Apache Tomcat 9

Version 9.0.50, Jun 28 2021

Manