listen queue backlog.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: 1 - 1048576.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: 128.</td>
<td></td>
</tr>
<tr>
<td>receive-buffer-size</td>
<td>Size (in bytes) of the operating system socket receive buffer.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: size in bytes between 0 and 2147483647, inclusive.</td>
<td></td>
</tr>
<tr>
<td>send-buffer-size</td>
<td>Size (in bytes) of the operating system socket send buffer.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: size in bytes between 0 and 2147483647, inclusive.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Comments</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>tcp-proxy</td>
<td>Name that uniquely identifies the exposed TCP service.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of an existing server pool that provides the TCP service.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>enabled</td>
<td>Whether the TCP service is enabled.</td>
<td>Range of values: true or false. Default: true.</td>
</tr>
<tr>
<td>session-idle-timeout</td>
<td>Maximum timeout in seconds for load balancer to wait for receiving/sending data in the session. Range of values: an interval in seconds between 0.001 and 3600 (1 hour), inclusive.</td>
<td></td>
</tr>
<tr>
<td>protocol</td>
<td>If the protocol is 'ftp', the TCP proxy would have additional ftp properties that can be set/get using otd_setTcpProxyProperties/otd_getTcpProxyProperties. Range of values: *, ftp. Default value: *.</td>
<td></td>
</tr>
</tbody>
</table>

When otd_createTcpProxy is executed with protocol as FTP, the FTP configuration is enabled for the TCP proxy with properties ssl-termination, origin-explicit-ftps and client-explicit-ftps as false, true and true respectively.

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['tcp-proxy'] = 'bar'
props['protocol'] = 'ftp'
props['origin-server-pool-name'] = 'tcp-origin-server-pool-1'
otd_createTcpProxy(props)
```

**See Also**

help, otd_deleteTcpProxy, otd_listTcpProxies, otd_getTcpProxyProperties, otd_setTcpProxyProperties

### 2.37 otd_createConfigurationVariable

**Description**

Use this command to define a variable for use in expressions, log formats, and obj.conf parameters.

**Syntax**

```python
otd_createConfigurationVariable(props)
```

The argument props is a dictionary that can contain the following properties:
### otd_createVirtualServer

#### Description

Use this command to create a new virtual server with initial values defined.

#### Syntax

```python
otd_createVirtualServer(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name that uniquely identifies the virtual server. Name can consist of one or more characters. Whitespace is not permitted.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool for which a virtual-server is to be created.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>canonical-server-name</td>
<td>Canonical hostname of the virtual server (requests using a different hostname will be redirected to this hostname). Can be a Hostname, fully qualified domain name, IP address, or a URL prefix that contains one. The URL prefix must not specify a path.</td>
<td></td>
</tr>
</tbody>
</table>

#### Example

```
props = {}
props['configuration'] = 'foo'
props['name'] = 'bar'
props['value'] = 'baz'
otd_createConfigurationVariable(props)
```

#### See Also

`help, otd_deleteConfigurationVariable, otd_listVirtualServerVariables`

---

### otd_createVirtualServer

#### Description

Use this command to create a new virtual server with initial values defined.

#### Syntax

```python
otd_createVirtualServer(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name that uniquely identifies the virtual server. Name can consist of one or more characters. Whitespace is not permitted.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool for which a virtual-server is to be created.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>canonical-server-name</td>
<td>Canonical hostname of the virtual server (requests using a different hostname will be redirected to this hostname). Can be a Hostname, fully qualified domain name, IP address, or a URL prefix that contains one. The URL prefix must not specify a path.</td>
<td></td>
</tr>
</tbody>
</table>

---

2.38 otd_createVirtualServer
## 2.39 otd_createVirtualServerVariable

### Description

Use this command to create a variable at the virtual server level. You can use the variable in expressions, log formats, and obj.conf parameters.

### Syntax

```
otd_createVirtualServerVariable(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server for which the variable is to be created.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>name</td>
<td>Variable name consisting of letters, numbers, and underscores. Variable names must not begin with a number.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>value</td>
<td>Value corresponding to the variable.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
Example
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['name'] = 'baz'
props['value'] = 'qux'
otd_createVirtualServerVariable(props)

See Also
help, otd_deleteConfigurationVariable, otd_listVirtualServerVariables

2.40 otd_deleteCacheRule

Description
Use this command to delete the cache rule with the specified name.

Syntax
otd_deleteCacheRule(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>cache-rule</td>
<td>Name of the cache rule to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['cache-rule'] = 'cache-rule-1'
otd_deleteCacheRule(props)

See Also
help, otd_createCacheRule, otd_listCacheRules

2.41 otd_deleteCompressionRule

Description
Use this command to delete the compression rule with the specified name.
Syntax

```python
otd_deleteCompressionRule(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>compression-rule</td>
<td>Name of the compression rule to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props["configuration"] = 'foo'
props["virtual-server"] = 'bar'
props["compression-rule"] = 'compression-rule-1'
```

See Also

`help, otd_createCompressionRule, otd_getCompressionRuleProperties, otd_listCompressionRules, otd_setCompressionRuleProperties`

### 2.42 otd_deleteConfigFile

**Description**

Use this command to delete an existing configuration file.

**Syntax**

```python
otd_deleteConfigFile(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>config-file</td>
<td>Name of the configuration file to be deleted. This can be any existing configuration file except <code>server.xml</code> and object-files referred by virtual servers.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props["configuration"] = 'foo'
```
props['config-file'] = 'bar.conf'
otd_deleteConfigFile(props)

See Also

help, otd_createConfiguration, otd_listConfigurations, activate,
otd_copyConfiguration, otd_saveConfigFile, otd_deleteConfiguration

2.43 otd_deleteConfiguration

Description

Use this command to delete the configuration if it does not have any instances associated with it.

Syntax

otd_deleteConfiguration(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the configuration to be deleted.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>domain-home</td>
<td>Path to the directory which contains an Oracle Traffic Director domain.</td>
<td>Mandatory for Offline, not valid for Online.</td>
</tr>
</tbody>
</table>

Example

# Online
props = {}
props['name'] = 'foo'
otd_deleteConfiguration(props)

# Offline
readDomain('/export/domains/otd_domain')
props = {}
props['name'] = 'foo'
otd_deleteConfiguration(props)
updateDomain()
closeDomain()

See Also

help, otd_createConfiguration, otd_listConfigurations, activate,
otd_copyConfiguration, otd_saveConfigFile, otd_deleteConfiguration

2.44 otd_deleteConfigurationWebappFirewallRulesetFile

Description

Use this command to delete a ruleset file for a web application firewall installed at the configuration level.
Syntax

```python
otd_deleteConfigurationWebappFirewallRulesetFile(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>ruleset-file</td>
<td>Name of the ruleset file that needs to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['ruleset-file'] = 'bar.conf'
otd_deleteConfigurationWebappFirewallRulesetFile(props)
```

See Also

`help`, `otd_installVirtualServerWebappFirewallRulesetFile`, `otd_listVirtualServerWebappFirewallRulesetFiles`

2.45 otd_deleteContentRule

Description

Use this command to delete a content rule.

Syntax

```python
otd_deleteContentRule(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>content-rule</td>
<td>Name of the content rule to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
```
2.46 otd_deleteCrl

Description
Use this command to delete a certificate revocation list (CRL).

Syntax
```
otd_deleteCrl(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>issuer</td>
<td>Name of the CRL issuer.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
```
props = {}
props['configuration'] = 'foo'
props['issuer'] = 'CN=GlobalSign ServerSign CA,OU=ServerSign CA,O=GlobalSign nv-sa,C=BE'
otd_deleteCrl(props)
```

See Also
```
help, otd_installCrl, otd_listCrls
```

2.47 otd_deleteErrorPage

Description
Use this command to delete the error page corresponding to the specified error code.

Syntax
```
otd_deleteErrorPage(props)
```

The argument `props` is a dictionary that can contain the following properties:
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>code</td>
<td>Error code for which the error page is to be deleted. Range of values: 400 - 599.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
def props = {}
    props['configuration'] = 'foo'
    props['virtual-server'] = 'bar'
    props['code'] = '408'
    otd_deleteErrorPage(props)
```

**See Also**

help, otd_createErrorPage, otd_listErrorPages

---

**2.48 otd_deleteEvent**

**Description**

Use this command to delete a scheduled event.

**Syntax**

```
otd_deleteEvent(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>event</td>
<td>Name that uniquely identifies the event.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
def props = {}
    props['configuration'] = 'foo'
    props['event'] = 'event-1'
    otd_deleteEvent(props)
```

**See Also**

help, otd_createEvent, otd_listEvents, otd_getEventProperties, otd_setEventProperties
2.49 otd_deleteEventSubscription

**Description**

Use this command to delete an event subscription.

**Syntax**

otd_deleteEventSubscription(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>event-subscription</td>
<td>User defined name for the event subscription.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example 2-2**

```python
props = {}
props['configuration'] = 'foo'
props['event-subscription'] = 'bar'
otd_deleteEventSubscription(props)
```

2.50 otd_deleteFailoverGroup

**Description**

Use this command to delete the specified failover group. To change the VIP or any property of a failover group, you should delete the failover group and create it afresh.

When deleting a failover group, if the administration node process is running as non-root on the node where the instances are located and if at least one failover group is still available, then you must run `otd_startFailover` on those nodes as a root user. This is to manually restart the failover. On the other hand, after deleting a failover group, if no other failover groups are available for the corresponding instances, then `otd_stopFailover` must be executed to stop the failover. If you do not execute either `otd_startFailover` or `otd_stopFailover`, then the VIP associated with the deleted failover group will continue to be available.

**Syntax**

otd_deleteFailoverGroup(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
### otd_deleteFailoverGroup

**Description**

Use this command to delete a failover group with the specified name.

**Syntax**

```python
otd_deleteFailoverGroup(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual-ip</td>
<td>The VIP for the failover group to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['virtual-ip'] = '10.128.10.10'
otd_deleteFailoverGroup(props)
```

**See Also**

help, otd_createFailoverGroup, otd_toggleFailoverGroupPrimary, otd_getFailoverGroupProperties, otd_startFailover, otd_stopFailover

### otd_deleteHttpListener

**Description**

Use this command to delete an HTTP listener socket with the specified name.

**Syntax**

```python
otd_deleteHttpListener(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>http-listener</td>
<td>Name of the HTTP listener to be deleted.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>force</td>
<td>Enables the forced deletion of the HTTP listener. Default: false.</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['http-listener'] = 'http-listener-1'
otd_deleteHttpListener(props)
```

**See Also**

help, otd_createHttpListener, otd_setHttpListenerProperties, otd_setHttpListenerProperties, otd_listHttpListeners
2.52 otd_deleteInstance

**Description**
Use this command to delete the specified instance.

**Syntax**

```python
tond_deleteInstance(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration to which the instance belongs to.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>instance</td>
<td>Name of the instance to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Note:**
When this command is executed in offline mode, the instance file artifacts are deleted only if the machine specified is on the same host as that of the admin server. Otherwise, the instance file artifacts will get deleted after the start of both admin server and node manager.

**Example**

```python
# Online
props = {}
props['configuration'] = 'foo'
props['instance'] = 'otd_foo_machine1'
otd_deleteInstance(props)

# Offline
readDomain('/export/.../domains/otd_domain')
props = {}
props['configuration'] = 'foo'
props['instance'] = 'otd_foo_machine1'
otd_deleteInstance(props)
updateDomain()
closeDomain()
```

**See Also**
`help, otd_createInstance, otd_listInstances, start, stop, softRestart`

2.53 otd_deleteMimeType

**Description**
Use this command to delete a MIME type.
otd_deleteMimeType

Syntax

```python
otd_deleteMimeType(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>content-type</td>
<td>The content type of the MIME types.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['content-type'] = 'bar'
otd_deleteMimeType(props)
```

See Also

help, otd_createMimeType, otd_listMimeTypes

2.54 otd_deleteOriginServer

Description

Use this command to delete an origin server with the specified host and port.

Syntax

```python
otd_deleteOriginServer(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>host</td>
<td>IP address/Host name of the origin server to be deleted.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>port</td>
<td>Port number of the origin server to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
```

Oracle Traffic Director WLST Commands 2-51
2.55 otd_deleteOriginServerPool

Description
Use this command to delete the origin-server pool with the specified name.

Syntax
otd_deleteOriginServerPool(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
props = {}
props['configuration'] = 'foo'
props['origin-server-pool'] = 'origin-server-pool-1'
otd_deleteOriginServerPool(props)

See Also

2.56 otd_deleteRequestLimit

Description
Use this command to delete the request limit with the specified name.

Syntax
otd_deleteRequestLimit(props)

The argument props is a dictionary that can contain the following properties:
### otd_deleteRoute

#### Description

Use this command to delete the route with the specified name.

#### Syntax

```python
otd_deleteRoute(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>route</td>
<td>Name of the route to be deleted.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

#### Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['route'] = 'route-1'
otd_deleteRoute(props)
```

#### See Also

- `help`
- `otd_createRequestLimit`, `otd_listRequestLimits`, `otd_getRequestLimitProperties`, `otd_setRequestLimitProperties`
2.58 otd_deleteStandaloneInstance

**Description**
Use this command to delete an Oracle Traffic Director instance with the specified name in an Oracle Traffic Director standalone domain.
This command can only be run in offline mode.

**Syntax**
```
otd_deleteStandaloneInstance(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance</td>
<td>Name of the instance to be deleted.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>domain-home</td>
<td>Path to the domain directory which should contain the</td>
<td>Mandatory.</td>
</tr>
<tr>
<td></td>
<td>Oracle Traffic Director standalone domain.</td>
<td></td>
</tr>
</tbody>
</table>

**Example**
```
props = {}
props['instance'] = 'foo'
props['domain-home'] = '/export/domains/otd_standalone'
otd_deleteStandaloneInstance(props)
```

**See Also**
`help, otd_createStandaloneDomain, otd_createStandaloneInstance`

2.59 otd_deleteTcpListener

**Description**
Use this command to delete the TCP listener with the specified name.

**Syntax**
```
otd_deleteTcpListener(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>


## otd_deleteTcpListener

**Description**

Use this command to delete the TCP listener with the specified name.

**Syntax**

```
otd_deleteTcpListener(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp-listener</td>
<td>Name of the TCP listener to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['tcp-listener'] = 'tcp-listener-1'
otd_deleteTcpListener(props)
```

**See Also**

`help`, `otd_createTcpListener`, `otd_listTcpListeners`, `otd_getTcpListenerProperties`, `otd_setTcpListenerProperties`

## otd_deleteTcpProxy

**Description**

Use this command to delete the TCP proxy with the specified name.

**Syntax**

```
otd_deleteTcpProxy(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>tcp-proxy</td>
<td>Name of the TCP proxy to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['tcp-proxy'] = 'bar'
otd_deleteTcpProxy(props)
```

**See Also**

`help`, `otd_createTcpProxy`, `otd_listTcpProxies`, `otd_getTcpProxyProperties`, `otd_setTcpProxyProperties`
2.61 otd_deleteConfigurationVariable

**Description**

Use this command to delete a variable defined at the configuration level.

**Syntax**

```python
otd_deleteConfigurationVariable(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the variable to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['name'] = 'bar'
.otd_deleteConfigurationVariable(props)
```

**See Also**

`help, otd_createConfigurationVariable , otd_listVirtualServerVariables`

2.62 otd_deleteVirtualServer

**Description**

Use this command to delete the virtual server with the specified name.

**Syntax**

```python
otd_deleteVirtualServer(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
2.63 otd_deleteVirtualServerVariable

Description
Use this command to delete the variable with the specified name defined at the virtual server level.

Syntax
otd_deleteVirtualServerVariable(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the variable to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['name'] = 'baz'
otd_deleteVirtualServerVariable(props)

See Also
help, otd_createConfigurationVariable, otd_listVirtualServerVariables

2.64 otd_deleteVirtualServerWebappFirewallRulesetFile

Description
Use this command to delete a ruleset file for a web application firewall installed at the virtual server level.
S


tod_deleteVirtualServerWebappFirewallRulesetFile(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>ruleset-filename</td>
<td>Name of the ruleset file that needs to be deleted.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['ruleset-file'] = 'baz.conf'

tod_deleteVirtualServerWebappFirewallRulesetFile(props)
```

See Also

help, tod_installVirtualServerWebappFirewallRulesetFile,

tod_listVirtualServerWebappFirewallRulesetFiles

2.65 tod_disableOriginServerPoolMaintenance

Description

Use this command to disable maintenance for the origin server pool.

Syntax

```python
tod_disableOriginServerPoolMaintenance(props)
```

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
```
props['origin-server-pool'] = 'origin-server-pool-1'
otd_disableOriginServerPoolMaintenance(props)

See Also
help, otd_enableOriginServerPoolMaintenance, otd_getOriginServerPoolMaintenanceProperties

2.66 otd_disablePerfDump

Description
Use this command to disable access to perfdump output through a URI.

Syntax
otd_disablePerfDump(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
props = {
  'configuration': 'foo',
  'virtual-server': 'bar',
}
notd_disablePerfDump(props)

See Also
help, otd_enablePerfDump, otd_getPerfDumpProperties

2.67 otd_disableRequestLimitEvents

Description
Use this command to disable events for a specific request limit.

Syntax
otd_disableRequestLimitEvents(props)

The argument 'props' is a dictionary that must contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
### Property Description Comments

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>request-limit</td>
<td>Name of the request-limit to be disabled.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

**Example 2-3 Example**

```python
props ={}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['request-limit'] = 'request-limit-1'
.otd_disableRequestLimitEvents(props)
```

#### 2.68 otd_disableRouteAuth

**Description**

Use this command to disable the route authentication.

**Syntax**

```plaintext
otd_disableRouteAuth(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>route</td>
<td>Name of the route.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

**Example**

```python
props ={}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['route'] = 'route'
.otd_disableRouteAuth(props)
```

**See Also**

`help, otd_enableRouteAuth`
2.69 otd_disableRouteBandwidthLimit

Description

Use this command to disable bandwidth limiting at the route level.

Syntax

```python
otd_disableRouteBandwidthLimit(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>route</td>
<td>Name of the route.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>type</td>
<td>Type of bandwidth limiting to be disabled. Range of values: request or response.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['route'] = 'route-1'
props['type'] = 'request'
otd_disableRouteBandwidthLimit(props)
```

See Also

`help`, `otd_enableRouteBandwidthLimit`, `otd_getVirtualServerRequestBandwidthLimitProperties`

2.70 otd_disableStatsXml

Description

Use this command to disable access to virtual server statistics in XML format through a URI.

Syntax

```python
otd_disableStatsXml(props)
```

The argument `props` is a dictionary that can contain the following properties:
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
define props = 
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otted_disableStatsXml(props)
```

**See Also**

`help`, `otted_getStatsXml`, `otted_getStatsXmlProperties`, `otted_enableStatsXml`

### 2.71 otd_disableStatusListener

**Description**

Use this command to disable Status Listener of an instance.

**Syntax**

```python
otted_disableStatusListener(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
define props = 
props['configuration'] = 'foo'
otted_disableStatusListener(props)
```

### 2.72 otd_disableVirtualServerAccessLog

**Description**

Use this command to disable the access log for a virtual server.

**Syntax**

```python
otted_disableVirtualServerAccessLog(props)
```

The argument `props` is a dictionary that can contain the following properties:
2.73 otd_disableWebAppFirewall

Description
Use this command to disable the web application firewall for the virtual server.

Syntax
otd_disableWebAppFirewall(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_disableWebAppFirewall(props)

See Also
help, otd_enableWebAppFirewall, otd_getWebAppFirewallProperties
2.74 otd_disableVirtualServerRequestBandwidthLimit

Description
Use this command to disable request bandwidth limiting at the virtual server level.

Syntax
otd_disableVirtualServerRequestBandwidthLimit(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

Example
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
отd_disableVirtualServerRequestBandwidthLimit(props)

See Also
help, otd_enableVirtualServerRequestBandwidthLimit, otd_getVirtualServerRequestBandwidthLimitProperties

2.75 otd_disableVirtualServerResponseBandwidthLimit

Description
Use this command to disable response bandwidth limiting at the virtual server level.

Syntax
otd_disableVirtualServerResponseBandwidthLimit(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_disableVirtualServerResponseBandwidthLimit(props)
```

See Also

help, otd_enableVirtualServerResponseBandwidthLimit,
otd_getVirtualServerRequestBandwidthLimitProperties

2.76 otd_enableOriginServerPoolMaintenance

Description

Use this command to enable the maintenance for an origin-server-pool.

Syntax

```python
otd_enableOriginServerPoolMaintenance(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>response-code</td>
<td>Specifies the response code of the request when it lands on a maintenance enabled origin server pool. When this is not configured, its implicit value will be 200 if response-file is specified, else it will be 503. response-code 200 is not allowed if response-file is not configured.</td>
<td>response-code 200 is not allowed if response-file is not configured.</td>
</tr>
<tr>
<td>response-file</td>
<td>Absolute path of an HTML file to send to the client when the request lands on a maintenance enabled origin server pool.</td>
<td></td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['origin-server-pool'] = 'origin-server-pool-1'
props['response-code'] = '503'
otd_enableOriginServerPoolMaintenance(props)
```

See Also

help, otd_disableOriginServerPoolMaintenance,
otd_getOriginServerPoolMaintenanceProperties
2.77 otd_enablePerfDump

**Description**

Enables access to `perfdump` output through a URI. The `perfdump` utility collects the Oracle Traffic Director performance data and displays it in ASCII format. This utility allows you to monitor a greater variety of statistics. With `perfdump`, the statistics are unified. Rather than monitoring a single process, statistics are multiplied by the number of processes. This gives you a more accurate view of the server performance.

**Syntax**

```python
otd_enablePerfDump(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>uri</td>
<td>The URI at which the <code>perfdump</code> report should be available. Default: <code>/perf</code>.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_enablePerfDump(props)
```

**See Also**

`help`, `otd_getPerfDump`, `otd_getPerfDumpProperties`, `otd_disablePerfDump`

2.78 otd_enableRouteAuth

**Description**

Use this command to enable the route authentication.

**Syntax**

```python
otd_enableRouteAuth(props)
```

The argument `props` is a dictionary that can contain the following properties:
### 2.79 `otd_enableRouteBandwidthLimit`

**Description**

Use this command to enable bandwidth limiting at the route level.

**Syntax**

```python
otd_enableRouteBandwidthLimit(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server for which bandwidth limit is to be enabled.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
### otd_enableRequestLimitEvents

#### Description

Use this command to enable events for a specified request limit.

#### Syntax

```
otd_enableRequestLimitEvents(props)
```

The argument `props` is a dictionary that can contain the following properties:

#### Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['route'] = 'route-1'
props['type'] = 'request'
props['max-bps-per-monitor'] = '512'
.otd_enableRouteBandwidthLimit(props)
```

---

### See Also

`help`, `otd_disableRouteBandwidthLimit`, `otd_getWebappFirewallProperties`
### Property | Description | Comments
---|---|---
configuration | Name of the configuration. | Mandatory.
virtual-server | Name of the virtual server. | Mandatory.
request-limit | Name of the request limit to be enabled. | Mandatory.
event-notification-interval | Time interval (in seconds). A notification message will be sent to subscribers once every interval and will include information on all monitors that exceeded the request limit that was configured for them. Range: An interval in seconds between 1 and 32767, inclusive | Mandatory.

### Example 2-5   Example
```python
props = {} props['configuration'] = 'foo' props['virtual-server'] = 'bar' props['request-limit'] = 'request-limit-1' props['event-notification-interval'] = '60' otd_enableRequestLimitEvents(props)
```

### 2.81 otd_enableStatsXml

#### Description
Use this command to enable access to virtual server statistics in XML format through a URI.

#### Syntax
```python
otd_enableStatsXml(props)
```

The argument `props` is a dictionary that can contain the following properties:

### Property | Description | Comments
---|---|---
configuration | Name of the configuration. | Mandatory.
virtual-server | Name of the virtual server. | Mandatory.
uri | The URI at which the statistics report in XML format should be available. Default: /stats-xml. |
Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_enableStatsXml(props)
```

See Also

`help, otd_getStatsXml, otd_getStatsXmlProperties, otd_disableStatsXml`

### 2.82 otd_enableStatusListener

**Description**

Use this command to enable listeners for status check requests. In addition, use this command to change the properties of an enabled Status Listener.

**Syntax**

```python
otd_enableStatusListener(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>port</td>
<td>Port on which to listen.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td></td>
<td>Range of values: port number 1 and 65535, inclusive.</td>
<td></td>
</tr>
<tr>
<td>ip</td>
<td>IP address on which to listen.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: hostname, or an IP address.</td>
<td></td>
</tr>
<tr>
<td>family</td>
<td>Protocol family.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: default, inet, or inet6.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: default</td>
<td></td>
</tr>
</tbody>
</table>

**Example 2-6**  

```python
props = {}
props['configuration'] = 'foo'
props['port'] = '12345'
otd_enableStatusListener(props)
```

### 2.83 otd_enableWebAppFirewall

**Description**

Use this command to enable the web application firewall for a specific virtual server.

**Syntax**

```python
otd_enableWebAppFirewall(props)
```

The argument `props` is a dictionary that can contain the following properties:
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_enableWebappFirewall(props)
```

**See Also**

help, otd_disableWebAppFirewall, otd_getWebappFirewallProperties

### 2.84 otd_enableVirtualServerAccessLog

**Description**

Use this command to enable the access log for a virtual server.

**Syntax**

```python
otd_VirtualServerAccessLog(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>log-file</td>
<td>Path to the file where access logs for this configuration will be stored. Default: <code>$DOMAIN_HOME/servers/$INSTANCE_NAME/logs/access.log</code></td>
<td></td>
</tr>
<tr>
<td>format</td>
<td>A format is a string that can be used to customize the format and the fields that are logged in the access log. Default: <code>%Ses-&gt;client.ip% - %Req-&gt;vars.auth-user% %SYSDATE% &quot;%Req-&gt;reqpb.clf-request%&quot; %Req-&gt;srvhdrs.clf-status% %Req-&gt;srvhdrs.content-length% %Req-&gt;vars.ecid% %Req-&gt;vars.origin-server%</code></td>
<td></td>
</tr>
</tbody>
</table>
## Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>log-ip</td>
<td>Whether to log the IP of the client into the access log. Range of values: true or false. Default: false.</td>
<td></td>
</tr>
</tbody>
</table>

### Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['log-file'] = 'logs/access.log'
.otd_enableVirtualServerAccessLog(props)
```

### See Also

help, otd_getVirtualServerAccessLogProperties, otd_disableVirtualServerAccessLog

## 2.85 otd_enableVirtualServerRequestBandwidthLimit

### Description

Use this command to enable request bandwidth limiting at the virtual server level.

### Syntax

```python
otd_enableVirtualServerRequestBandwidthLimit(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server for which bandwidth limit is to be enabled.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>max-bps-per-monitor</td>
<td>Value in bytes/sec for maximum request bandwidth for the entire bucket. Default: 0.</td>
<td>Setting it to 0 means no bandwidth limiting is done.</td>
</tr>
<tr>
<td>max-bps-per-connection</td>
<td>Value in bytes/sec for maximum request bandwidth per connection. Default: 0.</td>
<td>Setting it to 0 means no bandwidth limiting is done.</td>
</tr>
<tr>
<td>timeout</td>
<td>Value in second. Request is aborted when it had to wait in the queue for this much time. Default: 60.</td>
<td></td>
</tr>
<tr>
<td>monitor</td>
<td>Name of bucket to which the request belongs to. Default: $ip if type is &quot;response&quot;.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Comments</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>error-code</td>
<td>HTTP error code that is returned when request is aborted.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of value: 400-599.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: 503.</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['max-bps-per-monitor'] = '1024'
props['max-bps-per-connection'] = '1024'
otd_enableVirtualServerResponseBandwidthLimit(props)
```

**See Also**

help, otd_disableVirtualServerRequestBandwidthLimit,
otd_getWebappFirewallProperties

### 2.86 otd_enableVirtualServerResponseBandwidthLimit

**Description**

Use this command to enable response bandwidth limiting at the virtual server level.

**Syntax**

```python
otd_enableVirtualServerResponseBandwidthLimit(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server for which bandwidth limit is to be enabled.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>max-bps-per-monitor</td>
<td>Value in bytes/sec for maximum request bandwidth for the entire bucket.</td>
<td>Setting it to 0 means no bandwidth limiting is done.</td>
</tr>
<tr>
<td>max-bps-per-connection</td>
<td>Value in bytes/sec for maximum request bandwidth per connection.</td>
<td>Setting it to 0 means no bandwidth limiting is done.</td>
</tr>
<tr>
<td>timeout</td>
<td>Value in second. Request is aborted when it had to wait in the queue for this much time.</td>
<td>Default: 60.</td>
</tr>
</tbody>
</table>
### Property Description Comments

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>monitor</td>
<td>Name of bucket to which the request belongs to.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: $ip if type is &quot;response&quot;.</td>
<td></td>
</tr>
<tr>
<td>error-code</td>
<td>HTTP error code that is returned when request is aborted.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of value: 400-599.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: 503.</td>
<td></td>
</tr>
</tbody>
</table>

#### Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['max-bps-per-monitor'] = '1024'
otd_enableVirtualServerResponseBandwidthLimit(props)
```

#### See Also

help, otd_disableVirtualServerRequestBandwidthLimit, otd_getWebappFirewallProperties

### 2.87 otd_exportKeyStore

#### Description

Use this command to export all the certificates within a keystore into an Oracle wallet which will be placed in the config directory of the configuration. If wallet password is set then the exported wallet is a password protected wallet (ewallet.p12), otherwise it is an auto login only wallet (cwallet.sso).

#### Syntax

```python
otd_exportKeyStore(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

#### Example

```python
props = {}
props['configuration'] = 'foo'
otd_exportKeyStore(props)
```

#### See Also

help, exportKeyStoreCertificateRequest, deleteKeyStoreEntry, importKeyStoreCertificate, getKeyStoreCertificates, generateKeyPair
**2.88 otd_forwardProxyInfo**

**Description**
Use this command to forward the proxy information. Information about a particular proxy parameter is generated and forwarded to the origin server using a HTTP header. Note that the HTTP header used by default is different depending on whether or not the origin server is Oracle WebLogic Server.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Default HTTP Header for WLS</th>
<th>Default HTTP Header for non-WLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>jroute</td>
<td>Information about request routing used by the set-origin-server function and some Servlet containers to implement session stickiness.</td>
<td>Proxy-jroute</td>
<td>Proxy-jroute</td>
</tr>
<tr>
<td>via</td>
<td>Proxy servers and protocol versions that were involved in routing a request.</td>
<td>Via</td>
<td>Via</td>
</tr>
<tr>
<td>ip</td>
<td>Client's actual IP address.</td>
<td>Wl-proxy-client-ip</td>
<td>Client-ip</td>
</tr>
<tr>
<td>xforwarded-for</td>
<td>Used to keep track of the originating client IP connecting through a proxy.</td>
<td>X-forwarded-for</td>
<td>X-forwarded-for</td>
</tr>
<tr>
<td>proxy-agent</td>
<td>Proxy server product name and version.</td>
<td>Proxy-agent</td>
<td>Proxy-agent</td>
</tr>
<tr>
<td>cache-info</td>
<td>Client cache hits.</td>
<td>Cache-info</td>
<td>Cache-info</td>
</tr>
<tr>
<td>ssl</td>
<td>A value of true/false indicating whether the client connection was over SSL</td>
<td>Wl-proxy-ssl</td>
<td>Proxy-ssl</td>
</tr>
<tr>
<td>cipher</td>
<td>Client's SSL/TLS cipher suite.</td>
<td>Proxy-cipher</td>
<td>Proxy-cipher</td>
</tr>
<tr>
<td>keysize</td>
<td>Client's SSL/TLS key size.</td>
<td>Wl-proxy-client-keysize</td>
<td>Proxy-keysize</td>
</tr>
<tr>
<td>secret-keysize</td>
<td>Size of the client's SSL/TLS secret key.</td>
<td>Wl-proxy-client-secretkeysize</td>
<td>Proxy-secret-keysize</td>
</tr>
<tr>
<td>ssl-id</td>
<td>Client's SSL/TLS session ID.</td>
<td>Proxy-ssl-id</td>
<td>Proxy-ssl-id</td>
</tr>
<tr>
<td>auth-cert</td>
<td>Client's SSL/TLS certificate in X.509 format.</td>
<td>Wl-proxy-client-cert</td>
<td>Proxy-auth-cert</td>
</tr>
<tr>
<td>user-dn</td>
<td>Distinguished name of the subject of the client's SSL/TLS certificate.</td>
<td>Proxy-user-dn</td>
<td>Proxy-user-dn</td>
</tr>
</tbody>
</table>
Parameter Name | Description | Default HTTP Header for WLS | Default HTTP Header for non-WLS
--- | --- | --- | ---
issuern-dn | Distinguished name of the issuer of the client’s SSL/TLS certificate. | Proxy-issuer-dn | Proxy-issuer-dn

**Syntax**

`otd_forwardProxyInfo(props)`

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>configuration</code></td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td><code>virtual-server</code></td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td><code>route</code></td>
<td>Name of the route.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td><code>param</code></td>
<td>Name of the proxy parameter to be blocked.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>`range of values: jroute, via, ip, cipher, proxy-agent, keysize, secret-keysize, ssl-id, issuer-dn, user-dn, auth-cert, xforwarded-for, cache-info, ssl.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>header</code></td>
<td>Name of the HTTP header used to send the proxy parameter to the origin server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>Default: the default HTTP header corresponding to the specified proxy parameter.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:**

If an incoming request includes the specified header, Oracle Traffic Director will replace the header from the request that is forwarded to the origin server with the generated header.

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['route'] = 'route-1'
props['param'] = 'via'
otd_forwardProxyInfo(props)
```

**See Also**

`help`, `otd_listProxyInfo`, `otd_blockProxyInfo`
2.89 otd_getAccessLogBufferProperties

Description
Use this command to view the access-log buffer properties. The properties that this command returns are described in otd_setAccessLogBufferProperties.

Syntax
odo_getAccessLogBufferProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
dict = {}
dict[\'configuration\'] = \'foo\'
dict_getAccessLogBufferProperties(props)
```

See Also

2.90 otd_getCacheProperties

Description
Use this command to view the cache properties. The properties that this command returns are described in otd_setCacheProperties.

Syntax
odo_getCacheProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
dict = {}
dict[\'configuration\'] = \'foo\'
dict_getCacheProperties(props)
```
See Also
help, otd_setCacheRuleProperties

2.91 otd_getCacheRuleProperties

Description
Use this command to view the cache rule properties. The properties that this command returns are described in otd_setCacheRuleProperties.

Syntax
otd_getCacheRuleProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>cache-rule</td>
<td>Name of the cache rule.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['cache-rule'] = 'cache-rule-1'
otd_getCacheRuleProperties(props)

See Also
help, otd_setCacheProperties, otd_setCacheRuleProperties

2.92 otd_getCompressionRuleProperties

Description
Use this command to view compression rule properties. The properties that this command returns are described in otd_setCompressionRuleProperties.

Syntax
otd_getCompressionRuleProperties(props)

The argument props is a dictionary that can contain the following properties:
### Property Description Comments

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>compression-rule</td>
<td>Name of the compression rule.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

#### Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['compression-rule'] = 'compression-rule-1'
```

#### See Also

help, otd_createCompressionRule, otd_deleteCompressionRule,
otd_listCompressionRules, otd_setCompressionRuleProperties

### 2.93 otd_getConfigFile

#### Description

Use this command to view the contents of a configuration file.

#### Syntax

```python
otd_getConfigFile(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration file that needs to be fetched.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>config-file</td>
<td>Name of the configuration file that needs to be fetched.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

#### Example

```python
props = {}
props['configuration'] = 'foo'
props['config-file'] = 'foo-obj.conf'
```

```
```
See Also
help, otd_createConfiguration, otd_listConfigurations, activate,
otd_copyConfiguration, otd_saveConfigFile, otd_deleteConfiguration

2.94 otd_getConfigurationAccessLogProperties

Description
Use this command to view these access-log properties for a configuration:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Path to the file where access logs for this configuration will be stored.</td>
</tr>
<tr>
<td></td>
<td>Default: $DOMAIN_HOME/servers/$INSTANCE_NAME/logs/access.log</td>
</tr>
<tr>
<td>format</td>
<td>A format is a string that can be used to customize the format and the fields that are logged in the access log.</td>
</tr>
<tr>
<td></td>
<td>Default: %Ses-&gt;client.ip% - %Req-&gt;vars.auth-user% %SYSDATE% &quot;%Req-&gt;reqpb.clf-request%&quot; %Req-&gt;srvhdrs.clf-status% %Req-&gt;srvhdrs.content-length% %Req-&gt;vars.ecid% %Req-&gt;vars.origin-server%</td>
</tr>
<tr>
<td>default-access-log-format</td>
<td>Default format for the access log entries:</td>
</tr>
<tr>
<td></td>
<td>%Ses-&gt;client.ip% - %Req-&gt;vars.auth-user% %SYSDATE% &quot;%Req-&gt;reqpb.clf-request%&quot; %Req-&gt;srvhdrs.clf-status% %Req-&gt;srvhdrs.content-length% %Req-&gt;vars.ecid% %Req-&gt;vars.origin-server%</td>
</tr>
</tbody>
</table>

Syntax
otd_getConfigurationAccessLogProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
props = {}
props['configuration'] = 'foo'
oindent otd_getConfigurationAccessLogProperties(props)

See Also
2.95 otd_getConfigurationCrlProperties

Description
Use this command to view the certificate revocation list (CRL) properties. The properties that this command returns are described in otd_setConfigurationCrlProperties.

Syntax

```python
otd_getConfigurationCrlProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
otd_getConfigurationCrlProperties(props)
```

See Also

help, otd_setConfigurationCrlProperties

2.96 otd_getConfigurationProperties

Description
Use this command to view the configuration properties. The properties that this command returns are described in otd_setConfigurationProperties.

Syntax

```python
otd_getConfigurationProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
otd_getConfigurationProperties(props)
```
See Also
help, otd_setConfigurationProperties

2.97 otd_getContentRuleProperties

Description
Use this command to view the content rule properties. The properties that this command returns are described in `otd_setContentRuleProperties`.

Syntax
```python
otd_getContentRuleProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>content-rule</td>
<td>Name of the content rule.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['content-rule'] = 'content-rule-1'
otd_getContentRuleProperties(props)
```

See Also
help, otd_setContentRuleProperties, otd_listContentRules

2.98 otd_getDnsCacheProperties

Description
Use this command to view the Domain Name Server (DNS) cache properties. The properties that this command returns are described in `otd_setDnsCacheProperties`.

Syntax
```python
otd_getDnsCacheProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
otd_getDnsCacheProperties(props)
```

**See Also**

help, otd_setDnsCacheProperties

### 2.99 otd_getDnsProperties

**Description**

Use this command to view the Domain Name Server (DNS) properties. DNS associates a standard IP address such as, 192.0.3.11, with host names such as, www.example.com. The properties that this command returns are described in otd_setDnsProperties.

**Syntax**

```python
otd_getDnsProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
otd_getDnsProperties(props)
```

**See Also**

help, otd_setDnsProperties

### 2.100 otd_getEventProperties

**Description**

Use this command to get the event properties. The properties that this command returns are described in otd_setEventProperties.
**otd_getEventProperties**

**Syntax**

```python
otd_getEventProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>event</td>
<td>Name of the event.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['event'] = 'bar'
otd_getEventProperties(props)
```

**See Also**

`help`, `otd_deleteEvent`, `otd_listEvents`, `otd_setEventProperties`

### 2.101 otd_getEventSubscriptionProperties

**Description**

Use this command to get the event subscription properties.

**Syntax**

```python
otd_getEventSubscriptionProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>event-subscription</td>
<td>User defined name of the event subscription.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

**Example 2-7**

```python
props = {}
props['configuration'] = 'foo'
props['event-subscription'] = 'bar'
otd_getEventSubscriptionProperties(props)
```
2.102 otd_getFileCacheProperties

Description
Use this command to view the file cache properties. The properties that this command returns are described in otd_setFileCacheProperties.

Syntax
otd_getFileCacheProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
otd_getFileCacheProperties(props)
```

See Also
help, otd_setFileCacheProperties

2.103 otd_getFailoverGroupProperties

Description
Use this command to view the following properties of a failover group:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual-ip</td>
<td>The VIP for which we are creating a failover for. The VIP should belong to the same subnet as that of the nodes in the failover group, and must be accessible to clients. Range of values: host name or an IPv4/IPv6 address.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>primary-instance</td>
<td>An existing instance which is designated as the primary.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>backup-instance</td>
<td>An existing instance which is designated as the backup.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>primary-nic</td>
<td>A network interface, on the node where primary-instance is running, upon which the VIP must be managed.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>backup-nic</td>
<td>A network interface, on the node where backup-instance is running, upon which the VIP must be managed.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
### otd_getFailoverGroupProperties

**Property** | **Description** | **Comments**
--- | --- | ---
router-id | A VRRP necessity, identifies the VRRP router group that are participating in failover for a VIP. The value should be unique across failover groups. Range of values: positive integer, valid range is 1-255. Default: random number between 1 - 255 |  

**Syntax**

```
otd_getFailoverGroupProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

### Example

```
props = {}
props['configuration'] = 'foo'
props['virtual-ip'] = '192.0.2.1'
otd_getFailoverGroupProperties(props)
```

**See Also**

`help`, `otd_deleteFailoverGroup`, `otd_createFailoverGroup`, `otd_toggleFailoverGroupPrimary`

### 2.104 otd_getHealthCheckProperties

**Description**

Use this command to view the health-check properties. The properties that this command returns are described in `otd_setHealthCheckProperties`.

**Syntax**

```
otd_getHealthCheckProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

### Example

```
props = {}
props['configuration'] = 'foo'
props['virtual-ip'] = '192.0.2.1'
otd_getHealthCheckProperties(props)
```

**See Also**

`help`, `otd_deleteFailoverGroup`, `otd_createFailoverGroup`, `otd_toggleFailoverGroupPrimary`
2.105 otd_getHttpListenerProperties

Description
Use this command to view the HTTP listener properties. The properties that this command returns are described in otd_setHttpListenerProperties.

Syntax

```python
otd_getHttpListenerProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>http-listener</td>
<td>Name of the HTTP listener.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['http-listener'] = 'http-listener-1'
otd_getHttpListenerProperties(props)
```

See Also

`help, otd_setHttpListenerProperties, otd_createHttpListener, otd_setHttpListenerProperties, otd_listHttpListeners, otd_deleteHttpListener`
2.106 otd_getHttpListenerSslProperties

**Description**

Use this command to view the Secure Sockets Layer (SSL) properties for an HTTP listener. SSL is a software library establishing a secure connection between the client and server. SSL is used to implement HTTPS, the secure version of HTTP.

The properties that this command returns are described in `otd_setHttpListenerSslProperties`.

**Syntax**

```
ottd_getHttpListenerSslProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>http-listener</td>
<td>Name of the HTTP listener.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['http-listener'] = 'http-listener-1'
ottd_getHttpListenerSslProperties(props)
```

**See Also**

`help`, `otd_setHttpListenerSslProperties`

2.107 otd_getHttpProperties

**Description**

Use this command to view the HTTP properties. The properties that this command returns are described in `otd_setHttpProperties`.

**Syntax**

```
ottd_getHttpProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
Example

```python
events = {}
events['configuration'] = 'foo'
otd_getHttpProperties(props)
```

See Also

help, otd_setHttpProperties

2.108 otd_getHttpThreadPoolProperties

Description

Use this command to view the thread-pool properties. You can use thread pools to allocate a certain number of threads to a specific service. By defining a pool with the maximum number of threads as 1, only one request is allowed to the specified service function. The properties that this command returns are described in `otd_setHttpThreadPoolProperties`.

Syntax

```python
otd_getHttpThreadPoolProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
events = {}
events['configuration'] = 'foo'
otd_getHttpThreadPoolProperties(props)
```

See Also

help, otd_setHttpThreadPoolProperties

2.109 otd_getKeepaliveProperties

Description

Use this command to view the keep-alive properties. The keep-alive or HTTP/1.1 persistent connection handling subsystem in is designed to be scalable. If the configuration does not conform as required, the performance can be less than optimal if the workload is not persistent (that is, HTTP/1.0 without the KeepAlive header), or for a lightly loaded system that is primarily servicing keep-alive connections. The properties that this command returns are described in `otd_setKeepaliveProperties`.

Syntax

```python
otd_getKeepAliveProperties(props)
```
The argument \texttt{props} is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

### Example

```python
props = {}
props['configuration'] = 'foo'
otd_getKeepaliveProperties(props)
```

### See Also

\texttt{help, otd_setKeepaliveProperties}

#### 2.110 otd_getLogProperties

### Description

Use this command to view the log properties. The properties that this command returns are described in \texttt{otd_setLogProperties}.

### Syntax

```python
otd_getLogProperties(props)
```

The argument \texttt{props} is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

### Example

```python
props = {}
props['configuration'] = 'foo'
otd_getLogProperties(props)
```

### See Also

\texttt{help, otd_setLogProperties}

#### 2.111 otd_getOriginServerPoolMaintenanceProperties

### Description

Use this command to view the maintenance properties for the origin server pool. The properties that this command returns are described in \texttt{otd_enableOriginServerPoolMaintenance}.
## Syntax

```
otd_getOriginServerPoolMaintenanceProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

### Example

```
props = {}
props['configuration'] = 'foo'
props['origin-server-pool'] = 'origin-server-pool-1'
otd_getOriginServerPoolMaintenanceProperties(props)
```

### See Also

`help`, `otd_disableOriginServerPoolMaintenance`, `otd_enableOriginServerPoolMaintenance`

---

## 2.112 otd_getOriginServerPoolProperties

### Description

Use this command to view the origin-server pool properties. The properties that this command returns are described in `otd_setOriginServerPoolProperties`.

### Syntax

```
otd_getOriginServerPoolProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

### Example

```
props = {}
props['configuration'] = 'foo'
props['origin-server-pool'] = 'origin-server-pool-1'
otd_getOriginServerPoolProperties(props)
```
See Also
help, otd_createOriginServerPool, otd_deleteOriginServerPool,
ott_listOriginServerPools, otd_setOriginServerPoolProperties

2.113 otd_getOriginServerProperties

Description
Use this command to view origin server properties. The properties that this command
returns are described in otd_setOriginServerPoolProperties.

Syntax
otd_getOriginServerProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>host</td>
<td>IP address/host name of the origin server.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>port</td>
<td>Port number of the origin server.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

Example
props = {}
props['configuration'] = 'foo'
props['origin-server-pool'] = 'origin-server-pool-1'
props['host'] = 'www.example.com'
props['port'] = '12345'
ott_getOriginServerProperties(props)

See Also
help, otd_createOriginServer, otd_deleteOriginServer, otd_listOriginServers,
ott_setOriginServerPoolProperties

2.114 otd_getOriginServerPoolSslProperties

Description
Use this command to view the SSL properties of the origin server. The properties that
this command returns are described in otd_setOriginServerPoolSslProperties.
Syntax

```python
otd_getOriginServerPoolSslProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['origin-server-pool'] = 'origin-server-pool-1'
otd_getOriginServerPoolSslProperties(props)
```

See Also

`help, otd_setOriginServerPoolSslProperties`

### 2.115 otd_getPartitionAccessLogProperties

**Description**

Use this command to view these access-log properties for a partition:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log-file</td>
<td>Path to the file where access logs for the partition will be stored.</td>
</tr>
<tr>
<td></td>
<td>Default: <code>$/DOMAIN_HOME/servers/$INSTANCE_NAME/logs/$PARTITION_NAME.log</code></td>
</tr>
<tr>
<td>format</td>
<td>A format is a string that can be used to customize the format and the fields that are logged in the partition access log.</td>
</tr>
<tr>
<td></td>
<td>Default: <code>%Ses-&gt;client.ip% - %Req-&gt;vars.auth-user% %SYSDATE% &quot;%Req-&gt;reqpb.clf-request%&quot; %Req-&gt;srvhdrs.clf-status% %Req-&gt;srvhdrs.content-length% %Req-&gt;vars.ecid% %Req-&gt;vars.origin-server%</code></td>
</tr>
<tr>
<td>default-access-log-format</td>
<td>Default format for the partition access log entries:</td>
</tr>
<tr>
<td></td>
<td><code>%Ses-&gt;client.ip% - %Req-&gt;vars.auth-user% %SYSDATE% &quot;%Req-&gt;reqpb.clf-request %&quot; %Req-&gt;srvhdrs.clf-status% %Req-&gt;srvhdrs.content-length% %Req-&gt;vars.ecid% %Req-&gt;vars.origin-server%</code></td>
</tr>
</tbody>
</table>

Syntax

```python
otd_getPartitionAccessLogProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration. This must be the name of the configuration that is specified while registering the Oracle Traffic Director runtime with the Lifecycle Manager.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>partition</td>
<td>Name of the partition.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'mt'
props['partition'] = 'WLSPartition'
otd_getPartitionAccessLogProperties(props)
```

See Also
help, otd_listPartitions, otd_setPartitionAccessLogProperties

2.116 otd_getPerfDump

Description
Use this command to view the runtime statistics for various subsystems as a text report on the browser.

Syntax

```python
otd_getPerfDump(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance-name</td>
<td>Name of the instance.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>domain-home</td>
<td>Path to the directory which contains the Oracle Traffic Director domain</td>
<td>Mandatory for Offline, not valid for Online.</td>
</tr>
</tbody>
</table>

Example

```python
# Online
props = {}
props['instance-name'] = 'otd_abcl23.example.com'
otd_getPerfDump(props)

# Offline
props = {}
props['domain-home'] = '/export/domains/otd_domain'
props['instance-name'] = 'otd_abcl23.example.com'
otd_getPerfDump(props)
```
2.117 otd_getPerfDumpProperties

Description

Use this command to get the following perfdump properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether perfdump is enabled.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td></td>
<td>Default is false.</td>
<td></td>
</tr>
<tr>
<td>uri</td>
<td>The URI at which the perfdump report should be available.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td></td>
<td>Default: /perf.</td>
<td></td>
</tr>
</tbody>
</table>

Syntax

```python
otd_getPerfDumpProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_getPerfDumpProperties(props)
```

See Also

help, otd_getPerfDump, otd_enablePerfDump, otd_disablePerfDump

2.118 otd_getRequestLimitProperties

Description

Use this command to view the request-limit properties. The properties that this command returns are described in otd_setRequestLimitProperties.

Syntax

```python
otd_getRequestLimitProperties(props)
```
The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>request-limit</td>
<td>Name of the request limit rule.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>event-notification-interval</td>
<td>Time interval (in seconds). A notification message will be sent to subscribers once every interval and will include information on all monitors that exceeded the request limit that was configured for them.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['request-limit'] = 'request-limit-1'
otd_getRequestLimitProperties(props)
```

See Also

`help, otd_listRequestLimits, otd_deleteRequestLimit, otd_createRequestLimit, otd_setRequestLimitProperties`

2.119 `otd_getRouteAuthProperties`

**Description**

Use this command to view the following route authentication properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>auth-user</td>
<td>Specifies the authenticated user.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>auth-header</td>
<td>Specifies the name of the authentication header.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default is Authorization.</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

```python
otd_getRouteAuthProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:
2.120 otd_getRouteBandwidthLimitProperties

Description
Use this command to get bandwidth limiting properties at the route level.

Syntax
otd_getRouteBandwidthLimitProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>route</td>
<td>Name of the route.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>type</td>
<td>Type of bandwidth limiting.</td>
<td>Range of values: request or response.</td>
</tr>
</tbody>
</table>

Example
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['route'] = 'route-1'
otd_getRouteBandwidthLimitProperties(props)
props['virtual-server'] = 'bar'
props['route'] = 'route-1'
props['type'] = 'request'
otd_getRouteBandwidthLimitProperties(props)

See Also
help, otd_enableRouteBandwidthLimit, otd_disableRouteBandwidthLimit

2.121 otd_getRouteProperties

Description
Use this command to view route properties. The properties that this command returns are described in otd_setRouteProperties.

Syntax
otd_getRouteProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>route</td>
<td>Name of the route.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['route'] = 'route-1'
otd_getRouteProperties(props)

See Also
help, otd_createRoute, otd_listRoutes, otd_setRouteProperties, otd_deleteRoute

2.122 otd_getSnmpProperties

Description
Use this command to view the Simple Network Management Protocol (SNMP) properties. The properties that this command returns are described in otd_setSnmpProperties.

Syntax
otd_getSnmpProperties(props)
The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
globals = {}
globals['configuration'] = 'foo'
print otd_getSnmpProperties(globals)
```

**See Also**

help, otd_stopSnmpSubAgent, otd_startSnmpSubAgent, otd_setSnmpProperties

### 2.123 otd_getSslSessionCacheProperties

**Description**

Use this command to view the properties that are currently defined for caching SSL session data. The properties that this command returns are described in `otd_setSslSessionCacheProperties`.

**Syntax**

```python
print otd_getSslSessionCacheProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
globals = {}
globals['configuration'] = 'foo'
print otd_getSslSessionCacheProperties(globals)
```

**See Also**

help, otd_setSslSessionCacheProperties

### 2.124 otd_getStatsProperties

**Description**

Use this command to view properties of the statistics collection subsystem. The properties that this command returns are described in `otd_setStatsProperties`.
**otd_getStatsProperties**

Syntax

```
otd_getStatsProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
otd_getStatsProperties(props)
```

**See Also**

`help, otd_setStatsProperties`

---

## 2.125 otd_getStatsXml

**Description**

Use this command to view runtime statistics for various subsystems in XML format.

**Syntax**

```
otd_getStatsXml(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance</td>
<td>Name of the instance.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>domain-home</td>
<td>Path to the directory which contains the Oracle Traffic Director domain.</td>
<td>Mandatory for Offline, not valid for Online.</td>
</tr>
</tbody>
</table>

**Example**

**Online syntax:**

```python
props = {}
props['instance'] = 'otd_foo_machine1'
otd_getStatsXml(props)
```

**Offline syntax:**

```python
props = {}
props['domain-home'] = '/export/domains/otd_domain'
props['instance'] = 'otd_foo_machine1'
otd_getStatsXml(props)
```
See Also
help, otd_getStatsXmlProperties, otd_enableStatsXml, otd_disableStatsXml

2.126 otd_getStatsXmlProperties

Description
Use this command to view these properties defined for gathering and reporting statistical data in XML format:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether access to virtual-server statistics in XML format through a URI is enabled.</td>
<td>false</td>
</tr>
<tr>
<td>url</td>
<td>The URI at which the statistics report in XML format should be available.</td>
<td>/stats-xml</td>
</tr>
</tbody>
</table>

Syntax

otd_getStatsXmlProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_getStatsXmlProperties(props)
```

See Also
help, otd_enableStatsXml, otd_disableStatsXml

2.127 otd_getStatusListenerProperties

Description
Use this command to view the Status Listener properties.

Syntax

otd_getStatusListenerProperties(props)

The argument props is a dictionary that can contain the following properties:
**Property** | **Description** | **Comments**
---|---|---
configuration | Name of the configuration. | Mandatory.

*Example 2-8  Example*

```python
props = {}
props['configuration'] = 'foo'
.otd_getStatusListenerProperties(props)
```

### 2.128 otd_getStatusListenerSslProperties

**Description**
Use this command to view the SSL properties of a Status Listener.

**Syntax**

```python
.otd_getStatusListenerSslProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

**Property** | **Description** | **Comments**
---|---|---
configuration | Name of the configuration. | Mandatory.

*Example 2-9  Example*

```python
props = {}
props['configuration'] = 'foo'
.otd_getStatusListenerSslProperties(props)
```

### 2.129 otd_getTcpAccessLogProperties

**Description**
Use this command to view these properties of the TCP access log. The properties that this command returns are described in `otd_setTcpAccessLogProperties`.

**Property** | **Description**
---|---
file | Path to the file where the access log for this configuration will be stored.
Default: $DOMAIN_HOME/servers/$INSTANCE_NAME/logs/tcp-access.log

**Syntax**

```python
.otd_getTcpAccessLogProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:
2.130 otd_getTcpListenerProperties

**Description**

Use this command to view the properties of the TCP listener. The properties that this command returns are described in `otd_setTcpListenerProperties`.

**Syntax**

```python
otd_getTcpListenerProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>tcp-listener</td>
<td>Name of the TCP listener.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
def example():
    props = {}
    props['configuration'] = 'foo'
    props['tcp-listener'] = 'tcp-listener-1'
    otd_getTcpListenerProperties(props)
```

**See Also**

`help`, `otd_createTcpListener`, `otd_deleteTcpListener`, `otd_listTcpListeners`, `otd_setTcpListenerProperties`
2.131 otd_getTcpListenerSslProperties

Description
Use this command to view the Secure Sockets Layer (SSL) properties for a TCP listener. SSL is a software library establishing a secure connection between the client and server. SSL is used to implement HTTPS, the secure version of HTTP.

The properties that this command returns are described in otd_setTcpListenerSslProperties.

Syntax
otd_getTcpListenerSslProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>tcp-listener</td>
<td>Name of the TCP listener.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
```
props = {}
props['configuration'] = 'foo'
props['tcp-listener'] = 'tcp-listener-1'
otd_getTcpListenerSslProperties(props)
```

See Also
help, otd_setTcpListenerSslProperties

2.132 otd_getTcpProxyProperties

Description
Use this command to view the properties of the TCP proxy.

Syntax
otd_getTcpProxyProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Property | Description | Comments
---------|-------------|--------
tcp-proxy | Name that uniquely identifies the exposed TCP service. | Mandatory.

The properties that this command returns are described in `otd_setTcpProxyProperties`.

Example

```python
def example():
    props = {}
    props['configuration'] = 'foo'
    props['tcp-proxy'] = 'bar'
    otd_getTcpProxyProperties(props)
```

See Also

`help`, `otd_createTcpProxy`, `otd_deleteTcpProxy`, `otd_listTcpProxies`, `otd_setTcpProxyProperties`

### 2.133 `otd_getTcpThreadPoolProperties`

**Description**
Use this command to view the properties of the TCP thread pool. The properties that this command returns are described in `otd_setTcpThreadPoolProperties`.

**Syntax**

```python
def example():
    otd_getTcpThreadPoolProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

Property | Description | Comments
---------|-------------|--------
configuration | Name of the configuration. | Mandatory.

Example

```python
def example():
    props = {};
    props['configuration'] = 'foo'
    otd_getTcpThreadPoolProperties(props)
```

See Also

`help`, `otd_setTcpThreadPoolProperties`

### 2.134 `otd_getVirtualServerAccessLogProperties`

**Description**
Use this command to view the following access-log properties:
## enabled

Whether the server writes to this access log.

Range of values: true or false.

Default: true.

## file

Path to the file where access logs for this configuration will be stored.

Default: `$DOMAIN_HOME/servers/$INSTANCE_NAME/logs/access.log`

## format

A format is a string that can be used to customize the format and the fields that are logged in the access log.

Default: `%Ses->client.ip% - %Req->vars.auth-user% %SYSDATE% "%Req->reqpb.clf-request%" %Req->srvhdrs.clf-status% %Req->srvhdrs.content-length% %Req->vars.ecid% %Req->vars.origin-server%`

## log-ip

Whether to log the IP of the client into the access log.

Range of values: true or false.

Default: false.

## default-access-log-format

Default format for the access log entries:

```
%Ses->client.ip% - %Req->vars.auth-user% %SYSDATE% "%Req->reqpb.clf-request%" %Req->srvhdrs.clf-status% %Req->srvhdrs.content-length% %Req->vars.ecid% %Req->vars.origin-server%
```

### Syntax

```
com.oracle.td.otd.getVirtualServerAccessLogProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

### Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
com.oracle.td.otd.getVirtualServerAccessLogProperties(props)
```

### See Also

2.135 otd_getVirtualServerRequestBandwidthLimitProperties

Description
Use this command to get request bandwidth limiting properties at the virtual server level.

Syntax
otd_getVirtualServerRequestBandwidthLimitProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
ottd_getVirtualServerRequestBandwidthLimitProperties(props)

See Also
help, otd_enableVirtualServerRequestBandwidthLimit, otd_disableVirtualServerRequestBandwidthLimit

2.136 otd_getVirtualServerResponseBandwidthLimitProperties

Description
Use this command to get response bandwidth limiting properties at the virtual server level.

Syntax
otd_getVirtualServerResponseBandwidthLimitProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_getVirtualServerBandwidthLimitProperties(props)
```

See Also
- help, otd_enableVirtualServerResponseBandwidthLimit,
- otd_disableVirtualServerResponseBandwidthLimit

### 2.137 otd_getVirtualServerProperties

**Description**

Use this command to view the properties of a virtual server. The properties that this command returns are described in otd_setVirtualServerProperties.

**Syntax**

```python
otd_getVirtualServerProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_getVirtualServerProperties(props)
```

See Also
- help, otd_createVirtualServer, otd_setVirtualServerProperties, otd_listVirtualServers, otd_copyVirtualServer, otd_deleteVirtualServer

### 2.138 otd_getVirtualServerSslProperties

**Description**

Use this command to get the SSL properties for a virtual server. The properties that this command returns are documented in otd_setVirtualServerSslProperties.

**Syntax**

```python
otd_getVirtualServerSslProperties(props)
```

2-108 WebLogic Scripting Tool Command Reference for Oracle Traffic Director
The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_getVirtualServerSslProperties(props)
```

**See Also**

`help, otd_setVirtualServerSslProperties`

---

2.139 `otd_getWebappFirewallProperties`

**Description**

Use this command to view the properties of a web application firewall. The properties that this command returns are described in `otd_setWebappFirewallProperties`.

**Syntax**

```python
otd_getWebappFirewallProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_getWebappFirewallProperties(props)
```

**See Also**

`help, otd_createVirtualServer, otd_setVirtualServerProperties, otd_listVirtualServers, otd_copyVirtualServer, otd_deleteVirtualServer, otd_getVirtualServerProperties`
2.140 otd_installConfigurationWebappFirewallRulesetFile

**Description**

Use this command to upload a file containing Web Application Firewall (WAF) rules into the server configuration directory. These rules will apply server-wide across all virtual servers.

**Syntax**

```python
otd_installConfigurationWebappFirewallRulesetFile(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>file-path</td>
<td>The full path of the ruleset file to be installed.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>file-on-server</td>
<td>Whether the file to be installed is available on the administration server host.</td>
<td>Default is false.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['file-path'] = '/export/bar.conf'
otd_installConfigurationWebappFirewallRulesetFile(props)
```

**See Also**

`help`, `otd_deleteVirtualServerWebappFirewallRulesetFile`, `otd_listVirtualServerWebappFirewallRulesetFiles`

---

2.141 otd_installCrl

**Description**

Use this command to install a certificate revocation list (CRL) issued by a Certificate Authority (CA) into the server configuration directory. A CRL lists all certificates that either client or server users should no longer trust.

**Syntax**

```python
otd_installCrl(props)
```

The argument `props` is a dictionary that can contain the following properties:
### otd_installVirtualServerWebappFirewallRulesetFile

#### Description

Use this command to upload the web application firewall ruleset files into the server configuration directory. These rules will apply only to requests handled by the specified virtual server.

#### Syntax

```python
otd_installVirtualServerWebappFirewallRulesetFile(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>file-path</td>
<td>The full path of the ruleset file to be installed. This file should be available on the administration server host.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>file-on-server</td>
<td>Whether the file to be installed is available on the administration server host. Default is false.</td>
<td></td>
</tr>
</tbody>
</table>
Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['file-path'] = '/export/rulesets/baz.conf'
otd_installVirtualServerWebappFirewallRulesetFile(props)
```

See Also

help, otd_deleteVirtualServerWebappFirewallRulesetFile, otd_listVirtualServerWebappFirewallRulesetFiles

2.143 otd_listCacheRules

Description

Use this command to view a list of caching rules defined for the specified virtual server.

Syntax

```python
otd_listCacheRules(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

This command returns a list of strings each representing the name of a cache rule.

Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_listCacheRules(props)
```

See Also

help, otd_createCacheRule, otd_deleteCacheRule, otd_setCacheRuleProperties

2.144 otd_listCertificates

Description

Use this command to list all the certificates of type 'Certificate' present in the keystore.

Syntax

```python
otd_listCertificates(props)
```
The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

This command returns a list of maps, each map representing one certificate with properties `alias`, `subject`, `issuer`, `serial-number` and `key-type`.

**Example**

```python
props = {}
props['configuration'] = 'foo'
otd_listCertificates(props)
```

**See Also**

`help, exportKeyStoreCertificateRequest, deleteKeyStoreEntry, importKeyStoreCertificate, getKeyStoreCertificates, generateKeyPair`

### 2.145 otd_listCompressionRules

**Description**

Use this command to list compression rules defined for the specified virtual server.

**Syntax**

`otd_listCompressionRules(props)`

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

This command returns a list of strings each representing the name of a compression rule.

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_listCompressionRules(props)
```

**See Also**

`help, otd_createCompressionRule, otd_deleteCompressionRule, otd_setCompressionRuleProperties, otd_getCompressionRuleProperties`
2.146 otd_listConfigFiles

**Description**

Use this command to list configuration files pertaining to the specified configuration.

**Syntax**

```
otd_listConfigFiles(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

This command returns a list of strings each representing the name of a configuration file.

**Example**

```python
props = {}
props['configuration'] = 'foo'
otd_listConfigFiles(props)
```

**See Also**

`help, otd_createConfiguration, activate, otd_copyConfiguration, otd_saveConfigFile, otd_deleteConfiguration, otd_getConfigFile, otd_listConfigurations`

2.147 otd_listConfigurations

**Description**

Use this command to return a list of strings each representing the name of an existing configuration.

**Syntax**

```
otd_listConfigurations()
```

**Example**

```python
# Online
otd_listConfigurations()
```

```python
# Offline
readDomain('/export/domains/otd_domain')
otd_listConfigurations()
closeDomain()
```
See Also
help, otd_createConfiguration, activate, otd_copyConfiguration, otd_saveConfigFile, otd_deleteConfiguration, otd_getConfigFile, otd_listConfigFiles

2.148 otd_listConfigurationWebappFirewallRulesetFiles

Description
Use this command to list all web application firewall rulesets defined for a configuration.

Syntax

```python
otd_listConfigurationWebappFirewallRulesetFiles(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

This command returns a list of strings each representing the name of a web application ruleset file.

Example

```python
props = {}
prefs['configuration'] = 'foo'
otd_listConfigurationWebappFirewallRulesetFiles(props)
```

See Also
help, otd_installVirtualServerWebappFirewallRulesetFile, otd_listVirtualServerWebappFirewallRulesetFiles, otd_deleteVirtualServerWebappFirewallRulesetFile

2.149 otd_listContentRules

Description
Use this command to list the content rules.

Syntax

```python
otd_listContentRules(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
### 2.150 otd_listCrls

**Description**
Use this command to list all installed certificate revocation lists (CRLs).

**Syntax**
`otd_listCrls(props)`

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**
```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_listCrls(props)
```

**See Also**
- `help`
- `otd_installCrl`
- `otd_deleteCrl`

### 2.151 otd_listErrorPages

**Description**
Use this command to list all the error pages and their corresponding error codes.

**Syntax**
`otd_listErrorPages(props)`

**Example**
```python
props = {}
props['configuration'] = 'foo'
otd_listErrorPages(props)
```

**See Also**
- `help`
- `otd_installCrl`
- `otd_deleteCrl`
The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_listErrorPages(props)
```

**See Also**

help, otd_createErrorPage, otd_deleteErrorPage

### 2.152 otd_listEvents

**Description**

Use this command to list all scheduled events for a configuration.

**Syntax**

```python
otd_listEvents(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
otd_listEvents(props)
```

**See Also**

help, otd_createEvent, otd_deleteEvent, otd_getEventProperties, otd_setEventProperties
2.153 otd_listFailoverGroups

**Description**

Use this command to return a list of strings each representing the `virtual-ip` of an existing failover group.

**Syntax**

```python
otd_listFailoverGroups(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
otd_listFailoverGroups(props)
```

**See Also**

`help`, `otd_createFailoverGroup`, `otd_deleteFailoverGroup`, `otd_getFailoverGroupProperties`, `otd_toggleFailoverGroupPrimary`

2.154 otd_listFailoverInstances

**Description**

Use this command to list all the instances present in an active-active failover group type. This command returns a list of maps, each map representing one instance with properties `instance` and `NIC`.

**Syntax**

```python
otd_listFailoverInstances(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-ip</td>
<td>Virtual IP that uniquely identifies the failure group.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
Example 2-10 Example
To list failover groups, see the following example with the below details.
```
Configuration: ha
Instance: 1.example.com
Virtual IP address: 10.128.67.44

```python
props = {}
props['configuration'] = 'ha'
props['virtual-ip'] = '10.128.67.44'
props['instance'] = '1.example.com'
```

```
2.155 otd_listHttpListeners
```

**Description**

Use this command to list the names of the HTTP listeners defined for the configuration.

**Syntax**

```python
otd_listHttpListeners(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

This command returns a list of strings each representing the name of an HTTP listener.

**Example**

```python
props = {}
props['configuration'] = 'foo'
```

```python
otd_listHttpListeners(props)
```

**See Also**

`help, otd_createHttpListener, otd_setHttpListenerProperties, otd_setHttpListenerProperties, otd_deleteHttpListener`

```
2.156 otd_listInstances
```

**Description**

Use this command to list all instances of this configuration.

**Syntax**

```python
otd_listInstances(props)
```

The argument `props` is a dictionary that can contain the following properties:
This command returns a list of strings each representing the name of an instance.

Example

```python
#Online
props = {}
props['configuration'] = 'foo'
otd_listInstances(props)

#Offline
readDomain(':/export/domains/otd_domain')
props = {}
props['configuration'] = 'foo'
otd_listInstances(props)
closeDomain()
```

See Also

help, otd_createInstance, otd_deleteInstance, start, stop, softRestart

2.157 otd_listMimeTypes

Description

Use this command to list MIME types.

Syntax

```python
otd_listMimeTypes(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
otd_listMimeTypes(props)
```

See Also

help, otd_createMimeType, otd_deleteMimeType
2.158 otd_listOriginServers

Description
Use this command to view a list of origin-servers defined in a pool.

Syntax

```python
otd_listOriginServers(props)
```

The argument `props` is a dictionary that can contain the following properties, in addition to the properties described in `otd_createOriginServer`:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['origin-server-pool'] = 'origin-server-pool-1'
otd_listOriginServers(props)
```

See Also

`help, otd_createOriginServer, otd_deleteOriginServer, otd_getOriginServerProperties, otd_setOriginServerProperties`

2.159 otd_listOriginServerPools

Description
Use this command to list origin-server pools defined for a configuration.

Syntax

```python
otd_listOriginServerPools(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

The command returns a list of strings each representing the name of an origin server pool.
**Example**

```python
props = {}
props['configuration'] = 'foo'
otd_listOriginServerPools(props)
```

**See Also**


### 2.160 otd_listPartitions

**Description**

Use this command to list all the Oracle Traffic Director partitions in a given configuration. The Oracle Traffic Director partition name should be same as the WLS partition name that it front-ends. In that case, it lists all the WLS partitions that are front-ended by Oracle Traffic Director.

**Syntax**

```python
otd_listPartitions(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration. This must be the name of the configuration that is specified while registering the Oracle Traffic Director runtime with the Lifecycle Manager.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'mt'
otd_listPartitions(props)
```

**See Also**


### 2.161 otd_listProxyInfo

**Description**

Use this command to list the information about the proxy parameters configured for the route.

**Syntax**

```python
otd_listProxyInfo(props)
```

The argument `props` is a dictionary that can contain the following properties:
2.162 otd_listRequestLimits

Description
Use this command to list the request limit conditions defined for a virtual server.

Syntax

```python
otd_listRequestLimits(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

This command returns a list of strings each representing the name of a request limit.

Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['route'] = 'route-1'
otd_listRequestLimits(props)
```
See Also
help, otd_getRequestLimitProperties, otd_setRequestLimitProperties, otd_deleteRequestLimit, otd_createRequestLimit

2.163 otd_listResourceGroups

Description
Provides information about all the resource-groups that exist under a given partition. The resource-group information contains the information about all the virtual-targets that the resource-group is targeted to. The virtual-target information in turn includes the virtual-target name and the corresponding Oracle Traffic Director artifacts information such as route name, virtual-server name and origin-server-pool name.

Syntax
otd_listResourceGroups(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration. This must be the name of the configuration that is specified while registering the Oracle Traffic Director runtime with the Lifecycle Manager.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>partition</td>
<td>Name of the partition.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
props = {}
props['configuration'] = 'mt'
props['partition'] = 'WLSPartition'
otd_listResourceGroups(props)

See Also
help, otd_listPartitions, otd_getPartitionAccessLogProperties, otd_setPartitionAccessLogProperties

2.164 otd_listRoutes

Description
Use this command to list the rules defined for a virtual server.

Syntax
otd_listRoutes(props)

The argument props is a dictionary that can contain the following properties:
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

This command returns a list of strings each representing the name of a route.

**Example**
```
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_listRoutes(props)
```

**See Also**
help, otd_createRoute, otd_deleteRoute, otd_getRouteProperties, otd_setRouteProperties

### 2.165 otd_listTcpListeners

**Description**
Use this command to list all the TCP listeners.

**Syntax**
```
otd_listTcpListeners(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

This command returns a list of strings each representing the name of a TCP listener.

**Example**
```
props = {}
props['configuration'] = 'foo'
otd_listTcpListeners(props)
```

**See Also**
help, otd_createTcpListener, otd_deleteTcpListener, otd_getTcpListenerProperties, otd_setTcpListenerProperties
2.166 otd_listTcpProxies

**Description**

Use this command to list all the TCP proxies.

**Syntax**

```python
otd_listTcpProxies(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

This command returns a list of strings each representing the name of a TCP proxy.

**Example**

```python
props = {}
props['configuration'] = 'foo'
otd_listTcpProxies(props)
```

**See Also**

`help`, `otd_createTcpProxy`, `otd_deleteTcpProxy`, `otd_getTcpProxyProperties`, `otd_setTcpProxyProperties`

2.167 otd_listConfigurationVariables

**Description**

Use this command to list all the variables defined at the configuration level.

**Syntax**

```python
otd_listConfigurationVariables(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
otd_listConfigurationVariables(props)
```
See Also
help, otd_createConfigurationVariable, otd_deleteConfigurationVariable

2.168 otd_listVirtualServers

Description
Use this command to list all virtual-servers defined for a configuration.

Syntax

```python
otd_listVirtualServers(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

This command returns a list of strings each representing the name of a virtual server.

Example

```python
props = {}
props['configuration'] = 'foo'
otd_listVirtualServers(props)
```

See Also
help, otd_createVirtualServer, otd_setVirtualServerProperties, otd_getVirtualServerProperties, otd_deleteVirtualServer, otd_copyVirtualServer

2.169 otd_listVirtualServerVariables

Description
Use this command to list all variables defined at the configuration level.

Syntax

```python
otd_listVirtualServerVariables(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_listVirtualServerVariables(props)
```

See Also

`help, otd_createVirtualServerVariable, otd_deleteVirtualServerVariable`

### 2.170 otd_listVirtualServerWebappFirewallRulesetFiles

**Description**

Use this command to list all web application firewall rulesets defined for a virtual server.

**Syntax**

```python
otd_listVirtualServerWebappFirewallRulesetFiles(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

This command returns a list of strings each representing the name of a web application ruleset file.

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
otd_listVirtualServerWebappFirewallRulesetFiles(props)
```

See Also

`help, otd_installVirtualServerWebappFirewallRulesetFile, otd_deleteVirtualServerWebappFirewallRulesetFile`

### 2.171 otd_removeFailoverInstance

**Description**

Use this command to remove a failover instance. This command is valid only for the active-active failover type.
Syntax

```python
otd_removeFailoverInstance(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-ip</td>
<td>Virtual IP that uniquely identifies the failure group.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>instance</td>
<td>An instance which is part of this failover group.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example 2-11  Example**

Configuration: ha

Instance: 1.example.com

Virtual IP address: 10.128.67.44

```python
props = {}
props['configuration'] = 'ha'
props['virtual-ip'] = '10.128.67.44'
props['instance'] = '1.example.com'
```

```python
otd_removeFailoverInstance(props)
```

See Also

`help, otd_addFailoverInstance, otd_setFailoverInstanceOrder`

2.172 otd_rotateLog

**Description**

Use this command to rotate the server log and access log files. The server saves the old log files and marks the saved files with a name that includes the date and time when they were saved.

**Syntax**

```python
otd_rotateLog(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance-name</td>
<td>Name of the node whose logs are to be rotated.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
Example

```python
props = {}
props['instance-name'] = 'otd_foo_machine1'
otd_rotateLog(props)
```

See Also

`help`

### 2.173 otd_saveConfigFile

**Description**

Use this command to upload changes to an existing configuration file or create a new one.

**Syntax**

```python
otd_saveConfigFile(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>file-path</td>
<td>Absolute path to the local file to be uploaded to the configuration directory.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>config-file</td>
<td>Name of the configuration file.</td>
<td></td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['file-path'] = '/export/config_files/foo-obj.conf'
otd_saveConfigFile(props)
```

See Also

`help, otd_createConfiguration, otd_listConfigurations, otd_deleteConfiguration, otd_copyConfiguration, otd_listConfigFiles, otd_getConfigFile, activate`

### 2.174 otd_setAccessLogBufferProperties

**Description**

Use this command to set the following properties of the access-log buffer.
## Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether the system buffers access log updates.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>buffer-size</td>
<td>Size (in bytes) of individual access log buffers.</td>
</tr>
<tr>
<td></td>
<td>Range of values: 4096 - 1048576.</td>
</tr>
<tr>
<td></td>
<td>Default: 8192.</td>
</tr>
<tr>
<td>direct-io</td>
<td>Indicates whether the file system should cache access-log writes.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>The default value, false, indicates that the file system should cache access log writes. Setting the value to true indicates that the file system should not cache access log writes. The setting is purely advisory; either the server or the operating system may choose to ignore it.</td>
</tr>
<tr>
<td>max-buffers</td>
<td>Maximum number of access log buffers per server.</td>
</tr>
<tr>
<td></td>
<td>Range of values: 1 - 65536.</td>
</tr>
<tr>
<td></td>
<td>Default: 1000.</td>
</tr>
<tr>
<td>max-buffers-per-file</td>
<td>Maximum number of access log buffers per access log file.</td>
</tr>
<tr>
<td></td>
<td>Range of values: 1 - 128.</td>
</tr>
<tr>
<td></td>
<td>Default: auto-tuned.</td>
</tr>
<tr>
<td>max-age</td>
<td>Maximum amount of time to buffer a given access log entry.</td>
</tr>
<tr>
<td></td>
<td>Range of values: an interval in seconds between 0.001 and 3600 (1 hour), inclusive.</td>
</tr>
<tr>
<td></td>
<td>Default: 1.</td>
</tr>
</tbody>
</table>

## Syntax

```python
otd_setAccessLogBufferProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set (as described above):

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

## Example

```python
props = {}
props['configuration'] = 'foo'
props['max-buffers'] = '2000'
otd_setAccessLogBufferProperties(props)
```
2.175 otd_setCacheProperties

Description
Use this command to define or change the following caching properties for a configuration:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Defines if caching is enabled or not. Range of values: true or false. Default: true.</td>
</tr>
<tr>
<td>max-entries</td>
<td>Maximum number of objects for which to cache content. Range of values: 1 - 1073741824. Default: 1024.</td>
</tr>
<tr>
<td>max-heap-object-size</td>
<td>Maximum size of response (single entry) (in bytes) to cache on the heap. If HTTP response object is bigger than max-heap-object-size, it will not be cached. Range of values: maximum size in bytes between 0 and 2147483647, inclusive. -1 indicates that there is no maximum size. Default: 524288.</td>
</tr>
<tr>
<td>max-heap-size</td>
<td>Maximum amount (in bytes) of heap to use for caching response objects. It should not be more than available memory or process address space. Range of values: maximum amount of address space in bytes between 0 and 1099511627776, inclusive. Default: 10485760.</td>
</tr>
</tbody>
</table>

Syntax

```python
otd_setCacheProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set (as described above):

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
Example

```java
props = {}
props['configuration'] = 'foo'
props['max-heap-space'] = '20971520'
otd_setCacheProperties(props)
```

See Also

`help`, `otd_getCacheProperties`

2.176 otd_setCacheRuleProperties

Description

Use this command to set the following cache rule properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>A condition is an expression which if evaluates to true, will result in the rule being executed. Conditions are constructed from literals, variables, functions and operators.</td>
<td></td>
</tr>
<tr>
<td>enabled</td>
<td>Specifies whether the caching rule is enabled.</td>
<td></td>
</tr>
<tr>
<td>max-reload-interval</td>
<td>Specifies the maximum time (in seconds) allowed between consecutive up-to-date checks.</td>
<td></td>
</tr>
<tr>
<td>min-reload-time</td>
<td>Specifies the minimum time (in seconds) allowed between consecutive up-to-date checks of a cached document.</td>
<td></td>
</tr>
<tr>
<td>last-modified-factor</td>
<td>Represents the factor used in estimating the expiry time, which defines how long a document will be up-to-date based on time it was last modified. This property is used only when the explicit age of the document is not available.</td>
<td></td>
</tr>
<tr>
<td>min-object-size</td>
<td>The maximum size, in bytes, of any document to be cached. This setting enables users to limit the maximum size of cached documents, so that no single document can take too much space. This value cannot exceed the value of max-heap-object-size.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Comments</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>max-object-size</td>
<td>Specifies the minimum size (in bytes) of any document to be cached.</td>
<td>Range of values: any positive Integer.</td>
</tr>
<tr>
<td>query-maxlen</td>
<td>Specifies the number of characters in the query string. If this property is set to 0, URIs with query strings are not cached.</td>
<td>Range of values: any positive Integer. Default: 0.</td>
</tr>
<tr>
<td>compression</td>
<td>If this property value is set to true, the document is compressed before storing in the cache.</td>
<td>Range of values: true or false. Default: false.</td>
</tr>
<tr>
<td>cache-https-response</td>
<td>If this property value is set to true, responses from the HTTPS servers are also cached.</td>
<td>Range of values: true or false. Default: false.</td>
</tr>
</tbody>
</table>

**Syntax**

```
otd_setCacheRuleProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>cache-rule</td>
<td>Name of the cache rule.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['cache-rule'] = 'cache-rule-1'
props['min-object-size'] = '512'
otd_setCacheRuleProperties(props)
```

**See Also**

`help`, `otd_setCacheProperties`, `otd_getCacheRuleProperties`
2.177 otd_setCompressionRuleProperties

Description
Use this command to set or change the following properties of a compression rule for a virtual server:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>A condition is an expression which if evaluates to true, will result in the rule being executed. Conditions are constructed from literals, variables, functions and operators.</td>
<td></td>
</tr>
<tr>
<td>insert-vary-header</td>
<td>Select to insert a vary:Accept-encoding header.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
<td></td>
</tr>
<tr>
<td>compression-level</td>
<td>Specifies the compression level for the virtual server.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Specifying 1 yields better speed and specifying 9 yields best compression.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: auto-tuned.</td>
<td></td>
</tr>
<tr>
<td>fragment-size</td>
<td>Specifies the memory fragment size (in bytes) that is used by the compression library to control the compression rate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: any positive Integer.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: 8192.</td>
<td></td>
</tr>
</tbody>
</table>

Syntax

otd_setCompressionRuleProperties(props)

The argument props is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>compression-rule</td>
<td>Name of the compression rule.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['compression-rule'] = 'compression-rule-1'
```
2.178 otd_setConfigurationAccessLogProperties

**Description**
Use this command to set access-log properties for a configuration.

**Syntax**
```
otd_setConfigurationAccessLogProperties
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>log-file</td>
<td>Path to the file where access log for this configuration will be stored.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: $DOMAIN_HOME/servers/$INSTANCE_NAME/logs/access.log.</td>
<td></td>
</tr>
<tr>
<td>format</td>
<td>A format is a string that can be used to customize the format and the fields that are logged in the access log.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: %Ses-&gt;client.ip% - %Req-&gt;vars.auth-user%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>%SYSDATE% &quot;%Req-&gt;reqpb.clf-request% %Req-&gt;srvhdrs.clf-status% %Req-&gt;srvhdrs.content-length% %Req-&gt;vars.ecid% %Req-&gt;vars.origin-server%</td>
<td></td>
</tr>
</tbody>
</table>

**Example**
```
props = {}
props['configuration'] = 'foo'
props['log-file'] = 'logs/access.log'
otd_setConfigurationAccessLogProperties(props)
```

**See Also**
`help, otd_createCompressionRule, otd_deleteCompressionRule, otd_listCompressionRules, otd_getCompressionRuleProperties`

2.179 otd_setConfigurationCrlProperties

**Description**
Use this command to define or change the following certificate revocation list (CRL) properties for a configuration:
Property | Description
--- | ---
enabled | Specifies whether the properties are enabled.
| Range of values: true or false.
| Default: true.
crl-cache-size | Size of CRL cache.
| Range of values: size in bytes between 0 and 2147483647, inclusive.
| Default: 52428800.
crl-path | Directory that contains dynamically updated CRL files.
| Range of values: pathname.
| Default: crl.

Syntax

```
otd_setConfigurationCrlProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['crl-cache-size'] = '104857600'
otd_setConfigurationCrlProperties(props)
```

See Also

`help, otd_getConfigurationCrlProperties`

2.180 otd_setConfigurationProperties

Description

Use this command to set the following configuration properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>temp-path</td>
<td>Any valid directory where the server stores its temporary files.</td>
<td>not supported on windows</td>
</tr>
</tbody>
</table>
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>webapp-firewall-ruleset</td>
<td>Wildcard pattern that matches one or more path names or a path to a file containing Web Application Firewall (WAF) rules/configurations. Multiple values (separated by commas) can also be specified.</td>
<td>Multi-valued.</td>
</tr>
<tr>
<td>default-language</td>
<td>An IANA language tag specifying the default language for messages displayed to administrators and content served to clients.</td>
<td></td>
</tr>
<tr>
<td>negotiate-client-language</td>
<td>Whether the server attempts to use the Accept-language HTTP header to negotiate the content language with clients. Range of values: true or false. Default: false.</td>
<td></td>
</tr>
<tr>
<td>fips</td>
<td>Turns on FIPS-140 mode of operation for security library. Range of values: true or false. Default: false.</td>
<td></td>
</tr>
<tr>
<td>max-fd</td>
<td>Sets the maximum value of file descriptor availability. Default: 2097152.</td>
<td></td>
</tr>
</tbody>
</table>

#### Syntax

```
otd_setConfigurationProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

#### Example

```python
props = {}
props['configuration'] = 'foo'
props['webapp-firewall-ruleset'] = 'rulesets'
otd_setConfigurationProperties(props)
```

#### See Also

`help, otd_getConfigurationProperties`

### 2.181 otd_setContentRuleProperties

#### Description

Use this command to set content rule properties.
Syntax

otd_setContentRuleProperties(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>content-rule</td>
<td>Name of the content rule.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>uri-prefix</td>
<td>URI prefix that has to be mapped to a directory.</td>
<td></td>
</tr>
<tr>
<td>directory-path</td>
<td>Absolute server path and a valid directory for storing documents.</td>
<td></td>
</tr>
<tr>
<td>index-files</td>
<td>Index files are a list of welcome files or startup pages.</td>
<td>Default: index.html,index.htm.</td>
</tr>
<tr>
<td>default-content-type</td>
<td>The type of the default content that you want to edit.</td>
<td>Default: text/plain.</td>
</tr>
<tr>
<td>allow-directory-listing</td>
<td>Enable directory listing for a directory that does not have a welcome page.</td>
<td>Range of values: true or false. Default: true.</td>
</tr>
</tbody>
</table>

Example

```python
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['content-rule'] = 'content-rule-1'
props['index-files'] = 'home.htm'
otd_setContentRuleProperties(props)
```

See Also

help, otd_getContentRuleProperties, otd_listContentRules, otd_createContentRule, otd_deleteContentRule

2.182 otd_setDnsCacheProperties

Description

Use this command to set the following Domain Name Server (DNS) cache properties:
## otd_setDnsProperties

Use this command to set the following Domain Name Server (DNS) lookup properties for a configuration:

### Property

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| enabled      | Defines whether the server does DNS lookups.  
               | Range of values: true or false.  
               | Default: true.                   |
| max-age      | Maximum amount of time (in seconds) to cache a DNS lookup result.  
               | Range of values: an interval in seconds between 0.001 (1 millisecond) and 604800 (1 week), inclusive.  
               | Default: 120.                  |
| max-entries  | Maximum number of DNS lookup results to cache.  
               | Range of values: 1 - 1048576.  
               | Default: 1024.                 |

### Syntax

```python
otd_setDnsProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| configuration | Name of the configuration.  
               | Mandatory.                              |

### Example

```python
props = {}  
props['configuration'] = 'foo'  
props['max-age'] = '240'  
props['max-entries'] = '1024'  
otd_setDnsProperties(props)
```

### See Also

`help`, `otd_getDnsCacheProperties`
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| async    | Whether the server uses its own asynchronous DNS resolver instead of the operating system's synchronous resolver.  
Range of values: true or false.  
Default: false. |
| timeout  | Timeout (in seconds) for asynchronous DNS lookups.  
Range of values: an interval in seconds between 0.001 and 3600 (1 hour), inclusive.  
Default: 12. |

#### Syntax

```python
otd_setDnsProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

#### Example

```python
props = {}
props['configuration'] = 'foo'
props['async'] = 'true'
props['timeout'] = '24'
otd_setDnsProperties(props)
```

#### See Also

`help, otd_getDnsProperties, otd_setDnsCacheProperties, otd_getDnsCacheProperties`

## 2.184 otd_setEventProperties

### Description

Use this command to set the event properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
</table>
| command      | The command that the event executes.  
Range of values: the value can be restart, reconfig, rotate-log, rotate-access-log, and update-crl, or any executable command. |
| day-of-month | Day of the month at which this event should occur.  
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>day-of-week</td>
<td>Day of the week at which this event should occur.</td>
<td></td>
</tr>
<tr>
<td>enabled</td>
<td>Whether the event is enabled at runtime.</td>
<td>Range of values: true or false. Default: true.</td>
</tr>
<tr>
<td>interval</td>
<td>Time interval at which this event should occur.</td>
<td>Range of values: an interval in seconds between 60 (1 minute) and 86400 (1 day), inclusive.</td>
</tr>
<tr>
<td>month</td>
<td>Month at which this event should occur.</td>
<td>Range of values: 1-12.</td>
</tr>
<tr>
<td>time</td>
<td>Time, for example, 12:30, when this event is to be started.</td>
<td>Range of values: the format of the time is hh:mm.</td>
</tr>
</tbody>
</table>

**Syntax**

```plaintext
otd_setEventProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>event</td>
<td>Name of the event.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
props = {}
props['configuration'] = 'foo'
props['event'] = 'bar'
props['time'] = '10:24'
otd_setEventProperties(props)
```

**See Also**

help, otd_deleteEvent, otd_listEvents, otd_getEventProperties

**2.185 otd_setEventSubscriptionProperties**

**Description**

Use this command to set the event subscription properties.
Syntax

```python
ton_d_setEventSubscriptionProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>event-subscription</td>
<td>User defined name of the event subscription.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>enabled</td>
<td>Indicates whether the event-subscription is enabled/disabled.</td>
<td>Range: true or false</td>
</tr>
<tr>
<td>URL</td>
<td>Specifies the subscription URL. If this is configured, Oracle Traffic Director publishes the notifications to this URL.</td>
<td>Range: A Valid HTTP URL.</td>
</tr>
</tbody>
</table>

**Example 2-12 Example**

```python
props = {}
props['configuration'] = 'foo'
props['event-subscription'] = 'bar'
props['enabled'] = 'false'
ton_d_setEventSubscriptionProperties(props)
```

**See Also**

help, otd_createEventSubscription, otd_deleteEventSubscription, otd_getEventSubscriptionProperties, otd_listEventSubscriptions

### 2.186 otd_setFailoverInstanceOrder

**Description**

Use this command to change the failover instance order. This command is valid only for the active-active failover type.

**Syntax**

```python
ond_setFailoverInstanceOrder(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>virtual-ip</td>
<td>Virtual IP that uniquely identifies the failure group.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>instances</td>
<td>List of instances in the order in which failover must happen.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
Example 2-13  Example

```python
props = {}
props['configuration'] = 'ha'
props['virtual-ip'] = '10.128.67.44'
props['instances'] = '1.example.com,2.example.com'
```

```python
otd_setFailoverInstanceOrder(props)
```

See Also
otd_addFailoverInstance, otd_removeFailoverInstance, otd_listFailoverInstances

## 2.187 otd_setFileCacheProperties

### Description
Sets file cache properties.

### Syntax
```
otd_setFileCacheProperties(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
</tr>
<tr>
<td>enabled</td>
<td>Indicates whether the server caches the file content and meta information.</td>
</tr>
<tr>
<td></td>
<td>Default: true</td>
</tr>
<tr>
<td>max-age</td>
<td>The maximum amount of time (in seconds) to cache the file content and/or meta information.</td>
</tr>
<tr>
<td></td>
<td>Range of values: the range of values is 0.001-3600.</td>
</tr>
<tr>
<td></td>
<td>Default: 30.</td>
</tr>
<tr>
<td>max-entries</td>
<td>The maximum number of paths for which the file content and/or meta information should be cached.</td>
</tr>
<tr>
<td></td>
<td>Range of values: 1 - 1073741824</td>
</tr>
<tr>
<td></td>
<td>Default: 1024.</td>
</tr>
<tr>
<td>max-open-files</td>
<td>The maximum number of file descriptors the file cache will keep open.</td>
</tr>
<tr>
<td></td>
<td>Range of values: 1 - 1073741824</td>
</tr>
<tr>
<td>sendfile</td>
<td>Indicates whether the server will attempt to use the operating system's <code>sendfile</code>, <code>sendfilev</code>, <code>send_file</code>, or <code>TransmitFile</code> system call.</td>
</tr>
<tr>
<td></td>
<td>The default value is true on Windows and false on other platforms.</td>
</tr>
<tr>
<td>copy-files</td>
<td>Indicates whether the server copies the cached files to a temporary directory.</td>
</tr>
<tr>
<td></td>
<td>The default value is true on Windows and false on other platforms.</td>
</tr>
<tr>
<td>copy-path</td>
<td>The name of the temporary directory that the server uses when copy-files is true.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>replacement</td>
<td>The cache entry replacement algorithm. The values can be false, lru, or lfu. Default: lru.</td>
</tr>
<tr>
<td>cache-content</td>
<td>Indicates whether the server caches the file content. Default: true</td>
</tr>
<tr>
<td>max-heap-space</td>
<td>The maximum amount (in bytes) of heap to use for caching files. Range of values: 0-9223372036854775807. Default: 10485760.</td>
</tr>
<tr>
<td>max-mmap-file-size</td>
<td>The maximum size (in bytes) of files to mmap. Range of values: 0-2147483647. Default: 0.</td>
</tr>
<tr>
<td>max-mmap-space</td>
<td>The maximum amount (in bytes) of mmap address space to use for caching files. Range of values: 0-9223372036854775807. Default: 0.</td>
</tr>
<tr>
<td>sendfile-size</td>
<td>A hint to send the file in chunks of at most this value Range of values: 0-2147483647. Default: 0.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['max-age'] = '1200'
props['otd_fileCacheProperties'] = '1200'
```

**See Also**

help, otd_getFileCacheProperties

## 2.188 otd_setHealthCheckProperties

**Description**

Use this command to set the following health-check properties for an origin server pool:
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>Health check mechanism. Range of values: http, tcp or command Default: origin server pool type</td>
</tr>
<tr>
<td>interval</td>
<td>The time interval in seconds between two health check pings. Range of values: an interval in seconds between 0.001 and 3600 (1 hour), inclusive. Default: 30</td>
</tr>
<tr>
<td>timeout</td>
<td>The timeout value in seconds for a ping request or connection. Range of values: an interval in seconds between 0.001 and 3600 (1 hour), inclusive. Default: 5</td>
</tr>
<tr>
<td>failover-threshold</td>
<td>The number of consecutive failures for marking a server down. Range of values: 1 - 256 Default: 3</td>
</tr>
<tr>
<td>request-method</td>
<td>The method used in HTTP ping requests. Range of values: OPTIONS or GET Default: OPTIONS</td>
</tr>
<tr>
<td>request-uri</td>
<td>The URI to use for HTTP health check request. Range of values: URI (virtual directory) that begins with /. Default: /</td>
</tr>
<tr>
<td>response-code-match</td>
<td>A modified regular expression to specify what response status codes are acceptable for a healthy origin server. The expression is a union of 3-character patterns that contain only digits or ‘x’, where ‘x’ stands for any digit. For example, the following 3 expressions are valid: 200, 2xx or 304, 1xx or 2xx or 3xx or 4xx. If the parameter is not specified, all codes except 5xx server errors are considered acceptable.</td>
</tr>
<tr>
<td>response-body-match</td>
<td>A regular expression used to match the HTTP response body in order to determine if the server is healthy. The origin server will be marked UP if the ping response matches the regular expression (if this parameter is specified) and the response status code is not a 5xx server error (if this parameter is not specified). If response body match is enabled, request method should be set to GET.</td>
</tr>
<tr>
<td>response-body-match-size</td>
<td>The maximum length of response body to be matched. Range of values: size in bytes between 0 and 2147483647, inclusive. Default: 2048.</td>
</tr>
<tr>
<td>dynamic-server-discovery</td>
<td>Indicates whether the server caches the file content. Range of values: true or false. Default: false.</td>
</tr>
<tr>
<td>command</td>
<td>The full path of the external health check executable.</td>
</tr>
</tbody>
</table>
Syntax

```python
otd_setHealthCheckProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['origin-server-pool'] = 'origin-server-pool-1'
props['response-body-match-size'] = '4096'
otd_setHealthCheckProperties(props)
```

**See Also**

`help, otd_getHealthCheckProperties`

### 2.189 otd_setHttpListenerProperties

**Description**

Use this command to set the following HTTP listener properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether the listener is enabled at runtime.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
<td>Default: true.</td>
</tr>
<tr>
<td>ip</td>
<td>IP address on which to listen.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: *, a hostname, or an IP address.</td>
<td></td>
</tr>
<tr>
<td>port</td>
<td>Port on which to listen.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: port number between 1 and 65535, inclusive.</td>
<td></td>
</tr>
<tr>
<td>acceptor-threads</td>
<td>Number of threads dedicated to accepting connections received by this listener.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Comments</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>server-name</td>
<td>Default server name. May include a scheme (for example, http://) prefix and port (for example,:80) suffix. Can be a hostname, fully qualified domain name, IP address, or a URL prefix that contains one. The URL prefix must not specify a path.</td>
<td></td>
</tr>
<tr>
<td>blocking-io</td>
<td>Whether the server uses blocking IO.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: false.</td>
<td></td>
</tr>
<tr>
<td>blocking-accept</td>
<td>Enables/Disables blocking of the server Listen Socket while retaining client end points as non blocking (useful when MaxProcs &gt; 1).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: false.</td>
<td></td>
</tr>
<tr>
<td>handle-protocol-mismatch</td>
<td>Range of values: true or false.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
<td></td>
</tr>
<tr>
<td>family</td>
<td>The socket family used to connect to the origin server.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: default, inet, inet6, or inet-sdp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: auto-tuned.</td>
<td></td>
</tr>
<tr>
<td>listen-queue-size</td>
<td>Maximum size of the operating system listen queue backlog.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: 1 - 1048576.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: 128.</td>
<td></td>
</tr>
<tr>
<td>receive-buffer-size</td>
<td>Size (in bytes) of the operating system socket receive buffer.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: size in bytes between 0 and 2147483647, inclusive.</td>
<td></td>
</tr>
<tr>
<td>send-buffer-size</td>
<td>Size (in bytes) of the operating system socket send buffer.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: size in bytes between 0 and 2147483647, inclusive.</td>
<td></td>
</tr>
<tr>
<td>max-requests-per-connection</td>
<td>Maximum number of keep-alive requests that will be handled per HTTP connection after which the keep-alive connection will be closed. -1 indicates no limit.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Range of values: 1-, any positive Integer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: -1.</td>
<td></td>
</tr>
<tr>
<td>default-virtual-server-name</td>
<td>Name of the virtual server that processes requests that did not match a host.</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>Description of the HTTP listener for the administrator's reference.</td>
<td></td>
</tr>
</tbody>
</table>
Syntax

```python
otd_setHttpListenerProperties(props)
```

The argument `props` is a dictionary that can contain the following properties (in addition to the properties that can be set as described above):

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>http-listener</td>
<td>Name that uniquely identifies the HTTP listener.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props["configuration"] = 'foo'
props["http-listener"] = 'http-listener-1'
props["max-requests-per-connection"] = '1024'
```

See Also

`help, otd_createHttpListener, otd_getHttpListenerProperties, otd_listHttpListeners, otd_deleteHttpListener`

### 2.190 otd_setHttpProperties

**Description**

Use this command to set the following HTTP properties for a configuration:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server-header</td>
<td>Specifies the server header information such as server software and version. Default: Oracle-Traffic-Director/&lt;version&gt;</td>
</tr>
<tr>
<td>etag</td>
<td>Indicates if the server includes an Etag header field in its responses. Range of values: true or false. Default: true.</td>
</tr>
<tr>
<td>request-header-buffer-size</td>
<td>Size (in bytes) of buffer used to read HTTP request header. Range of values: size in bytes between 0 and 2147483647, inclusive. Default: 8192.</td>
</tr>
<tr>
<td>strict-request-headers</td>
<td>Whether the server rejects certain malformed HTTP request headers Range of values: true or false. Default: false.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| websocket-strict-upgrade | Enables/Disables the strict RFC 6455 adherence during the WebSocket upgrade request.  
Range of values: true or false.  
Default: false. |
| discard-misquoted-cookies | Whether to discard misquoted cookies.  
Range of values: true or false.  
Default: true. |
| max-request-headers     | Maximum number of header fields to allow in an HTTP request header.  
Range of values: 1 - 512.  
Default: 64. |
| body-buffer-size         | Defines the maximum size of the request body content that OTD will expose via the $body variable in obj.conf.  
Range of values: size in bytes between 0 and 2147483647, inclusive  
Default: 1024. |
| output-buffer-size       | Size (in bytes) of buffer used to buffer HTTP responses.  
Range of values: size in bytes between 0 and 2147483647, inclusive  
Default: 8192. |
| max-unchunk-size         | Maximum size (in bytes) of a chunked HTTP request body the server will unchunk.  
Range of values: size in bytes between 0 and 2147483647, inclusive  
Default: 8192. |
| unchunk-timeout          | Maximum time (in seconds) the server will spend waiting for a chunked HTTP request body to arrive.  
Range of values: an interval in seconds between 0 and 3600 (1 hour), inclusive. -1 indicates no timeout.  
Default: 60. |
| io-timeout               | Maximum time (in seconds) server will wait for an individual packet.  
Range of values: an interval in seconds between 0.001 and 3600 (1 hour), inclusive.  
Default: 30. |
| request-body-timeout     | Maximum time (in seconds) server will wait for the complete HTTP request body.  
Range of values: an interval in seconds between 0 and 604800 (1 week), inclusive. -1 indicates no timeout.  
Default: -1. |
| request-header-timeout   | Maximum time (in seconds) server will wait for the complete HTTP request header.  
Range of values: an interval in seconds between 0 and 3600 (1 hour), inclusive. -1 indicates no timeout.  
Default: 30. |
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| favicon   | Whether the server replies to requests for favicon.ico with its own built in icon file.  
|           | Range of values: true or false.  
|           | Default: true.                                                               |
| ecid      | Whether the server generates/propagates Execution Context and logs it with its error log.  
|           | Range of values: true or false.  
|           | Default: true.                                                               |

### Syntax

```python
otd_setHttpProperties(props)
```

The argument `props` is a dictionary that can contain the following properties (in addition to the properties that can be set as described above):

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
</tr>
</tbody>
</table>

### Example

```python
props = {}
props['configuration'] = 'foo'
props['unchunk-timeout'] = '120'
otd_setHttpProperties(props)
```

### See Also

`help, otd_getHttpProperties`

## 2.191 otd_setHttpListenerSslProperties

### Description

Use this command to set the SSL properties for a listener. SSL is a software library establishing a secure connection between the client and server. SSL is used to implement HTTPS, the secure version of HTTP. You can set the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
</table>
| enabled  | Whether SSL/TLS is enabled at runtime.  
|           | Range of values: true or false.  
|           | Default: true.                                                               |
| client-auth | Client certificate authentication method.  
|           | Range of values: one of required, optional, or false.  
|           | Default: false.                                                              |
### Property | Description
---|---
client-auth-timeout | Timeout (in seconds) after which client authentication handshake fails. Range of values: an interval in seconds between 0.001 and 3600 (1 hour), inclusive. Default: 60.
ssl3 | Whether SSL3 connections are accepted. Range of values: true or false. Default: false.
tls10 | Whether TLS 1.0 connections are accepted. Range of values: true or false. Default: true.
tls11 | Whether TLS 1.1 connections are accepted. Range of values: true or false. Default: true.
tls12 | Whether TLS 1.2 connections are accepted. Range of values: true or false. Default: true.
ciphers | Comma separated list of ciphers that must be enabled. Range of values: one (or) more ciphers that are supported. For a list of supported ciphers, see Ciphers. Default: all supported ciphers are enabled by default.
override-cipher-order | Whether cipher order should be overridden. Setting this flag to true will make OTD select the cipher suites in the order specified in server.xml instead of the order specified in the client’s ClientHello message. Range of values: true or false. Default: false.
supported-ciphers | List of supported ciphers. This is a read-only property. Range of values: for a list of supported ciphers, see Ciphers. Default: N.A.
server-cert-alias | Comma separated list of server certificate aliases present in the keystore. Maximum of one RSA server certificate alias and one EC server certificate alias.
Note:
The command `otd_setHttpListenerSslProperties` will enable ssl implicitly if `server-cert-alias` is set for the first time. It will enable ssl, even though `ssl=enabled` is not explicitly set.

Syntax

```python
otd_setHttpListenerSslProperties(props)
```

The argument `props` is a dictionary that can contain the following properties (in addition to the properties that can be set):

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>http-listener</td>
<td>Name of the HTTP listener.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['http-listener'] = 'http-listener-1'
props['tls10'] = 'false'
otd_setHttpListenerSslProperties(props)
```

See Also

`help`, `otd_getTcpListenerSslProperties`

2.192 `otd_setHttpThreadPoolProperties`

Description

Use this command to set the thread-pool properties for a configuration. The `min-threads` and `max-threads` properties configure the threads used to process HTTP requests. You can use thread pools to allocate a certain number of threads to a specific service. By defining a pool with the maximum number of threads as 1, only one request is allowed to the specified service function.

You can set the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether the thread pool is enabled or not. Range of values: true or false. Default: true.</td>
</tr>
</tbody>
</table>
## Property Description

### queue-size
Maximum number of concurrent HTTP connections that can be queued waiting for processing.
Range of values: 1 - 1048576.
Default: auto-tuned.

### min-threads
Minimum number of HTTP request processing threads.
Range of values: 1 - 20480.
Default: auto-tuned.

### max-threads
Maximum number of HTTP request processing threads.
Range of values: 1 - 20480.
Default: auto-tuned.

### stack-size
Stack size (in bytes) for HTTP request processing threads.
Range of values: stack size in bytes between 8192 and 268435456, inclusive. 0 indicates that the platform-specific default stack size should be used.
Default: 262144.

## Syntax

```python
otd_setHttpThreadPoolProperties(props)
```

The argument `props` is a dictionary that can contain the following properties (in addition to the properties that can be set):

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

## Example

```python
props = {}
props['configuration'] = 'foo'
props['stack-size'] = '8192'
otd_setHttpThreadPoolProperties(props)
```

## See Also

`help`, `otd_getHttpThreadPoolProperties`

### 2.193 otd_setKeepaliveProperties

## Description
Sets the following keep-alive subsystem properties:
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether the server supports keep-alive connections.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>threads</td>
<td>Number of keep-alive subsystem threads.</td>
</tr>
<tr>
<td></td>
<td>Range of values: 1 - 256.</td>
</tr>
<tr>
<td></td>
<td>Default: auto-tuned.</td>
</tr>
<tr>
<td>max-connections</td>
<td>Maximum number of concurrent keep-alive connections the server will support.</td>
</tr>
<tr>
<td></td>
<td>Range of values: 1 - 1048576.</td>
</tr>
<tr>
<td></td>
<td>Default: auto-tuned.</td>
</tr>
<tr>
<td>timeout</td>
<td>Timeout (in seconds) after which inactive keep-alive connection may be closed.</td>
</tr>
<tr>
<td></td>
<td>Range of values: an interval in seconds between 0.001 (1 millisecond) and 3600 (1 hour), inclusive.</td>
</tr>
<tr>
<td></td>
<td>-1 indicates no timeout.</td>
</tr>
<tr>
<td></td>
<td>Default: 30.</td>
</tr>
<tr>
<td>poll-interval</td>
<td>Interval (in seconds) between polls.</td>
</tr>
<tr>
<td></td>
<td>Range of values: an interval in seconds between 0.001 and 1, inclusive.</td>
</tr>
<tr>
<td></td>
<td>Default: 0.001.</td>
</tr>
</tbody>
</table>

### Syntax

```python
otd_setKeepaliveProperties(props)
```

The argument `props` is a dictionary that can contain the following properties (in addition to the properties that can be set):

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
</tr>
<tr>
<td></td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

### Example

```python
cfg = {}
cfg['configuration'] = 'foo'
cfg['threads'] = 128
otd_setKeepaliveProperties(cfg)
```

### See Also

`help, otd_getKeepaliveProperties`

## 2.194 otd_setLogProperties

### Description

Use this command to set the following log properties:
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log-stdout</td>
<td>Whether the server logs data that applications write to stdout.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>log-stderr</td>
<td>Whether the server logs data that applications write to stderr.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>log-virtual-server-name</td>
<td>Whether the server includes the virtual server name in log messages.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: false.</td>
</tr>
<tr>
<td>create-console</td>
<td>(Windows only) Whether the server creates a console window.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: false.</td>
</tr>
<tr>
<td>log-to-console</td>
<td>Whether the server writes log messages to the console.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>log-to-syslog</td>
<td>Whether the server writes log messages to syslog (Unix) or the Event Viewer (Windows).</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: false.</td>
</tr>
<tr>
<td>archive-command</td>
<td>Command executed after the server rotates a log file. The file name of the log file, after rotation, is passed as an argument to the archive command.</td>
</tr>
<tr>
<td>log-level</td>
<td>Log verbosity for the server as a whole.</td>
</tr>
<tr>
<td></td>
<td>Range of values: valid log levels are INCIDENT_ERROR:1, ERROR:1, ERROR:16, ERROR:32, WARNING:1, NOTIFICATION:1, TRACE:1, TRACE:16, and TRACE:32. TRACE:32 (finest) is the most verbose while INCIDENT_ERROR:1 (catastrophe) is the least verbose.</td>
</tr>
<tr>
<td></td>
<td>Default: NOTIFICATION:1.</td>
</tr>
<tr>
<td>log-file</td>
<td>Log file for the server as a whole.</td>
</tr>
<tr>
<td></td>
<td>Default: $DOMAIN_HOME/servers/$INSTANCE_NAME/logs/server.log.</td>
</tr>
</tbody>
</table>

**Syntax**

```
otd_setLogProperties(props)
```

The argument `props` is a dictionary that can contain the following properties (in addition to the properties that can be set):
## 2.195 otd_setOriginServerPoolSslProperties

**Description**

Use this command to set the SSL properties of the origin server pool.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether SSL/TLS is enabled at runtime.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>ssl3</td>
<td>Whether SSL3 connections are accepted.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: false.</td>
</tr>
<tr>
<td>tls10</td>
<td>Whether TLS 1.0 connections are accepted.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>tls11</td>
<td>Whether TLS 1.1 connections are accepted.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>tls12</td>
<td>Whether TLS 1.2 connections are accepted.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>validate-server-cert</td>
<td>Only applies to outgoing connections. Validate SSL certificate hostname on/off flag.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['log-level'] = 'TRACE:32'
otd_setLogProperties(props)
```

**See Also**

help, otd_getLogProperties, displayLogs
## otd_setOriginServerPoolProperties

Use this command to set the following origin-server pool properties:

### Syntax

```
otd_setOriginServerPoolSslProperties(props)
```

The argument `props` is a dictionary that can contain the following properties (in addition to the properties that can be set):

### Example

```python
props = {}
props['configuration'] = 'foo'
props['origin-server-pool'] = 'origin-server-pool-1'
props['tls10'] = 'false'
otd_setOriginServerPoolSslProperties(props)
```

### See Also

help

2.196 otd_setOriginServerPoolProperties

### Description

Use this command to set the following origin-server pool properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

### Property Description

- **ciphers**: Comma separated list of ciphers that must be enabled.
  Range of values: one (or) more ciphers that are supported. For a list of supported ciphers, see Ciphers.
  Default: all supported ciphers are enabled by default.

- **supported-ciphers**: List of supported ciphers. This is a read-only property.
  Range of values: for a list of supported ciphers, see Ciphers.
  Default: n/a

- **client-cert-alias**: A valid client certificate alias present in the keystore.
  Maximum of one RSA server certificate alias and one EC server certificate alias.
## Property Description Comments

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>family</td>
<td>The socket family used to connect to servers in this pool. Range of values: default, inet, inet6, or inet-sdp. Default: auto-tuned.</td>
<td></td>
</tr>
<tr>
<td>load-distribution</td>
<td>Algorithm that is used for load distribution of this server pool. Range of values: round-robin, least-connection-count, or least-response-time. Default: least-connection-count.</td>
<td></td>
</tr>
<tr>
<td>queue-timeout</td>
<td>Timeout (in seconds) for which the request waits in the queue for a connection to an origin-server. After the timeout, request is rejected. Range of values: 0.001 - 3600. Default: 30.</td>
<td></td>
</tr>
<tr>
<td>proxy-server</td>
<td>Name of the proxy-server in the form of host:port.</td>
<td></td>
</tr>
</tbody>
</table>

## Syntax

```
otd_setOriginServerPoolProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set (as described above):

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

## Example

```python
props = {}
props['configuration'] = 'foo'
props['origin-server-pool'] = 'origin-server-pool-1'
props['load-distribution'] = 'least-connection-count'
otd_setOriginServerPoolProperties(props)
```

## See Also

help, otd_getOriginServerPoolProperties, otd_listOriginServerPools, otd_deleteOriginServerPool, otd_createOriginServerPool
### 2.197 otd_setOriginServerProperties

**Description**

Use this command to set the following properties of an origin-server:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>Mode of this origin server. Range of values: enabled, disabled, draining. Default: enabled.</td>
</tr>
<tr>
<td>backup</td>
<td>The parameter specifies if the origin server is a backup server. Range of values: true or false. Default: false.</td>
</tr>
<tr>
<td>max-connections</td>
<td>The maximum number of concurrent connections to a server. -1 indicates that there is no maximum. Range of values: -1, 1 - 1048576. Default: -1.</td>
</tr>
<tr>
<td>ramp-up-time</td>
<td>The time in seconds to ramp the sending rate up to the capacity of a newly up origin server. If the parameter is not specified, request rate accelerating will not be activated for the server. Range of values: an interval in seconds between 0.001 and 3600 (1 hour), inclusive. Default: 0.001.</td>
</tr>
<tr>
<td>max-requests-per-connection</td>
<td>Maximum limit on times a connection to origin server can be reused for different requests. -1 indicates there is no limit. Range of values: -1, 1 - 2147483647. Default: -1.</td>
</tr>
<tr>
<td>max-request-bps</td>
<td>Total bandwidth limit in byte/second enforced on request. Range of values: 0 - 1099511627776. Default: 0.</td>
</tr>
<tr>
<td>max-response-bps</td>
<td>Total bandwidth limit in byte/second enforced on response. Range of values: 0 - 1099511627776. Default: 0.</td>
</tr>
<tr>
<td>bandwidth-queue-timeout</td>
<td>Request is aborted when it had to wait in the queue for bandwidth for this much time in second. Range of values: 0 - 86400. Default: 60.</td>
</tr>
</tbody>
</table>
Syntax

```python
otd_setOriginServerProperties(props)
```

The argument `props` is a dictionary that can contain the following properties (in addition to the properties that can be set):

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>host</td>
<td>IP address/Host name of the origin server.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>port</td>
<td>Port number of the origin server.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['origin-server-pool'] = 'origin-server-pool-1'
props['host'] = 'www.example.com'
props['port'] = '12345'
props['ramp-up-time'] = '1200'
otd_setOriginServerProperties(props)
```

**See Also**

`help`, `otd_getOriginServerProperties`, `otd_listOriginServers`, `otd_deleteOriginServer`, `otd_createOriginServer`

### 2.198 otd_setPartitionAccessLogProperties

**Description**

Use this command to set the access-log properties for a partition.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log-file</td>
<td>Path to the file where access logs for this partition will be stored. Default: $DOMAIN_HOME/servers/$INSTANCE_NAME/logs/$PARTITION_NAME.log</td>
</tr>
<tr>
<td>format</td>
<td>A format is a string that can be used to customize the format and the fields that are logged in the partition access log. Default: %Ses-&gt;client.ip% - %Req-&gt;vars.auth-user% %SYSDATE% &quot;%Req-&gt;reqpb.clf-request%&quot; %Req-&gt;srvhdrs.clf-status% %Req-&gt;srvhdrs.content-length% %Req-&gt;vars.ecid% %Req-&gt;vars.origin-server%</td>
</tr>
</tbody>
</table>
**Syntax**

```bash
otd_setPartitionAccessLogProperties (props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>configuration</code></td>
<td>Name of the configuration. This must be the name of the configuration that is specified while registering the Oracle Traffic Director runtime with the Lifecycle Manager.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td><code>partition</code></td>
<td>Name of the partition.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'mt'
props['partition'] = 'WLSPartition'
props['log-file'] = 'logs/WLSPartition.log'
```  

**See Also**

`help, otd_listPartitions, otd_getPartitionAccessLogProperties`

### 2.199 otd_setRequestLimitProperties

**Description**

Use this command to set the following request limit properties for a virtual server:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>condition</code></td>
<td>A condition is an expression which if evaluates to true, will result in the rule being executed. Conditions are constructed from literals, variables, functions and operators.</td>
<td></td>
</tr>
<tr>
<td><code>max-rps</code></td>
<td>Maximum number of requests that the virtual server can receive per second. Range of values: any positive Integer.</td>
<td></td>
</tr>
<tr>
<td><code>max-connections</code></td>
<td>Maximum number of concurrent matching connections. Range of values: any positive Integer.</td>
<td></td>
</tr>
<tr>
<td><code>queue-size</code></td>
<td>Maximum number of requests to be queued in the bucket. Range of values: any positive Integer. Default: 0.</td>
<td></td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>Request is aborted when it had to wait in the queue for this much time in second. Range of values: 1 - 86400. Default: 60.</td>
<td></td>
</tr>
</tbody>
</table>
### Property Description Comments

**error-code**

HTTP status code to return for blocked requests.
Range of values: 400 - 599.
Default: 503.

**monitor-attribute**

Request attribute to monitor.

---

**Syntax**

```python
otd_setRequestLimitProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set (as described above):

---

**Property**

- **configuration**
  
  Name of the configuration.
  
  Mandatory.

- **virtual-server**
  
  Name of the virtual server.
  
  Mandatory.

- **request-limit**
  
  Name of the request limit rule.
  
  Mandatory.

---

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['request-limit'] = 'request-limit-1'
props['max-connections'] = '1024'
отd_setRequestLimitProperties(props)
```

**See Also**

- help, otd_getRequestLimitProperties, otd_listRequestLimits, otd_deleteRequestLimit, otd_createRequestLimit

---

2.200 **otd_setRouteProperties**

**Description**

Use this command to set the following route properties for a virtual server.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>A condition is an expression which if evaluates to true, will result in the rule being executed. Conditions are constructed from literals, variables, functions and operators.</td>
<td>condition cannot be set for the uri-prefix based routes.</td>
</tr>
<tr>
<td>uri-prefix</td>
<td>A uri-prefix is a URI path with wildcard patterns. If a request URI matches with the uri-prefix then the rule will be executed.</td>
<td>uri-prefix cannot be set for the condition based routes.</td>
</tr>
<tr>
<td>origin-server-pool</td>
<td>Name of the origin server pool for this route.</td>
<td></td>
</tr>
<tr>
<td>offline-check-interval</td>
<td>Specifies the offline check interval.</td>
<td></td>
</tr>
<tr>
<td>server</td>
<td>Specifies the server name.</td>
<td></td>
</tr>
<tr>
<td>sticky-cookie</td>
<td>Name of the cookie that causes subsequent requests to stick to a particular origin server. Default: JSESSIONID.</td>
<td></td>
</tr>
<tr>
<td>sticky-param</td>
<td>Name of a URI parameter to inspect for route information. When the URI parameter is present in a request URI and its value contains a colon :, followed by a route ID, the request will 'stick' to the origin server identified by that route ID. Default: jsessionid.</td>
<td></td>
</tr>
<tr>
<td>route-header</td>
<td>Name of the HTTP request header that is used to communicate route IDs to the origin servers. Default: Proxy-jroute.</td>
<td></td>
</tr>
<tr>
<td>route-cookie</td>
<td>Name of the cookie generated by the server when it encounters a sticky-cookie cookie in a response. The route-cookie parameter stores the route ID that enables the server to direct subsequent requests back to the same origin server. Default: ORA_OTD_JROUTE.</td>
<td></td>
</tr>
<tr>
<td>rewrite-headers</td>
<td>List of HTTP request headers separated by commas.</td>
<td></td>
</tr>
<tr>
<td>use-keep-alive</td>
<td>Whether the HTTP client can use existing persistent connections for all types of requests. Range of values: true of false. Default: true.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Comments</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>keep-alive-timeout</td>
<td>Maximum number (in seconds) to retain persistent connectivity.</td>
<td>Range of values: any positive Integer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: 29.</td>
</tr>
<tr>
<td>timeout</td>
<td>Maximum number (in seconds) that a connection can be in an idle state.</td>
<td>Range of values: any positive Integer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: 300.</td>
</tr>
<tr>
<td>always-use-keep-alive</td>
<td>Whether the HTTP client can reuse existing connections for all types of requests.</td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: false.</td>
</tr>
<tr>
<td>protocol</td>
<td>Specifies the HTTP protocol version.</td>
<td></td>
</tr>
<tr>
<td>proxy-agent</td>
<td>Whether the proxy server product name and version has to be forwarded to the origin servers.</td>
<td></td>
</tr>
<tr>
<td>from</td>
<td>URI prefix to map. The prefix must not contain trailing slashes.</td>
<td></td>
</tr>
<tr>
<td>to</td>
<td>URL prefix to which the request should be mapped. The prefix must not contain trailing slashes.</td>
<td></td>
</tr>
<tr>
<td>log-headers</td>
<td>If true, the HTTP request and response headers for all connections with origin servers will be logged in the server log file.</td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: false.</td>
</tr>
<tr>
<td>websocket-upgrade-enabled</td>
<td>Specifies whether standard HTTP(S) connections should be upgraded to bi-directional, full-duplex WebSocket connections.</td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>websocket-idle-timeout</td>
<td>The maximum number of seconds a connection can be idle. If no value is specified, then the timeout from the TCP connection thread pool (300 seconds) is used.</td>
<td>Range of values: -1 or 0 - 3600.</td>
</tr>
<tr>
<td>buffer-size</td>
<td>The size of the buffer that is used by the server to store data before it is sent to the client.</td>
<td>Range of values: any positive Integer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default: 16384.</td>
</tr>
</tbody>
</table>
**otd_setRouteProperties**

**Property**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>The priority of the request. Range of values: high, normal, low. Default: normal.</td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

```python
otd_setRouteProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set (as described above):

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>route</td>
<td>Name of the route.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['route'] = 'route-1'
props['websocket-idle-timeout'] = '1200'
otd_setRouteProperties(props)
```

**See Also**

`help`, `otd_getRouteProperties`, `otd_listProxyInfo`, `otd_forwardProxyInfo`, `otd_blockProxyInfo`, `otd_listRoutes`, `otd_deleteRoute`, `otd_createRoute`

**2.201 otd_setSnmpProperties**

**Description**

Use this command to enable and define these settings for the SNMP subagents.

**Property**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether SNMP is enabled. Range of values: true or false. Default: true.</td>
</tr>
</tbody>
</table>
### Syntax

`otd_setSnmpProperties(props)`

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set (as described above):

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

### Example

```python
props = {}
props['configuration'] = 'foo'
props['organization'] = 'bar'
otd_setSnmpProperties(props)
```

### See Also

`help`, `otd_getSnmpProperties`, `otd_startSnmpSubAgent`, `otd_stopSnmpSubAgent`

## 2.202 `otd_setSslSessionCacheProperties`

### Description

Use this command to set the SSL session cache properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether the server caches SSL/TLS sessions. Range of values: true or false. Default: true.</td>
</tr>
</tbody>
</table>
## otd_setSslSessionCacheProperties

Use this command to set these properties of the SSL/TLS session cache subsystem.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max-entries</td>
<td>Maximum number of SSL/TLS sessions the server will cache.</td>
</tr>
<tr>
<td></td>
<td>Range of values: 1 - 524288.</td>
</tr>
<tr>
<td></td>
<td>Default: 10000.</td>
</tr>
<tr>
<td>max-ssl3-tls-session-age</td>
<td>Maximum amount of time to cache an SSL3/TLS session.</td>
</tr>
<tr>
<td></td>
<td>Range of values: 1 - 86400.</td>
</tr>
<tr>
<td></td>
<td>Default: 86400.</td>
</tr>
</tbody>
</table>

### Syntax

```
otd_setSslSessionCacheProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set (as described above):

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

### Example

```python
props = {}
props['configuration'] = 'foo'
props['max-entries'] = '20000'
props['max-ssl3-tls-session-age'] = '86400'
.otd_setSslSessionCacheProperties(props)
```

### See Also

```
help, otd_getSslSessionCacheProperties
```

## 2.203 otd_setStatsProperties

Use this command to set these properties of the statistics collection subsystem.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether the server collects statistics at runtime.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>interval</td>
<td>Interval (in seconds) at which statistics are updated.</td>
</tr>
<tr>
<td></td>
<td>Range of values: an interval in seconds between 0.001 and 3600 (1 hour), inclusive.</td>
</tr>
<tr>
<td></td>
<td>Default: 5.</td>
</tr>
</tbody>
</table>
**otd_setStatsProperties**

### Description

Whether performance buckets, used to track NSAPI function execution time, are enabled. Range of values: true or false. Default: true.

### Syntax

```python
otd_setStatsProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>profiling</td>
<td>Whether performance buckets, used to track NSAPI function execution time, are enabled. Range of values: true or false. Default: true.</td>
<td></td>
</tr>
</tbody>
</table>

### Example

```python
props = {}
props['configuration'] = 'foo'
props['interval'] = '10'
otd_setStatsProperties(props)
```

### See Also

*help, otd_getStatsProperties*

### 2.204 otd_setStatusListenerSslProperties

#### Description

Use this command to change the SSL properties of the Status Listener.

#### Syntax

```python
otd_setStatusListenerSslProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>enabled</td>
<td>Specifies if SSL is enabled for status listener. Values: True or false. Default: True</td>
<td></td>
</tr>
</tbody>
</table>
**Example 2-14  Example**

```python
props = {}
props['configuration'] = 'foo'
props['enabled'] = 'false'
otd_setStatusListenerSslProperties(props)
```

**See Also**

otd_enableStatusListener, otd_disableStatusListener, otd_getStatusListenerProperties, otd_getStatusListenerSslProperties

### 2.205 otd_setTcpAccessLogProperties

**Description**

Use this command to set the properties of the TCP access log.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>log-file</td>
<td>Path to the file where the TCP access log for this configuration will be stored. Default: <code>$DOMAIN_HOME/servers/$INSTANCE_NAME/logs/tcp-access.log</code></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

```python
otd_setTcpAccessLogProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['log-file'] = 'logs/tcp-access.log'
otd_setTcpAccessLogProperties(props)
```

**See Also**

help, otd_getTcpAccessLogProperties
2.206 otd_setTcpListenerProperties

Description

Use this command to set the TCP listener properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>Port on which to listen. Range of values: port number between 1 and 65535, inclusive.</td>
</tr>
</tbody>
</table>
| ip             | IP address on which to listen. Range of values: *, a hostname, or an IP address. Default: *.
| acceptor-threads | Acceptor threads for this listening end point. Range of values: 1 - 128. Default: auto-tuned. |
| enabled        | Whether the instance is enabled. Range of values: true or false. Default: true. |
| description    | Description of the TCP listener for the administrator’s reference. |
| receive-buffer-size | Size (in bytes) of the operating system socket receive buffer. Range of values: size in bytes between 0 and 2147483647, inclusive. |
| send-buffer-size  | Size (in bytes) of the operating system socket send buffer. Range of values: size in bytes between 0 and 2147483647, inclusive. |
| blocking-accept | Enables/Disables blocking of the server Listen Socket while retaining client end points as non blocking (useful when MaxProcs > 1). Range of values: true or false. Default: false. |
| tcp-proxy      | Name that identifies the exposed TCP service. Name can consist of one or more characters, whitespace is not permitted. |

Syntax

```python
otd_setTcpListenerProperties(props)
```
The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>tcp-listener</td>
<td>Name of the TCP listener.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
props['tcp-listener'] = 'tcp-listener-1'
props['listen-queue-size'] = '256'

otd_setTcpListenerProperties(props)
```

**See Also**

`help, otd_createTcpListener, otd_deleteTcpListener, otd_listTcpListeners, otd_getTcpListenerProperties`

### 2.207 otd_setTcpListenerSslProperties

**Description**

Use this command to set the Secure Sockets Layer (SSL) properties for a TCP listener:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether SSL/TLS is enabled at runtime. Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>client-auth</td>
<td>Client certificate authentication method. Range of values: one of required,</td>
</tr>
<tr>
<td></td>
<td>optional, or false.</td>
</tr>
<tr>
<td></td>
<td>Default: false.</td>
</tr>
<tr>
<td>client-auth-timeout</td>
<td>Timeout (in seconds) after which client authentication handshake fails. Range of values: an interval in seconds between 0.001 and 3600 (1 hour), inclusive.</td>
</tr>
<tr>
<td></td>
<td>Default: 60.</td>
</tr>
<tr>
<td>max-client-auth-data</td>
<td>Maximum amount of application-level data to buffer during a client authentication handshake. Range of values: size in bytes between 0 and 2147483647, inclusive.</td>
</tr>
<tr>
<td></td>
<td>Default: 1048576.</td>
</tr>
<tr>
<td>ssl3</td>
<td>Whether SSL3 connections are accepted. Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: false.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tls10</td>
<td>Whether TLS 1.0 connections are accepted. Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>tls11</td>
<td>Whether TLS 1.1 connections are accepted. Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>tls12</td>
<td>Whether TLS 1.2 connections are accepted. Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>ciphers</td>
<td>Comma separated list of ciphers that must be enabled. Range of values: one</td>
</tr>
<tr>
<td></td>
<td>or more ciphers that are supported. For a list of supported ciphers, see</td>
</tr>
<tr>
<td></td>
<td>Ciphers.</td>
</tr>
<tr>
<td></td>
<td>Default: all supported ciphers are enabled by default.</td>
</tr>
<tr>
<td>override-cipher-order</td>
<td>Whether cipher order should be overridden. Setting this flag to true will</td>
</tr>
<tr>
<td></td>
<td>make OTD select the cipher suites in the order specified in server.xml</td>
</tr>
<tr>
<td></td>
<td>instead of the order specified in the client's ClientHello message.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: false.</td>
</tr>
<tr>
<td>supported-ciphers</td>
<td>List of supported ciphers. This is a read-only property. Range of values: for a list of supported ciphers, see Ciphers.</td>
</tr>
<tr>
<td></td>
<td>Default: N.A.</td>
</tr>
<tr>
<td>server-cert-alias</td>
<td>Comma separated list of server certificate aliases present in the keystore.</td>
</tr>
<tr>
<td></td>
<td>Maximum of one RSA server certificate alias and one EC server certificate alias.</td>
</tr>
</tbody>
</table>

**Syntax**

```python
otd_setTcpListenerProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory</td>
</tr>
<tr>
<td>tcp-listener</td>
<td>Name of the TCP listener.</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

**Example**

```python
props = {}
props['configuration'] = 'foo'
```
otd_setTcpProxyProperties

- `props['tcp-listener'] = 'tcp-listener-1'`
- `props['tls10'] = 'false'`
- `otd_setTcpListenerSslProperties(props)`

**See Also**
- `help`, `otd_getTcpListenerSslProperties`

### 2.208 otd_setTcpProxyProperties

**Description**

Use this command to set the following properties of the TCP proxy for a configuration:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether the TCP service is enabled.</td>
</tr>
<tr>
<td></td>
<td>Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>session-idle-timeout</td>
<td>Maximum timeout in seconds for load balancer to wait for receiving/sending data in the session.</td>
</tr>
<tr>
<td></td>
<td>Range of values: an interval in seconds between 0.001 and 3600 (1 hour), inclusive.</td>
</tr>
<tr>
<td>origin-server-pool-name</td>
<td>Name of an existing server pool that provides the TCP service.</td>
</tr>
</tbody>
</table>

**Syntax**

```
otd_setTcpProxyProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set (as described above):

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>tcp-proxy</td>
<td>Name of the TCP proxy.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

**Example**

```
props = {}
props['configuration'] = 'foo'
props['tcp-proxy'] = 'bar'
props['session-idle-timeout'] = '1200'
otd_setTcpProxyProperties(props)
```

**See Also**
- `help`, `otd_createTcpProxy`, `otd_deleteTcpProxy`, `otd_listTcpProxies`, `otd_getTcpProxyProperties`
2.209 otd_setTcpThreadPoolProperties

**Description**

Use this command to set the thread-pool properties of a configuration. The properties configure the threads used to proxy data for upgraded WebSocket connections and generic TCP connections. You can use TCP thread pools to allocate a certain number of threads to a specific service.

You can set the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether the pool is enabled or not. Range of values: true or false. Default: true.</td>
</tr>
<tr>
<td>max-connections</td>
<td>Maximum number of connection pairs the server will support. Range of values: 1 - 1048576. Default: auto-tuned.</td>
</tr>
<tr>
<td>timeout</td>
<td>Idle timeout (in seconds) after which connection pairs will be closed. Value will be overridden by the TCP or WebSocket subsystem. Range of values: an interval in seconds between 0.001 (1 millisecond) and 3600 (1 hour), inclusive. -1 indicates no timeout. Default: 300.</td>
</tr>
<tr>
<td>poll-interval</td>
<td>Interval (in seconds) between polls. Range of values: an interval in seconds between 0.001 and 1, inclusive. Default: 0.01.</td>
</tr>
<tr>
<td>stack-size</td>
<td>Stack size in bytes for each thread. Range of values: stack size in bytes between 8192 and 268435456, inclusive. 0 indicates that the platform-specific default stack size should be used. Default: 32768.</td>
</tr>
</tbody>
</table>

**Syntax**

```
otd_setTcpThreadPoolProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:
### Example

```python
props = {}
props['configuration'] = 'foo'
props['stack-size'] = '8192'
otd_setTcpThreadPoolProperties(props)
```

### See Also

`help, otd_getTcpThreadPoolProperties`

### 2.210 otd_setVirtualServerProperties

#### Description

Use this command to set the properties of a virtual-server.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether the virtual server is enabled at runtime. Range of values: true or false. Default: true.</td>
</tr>
<tr>
<td>canonical-server-name</td>
<td>Canonical hostname of the virtual server (requests using a different hostname will be redirected to this hostname). Can be a Hostname, fully qualified domain name, ip address, or a url prefix that contains one. the url prefix must not specify a path.</td>
</tr>
<tr>
<td>log-file</td>
<td>Log file for the virtual server.</td>
</tr>
<tr>
<td>http-listener-name</td>
<td>Name of an HTTP listener associated with one or more of the virtual server's host hostnames. Multiple comma separated values can be specified.</td>
</tr>
<tr>
<td>host</td>
<td>Hostname the virtual server services. Multiple comma separated values can be specified where each value can be a wildcard pattern that matches one or more hostnames.</td>
</tr>
<tr>
<td>default-language</td>
<td>An IANA language tag specifying the default language for messages displayed to administrators and content served to clients.</td>
</tr>
<tr>
<td>negotiate-client-language</td>
<td>Whether the server attempts to use the Accept-language HTTP header to negotiate the content language with clients. Range of values: true or false. Default: false.</td>
</tr>
</tbody>
</table>
Syntax

```
otd_setVirtualServerProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```
props = {}
props["configuration"] = 'foo'
props["virtual-server"] = 'bar'
props["http-listener-name"] = 'http-listener-1'
otd_setVirtualServerProperties(props)
```

See Also

help, otd_deleteVirtualServer, otd_getVirtualServerProperties, otd_listVirtualServers, otd_copyVirtualServer

2.211 otd_setVirtualServerSslProperties

**Description**

Use this command to set the SSL properties for a virtual server.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>Whether SSL/TLS is enabled at runtime. Range of values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
</tr>
<tr>
<td>client-auth</td>
<td>Client certificate authentication method. Range of values: one of required,</td>
</tr>
<tr>
<td></td>
<td>optional, or false.</td>
</tr>
<tr>
<td></td>
<td>Default: false.</td>
</tr>
<tr>
<td>client-auth-timeout</td>
<td>Timeout (in seconds) after which client authentication handshake fails.</td>
</tr>
<tr>
<td></td>
<td>Range of values: an interval in seconds between 0.001 and 3600 (1 hour),</td>
</tr>
<tr>
<td></td>
<td>inclusive. Default: 60.</td>
</tr>
<tr>
<td>max-client-auth-data</td>
<td>Maximum amount of application-level data to buffer during a client</td>
</tr>
<tr>
<td></td>
<td>authentication handshake. Maximum amount of application-level data to buffer</td>
</tr>
<tr>
<td></td>
<td>during a client authentication handshake. Range of values: size in bytes</td>
</tr>
<tr>
<td></td>
<td>between 0 and 2147483647, inclusive. Default: 1048576.</td>
</tr>
</tbody>
</table>

Oracle Traffic Director WLST Commands 2-177
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssl3</td>
<td>Whether SSL3 connections are accepted. Range of values: true or false.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: false.</td>
<td></td>
</tr>
<tr>
<td>tls10</td>
<td>Whether TLS 1.0 connections are accepted. Range of values: true or false.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
<td></td>
</tr>
<tr>
<td>tls11</td>
<td>Whether TLS 1.1 connections are accepted. Range of values: true or false.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
<td></td>
</tr>
<tr>
<td>tls12</td>
<td>Whether TLS 1.2 connections are accepted. Range of values: true or false.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: true.</td>
<td></td>
</tr>
<tr>
<td>ciphers</td>
<td>Comma separated list of ciphers that must be enabled. Range of values: one (or) more ciphers that are supported. For a list of supported ciphers, see <a href="#">Ciphers</a>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default: all supported ciphers are enabled by default.</td>
<td></td>
</tr>
<tr>
<td>override-cipher-order</td>
<td>Whether cipher order should be overridden. Setting this flag to true will make OTD select the cipher suites in the order specified in server.xml instead of the order specified in the client's ClientHello message. Range of values: true or false. Default: false.</td>
<td></td>
</tr>
<tr>
<td>supported-ciphers</td>
<td>List of supported ciphers. This is a read-only property. Range of values: for a list of supported ciphers, see <a href="#">Ciphers</a>. Default: N.A.</td>
<td></td>
</tr>
<tr>
<td>server-cert-alias</td>
<td>Comma separated list of server certificate aliases present in the keystore. Maximum of one RSA server certificate alias and one EC server certificate alias.</td>
<td></td>
</tr>
</tbody>
</table>

### Syntax

```python
otd_setVirtualServerSslProperties(props)
```

The argument `props` is a dictionary that can contain the following properties in addition to those properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
Example

```python
props = {}
props['configuration'] = 'foo'
props['virtual-server'] = 'bar'
props['tls10'] = 'false'
otd_setVirtualServerSslProperties(props)
```

See Also

help, otd_getVirtualServerSslProperties

2.212 otd_setWalletPassword

Description

Sets the password on a wallet.

Syntax

```python
otd_setWalletPassword(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>password</td>
<td>Password consisting of a minimum length of 8 characters and contain alphabetic characters combined with numbers or special characters.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {}
props['configuration'] = 'foo'
props['password'] = 'barBazqux#'
otd_setWalletPassword(props)
```

See Also

help, exportKeyStoreCertificateRequest, otd_listCertificates, importKeyStoreCertificate, getKeyStoreCertificates, generateKeyPair

2.213 otd_setWebappFirewallProperties

Description

Use this command to set the following properties of a web application firewall:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ruleset</td>
<td>Path to a file containing Web Application Firewall (WAF)</td>
<td>Multi-valued.</td>
</tr>
<tr>
<td>rules</td>
<td>rules/configuration</td>
<td></td>
</tr>
</tbody>
</table>
otd_setWebappFirewallProperties(props)

The argument `props` is a dictionary that must contain the following keys in addition to the properties that can be set:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-server</td>
<td>Name of the virtual server.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example

```python
props = {
    'configuration': 'foo',
    'virtual-server': 'bar',
    'ruleset': 'rulesets'
}
```

```
otd_setWebappFirewallProperties(props)
```

See Also

`help, otd_createVirtualServer, otd_setVirtualServerProperties, otd_listVirtualServers, otd_copyVirtualServer, otd_deleteVirtualServer, otd_getVirtualServerProperties`

2.214 otd_startFailover

Description

Use this command to start the failover daemon on the local machine. Since the failover daemon needs to run as root, you should execute this command should with `sudo` privileges on the host on which the primary/backup instance of the failover group is running to start the failover on the instance.

This command can only be run in offline mode.

Syntax

```python
otd_startFailover(props)
```

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain-home</td>
<td>Path to the directory that contains the Oracle Traffic Director domain.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>instance-name</td>
<td>Name of the primary/backup Oracle Traffic Director instance which is part of the failover group.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
### otd_startSnmpSubAgent

#### Description
Use this command to start the Oracle Traffic Director Simple Network Management Protocol (SNMP) sub-agent on the specified machine.

#### Syntax
`otd_startSnmpSubAgent(props)`

The argument `props` is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>machine-name</td>
<td>Name specified while creating the machine in the Oracle WebLogic Server console, corresponding to the host name of the machine on which the Oracle Traffic Director instance is running.</td>
<td>Mandatory for Online, not valid for Offline.</td>
</tr>
<tr>
<td>domain-home</td>
<td>Path to the directory that contains the Oracle Traffic Director domain.</td>
<td>Mandatory for Offline, not valid for Online.</td>
</tr>
</tbody>
</table>

#### Example

```
# Online
props = {}
props['machine-name'] = 'abc123.example.com'
отd_startSnmpSubAgent(props)

# Offline
props = {}
props['domain-home'] = '/export/domains/otd_domain'
отd_startSnmpSubAgent(props)
```

---

### log-verbose

Whether keepalived should be started in verbose log level mode.

Default: false.

#### Example

```
props = {}
props['domain-home'] = '/export/domains/otd_domain'
props['instance-name'] = 'otd_abci23.example.com'
отd_startFailover(props)
```
See Also
help, otd_stopSnmpSubAgent, otd_setSnmpProperties, otd_getSnmpProperties

2.216 otd_stopFailover

Description
Use this command to stop the failover daemon on the local machine. Since the failover daemon needs to run as root, execute this command with sudo privileges on the host on which the primary/backup instance of the failover group is running to stop the failover on the instance.

This command can only be run in offline mode.

Syntax
otd_stopFailover(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain-home</td>
<td>Path to the directory that contains the Oracle Traffic Director domain.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>instance</td>
<td>Name of the primary/backup Oracle Traffic Director instance which is part of the failover group.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>

Example
props = {}
props['domain-home'] = '/export/otd_domain'
props['instance'] = 'otd_abcl23.example.com'
otd_stopFailover(props)

See Also
help, otd_createFailoverGroup, otd_deleteFailoverGroup, otd_toggleFailoverGroupPrimary, otd_startFailover

2.217 otd_stopSnmpSubAgent

Description
Use this command to stop the Simple Network Management Protocol (SNMP) sub-agent on the specified machine.

Syntax
otd_stopSnmpSubAgent(props)

The argument props is a dictionary that can contain the following properties:
Property Description Comments

machine-name Name specified while creating the machine in the Oracle WebLogic Server console, corresponding to the host name of the machine on which the Oracle Traffic Director instance is running. Mandatory for Online, not valid for Offline.

domain-home Path to the directory that contains the Oracle Traffic Director domain. Mandatory for Offline, not valid for Online.

Example

# Online
props = {}
props['machine-name'] = 'host.example.com'
otd_stopSnmpSubAgent(props)

# Offline
props = {}
props['domain-home'] = '/export/domains/otd_domain'
notd_stopSnmpSubAgent(props)

See Also

help, otd_startSnmpSubAgent, otd_setSnmpProperties, otd_getSnmpProperties

2.218 otd_toggleFailoverGroupPrimary

Description

Use this command to toggle the primary and backup instances in a failover group. If the failover is running already, you should execute the stopFailover and startfailover scripts on the hosts where the instances are running. This is to manually toggle the nodes. If this command is not executed, the instances will not be toggled. Also, when you execute otd_getFailoverGroupProperties, the result will show the configured primary and the backup instances, which will not be the same as the runtime primary and backup instances.

Syntax

otd_toggleFailoverGroupPrimary(props)

The argument props is a dictionary that can contain the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration</td>
<td>Name of the configuration.</td>
<td>Mandatory.</td>
</tr>
<tr>
<td>virtual-ip</td>
<td>Virtual IP that uniquely identifies the failover group.</td>
<td>Mandatory.</td>
</tr>
</tbody>
</table>
Example
props = {}
props['configuration'] = 'foo'
props['virtual-ip'] = '192.0.2.1'
otd_toggleFailoverGroupPrimary(props)

See Also
help, otd_deleteFailoverGroup, otd_createFailoverGroup,
otd_getFailoverGroupProperties, otd_startFailover, otd_stopFailover

2.219 pullComponentChanges

Description
Pulls configuration files from a particular instance of the configuration to the config
store in the admin server. The pulled configuration files overwrite their corresponding
server versions and any pending changes (conflicting with the pulled configuration
files) on the admin server are lost.

After executing this command, you must execute the command
enableOverwriteComponentChanges before activate. Otherwise, activate
will fail because of the local modifications on the instance.

Note:
This command can only be executed from an open edit session. You must
execute the command activate for the pulled configuration changes to be
deployed across all the instances of the configuration.

Syntax
pullComponentChanges(<instance_name>)

The argument <instance_name> is the name of the instance and is mandatory.

Example
startEdit()
pullComponentChanges('otd_test.example.com')
pull component otd_test.example.com changes on machine example.com:
add OTD/test/config/foo.conf
edit OTD/test/config/server.xml
edit OTD/test/config/test-obj.conf
remove OTD/test/config/obj.conf
activate()

See Also
help, enableOverwriteComponentChanges, resync/resyncAll,
showComponentChanges, stopEdit, undo
2.220 resync/resyncAll

Description
Over writes the modifications on an instance or all instances with their corresponding server versions from the admin server.

Syntax
resync(<instance_name>) / resyncAll()

The argument <instance_name> is the name of the instance and is mandatory.

Note:
This command cannot be executed from an open edit session. See enableOverwriteComponentChanges and activate for overriding instance changes within an open edit session.

Example
# resync
showComponentChanges('otd_test.example.com')
add OTD/test/config/bar.conf 1970.01.01-05:30:00 2014.11.07-17:35:15
edit OTD/test/config/proxyvs.obj.conf 2014.11.07-17:36:49 1970.01.01-05:29:59
edit OTD/test/config/server.xml 2014.11.07-17:36:49 2014.11.07-17:37:22
remove OTD/test/config/test-obj.conf 2014.11.07-17:36:49 1970.01.01-05:30:00
resync('otd_test.example.com')

showComponentChanges('otd_test.example.com')
component otd_test.example.com changes on machine example.com: no change found.

# resyncAll
showComponentChanges()
component otd_test.example.com changes on machine example.com:
add OTD/test/config/baz.conf 1970.01.01-05:30:00 2014.11.07-17:42:57
component otd_origin-server-1.example.com changes on machine example.com:
add OTD/origin-server-1/config/bar.conf 1970.01.01-05:30:00 2014.11.07-17:43:34
resyncAll()

showComponentChanges()
component otd_test.example.com changes on machine example.com: no change found.
component otd_origin-server-1.example.com changes on machine example.com: no change found.

See Also
help, enableOverwriteComponentChanges, pullComponentChanges, showComponentChanges, stopEdit, undo
### 2.221 showComponentChanges

**Description**

Lists all the configuration file modifications on instances.

**Syntax**

```
showComponentChanges(<instance_name>)
```

The argument `<instance_name>` is the name of the instance and is optional. If not specified, the command will display the modifications across all the instances.

---

**Note:**

Configuration changes in Oracle Traffic Director sometimes requires changes to multiple files such as `server.xml`, `obj.conf`, and `magnus.conf`. Hence configuration changes in Oracle Traffic Director should either be overridden or pulled with these files treated as a unit in order to avoid inconsistencies. As a result, even if one of these files is modified, all of them will be shown as modified since they are treated as a file unit.

---

**Example**

```
showComponentChanges()
component otd_test.example.com changes on machine example.com: no change found.
component otd_origin-server-1.example.com changes on machine example.com: no change found.
component otd_origin-server-2.example.com changes on machine example.com: no change found.
component otd_origin-server-3.example.com changes on machine example.com: no change found.

showComponentChanges('otd_test.example.com')
add OTD/test/config/foo.conf 1970.01.01-05:30:00 2014.11.01-05:30:00
edit OTD/test/config/server.xml 2014.11.06-19:48:15 2014.11.07-17:06:08
edit OTD/test/config/test-obj.conf 2014.11.06-16:59:32 1970.01.01-05:29:59
remove OTD/test/config/obj.conf 2014.11.06-19:48:15 1970.01.01-05:30:00
```

**See Also**

`help, enableOverwriteComponentChanges, pullComponentChanges, resync/resyncAll, stopEdit, undo`

### 2.222 softRestart

**Description**

Use this WLST command to restart or reconfigure the instance.

Reconfigure dynamically applies configuration changes on instances without a server restart. Only dynamically reconfigurable changes in the configuration take effect. Changes in the `user`, `temp-path`, `log`, `thread-pool`, `pkcs11`, `stats`, `dns`, `dns-cache`, `ssl-session-cache`, and `access-log-buffer` settings remain the same after a reconfiguration procedure is completed. A Restart-required exception will be
thrown if there are any such changes that require restart when a reconfiguration is
done.

Syntax
softRestart(name, [block], [properties])

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the system component to restart.</td>
</tr>
<tr>
<td>block</td>
<td>Optional. Boolean value specifying whether WLST should block user interaction until the server is restarted.</td>
</tr>
<tr>
<td>properties</td>
<td>Optional. Properties value specifying properties to pass to the system component.</td>
</tr>
</tbody>
</table>

Example
Reconfiguring the instance:
props = java.util.Properties()
props.setProperty("MODE", "RECONFIG")
cmo.softRestart(props)

Restarting the instance:
cmo.softRestart(java.util.Properties())

See Also
help, otd_deleteInstance, otd_listInstances, start, stop, otd_createInstance

2.223 start

Description
Starts an instance.

Syntax
start(name, [type])

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the system component to start.</td>
</tr>
<tr>
<td>type</td>
<td>Optional. Type, Server or Cluster. This argument defaults to Server. When starting a cluster, you must set this argument explicitly to Cluster, or the command will fail.</td>
</tr>
</tbody>
</table>
2.224 state

**Description**
Returns the state of an instance.

**Syntax**
```java
state(name, [type])
```

**Argument**
<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>type</td>
</tr>
</tbody>
</table>

**Example**
```java
state('otd_test.in.example.com')
```

**See Also**
help, otd_deleteInstance, otd_listInstances, otd_createInstance, stop, softRestart

2.225 stop

**Description**
Stops an instance.

**Syntax**
```java
stop(name, [type])
```

**Example**
```java
stop('host.example.com', 'SystemComponent')
```

**See Also**
help, otd_deleteInstance, otd_listInstances, otd_createInstance, stop, softRestart
### 2.226 stopEdit

**Description**

Stops the edit session, discards unsaved changes and releases the edit lock.

**Syntax**

```python
stopEdit([defaultAnswer])
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultAnswer</td>
<td>Optional. Default response, if you would prefer not to be prompted at the command line. Valid values are y and n. This argument defaults to null, and WLST prompts you for a response.</td>
</tr>
</tbody>
</table>

**Example**

The following example stops the current editing session. WLST prompts for verification before canceling.

```bash
wls:/mydomain/edit !> stopEdit()
Sure you would like to stop your edit session? (y/n)
y
Edit session has been stopped successfully.
wls:/mydomain/edit>
```

**See Also**

help, enableOverwriteComponentChanges, pullComponentChanges, resync/resyncAll, showComponentChanges, undo

---

### 2.227 undo

**Description**

This command reverts all unsaved (`undo()`) or unactivated (`undo('true')`) edits. This command does not release the edit session. The effect of this command is not limited to Oracle Traffic Director. All the changes done after starting an edit session to the various other components and managed servers will also be reverted.

**Syntax**

```python
undo([unactivatedChanges], [defaultAnswer])
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>unactivatedChanges</td>
<td>Optional. Boolean value specifying whether to undo all unactivated changes, including edits that have been saved to disk. This argument defaults to false, indicating that all edits since the last save operation are reverted.</td>
</tr>
</tbody>
</table>
**Argument**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultAnswer</td>
<td>Optional. Default response, if you would prefer not to be prompted at the command line. Valid values are y and n. This argument defaults to null, and WLST prompts you for a response.</td>
</tr>
</tbody>
</table>

**Example**

The following example reverts all changes since the last save operation. WLST prompts for verification before reverting.

```
wlst:/mydomain/edit !> undo()
Sure you would like to undo your changes? (y/n)
y
Discarded your in-memory changes successfully.
wls:/mydomain/edit>
```

The following example reverts all unactivated changes. WLST prompts for verification before reverting.

```
wls:/mydomain/edit !> undo('true')
Sure you would like to undo your changes? (y/n)
y
Discarded all your changes successfully.
wls:/mydomain/edit>
```

**See Also**

help, enableOverwriteComponentChanges, pullComponentChanges, resync/resyncAll, showComponentChanges, stopEdit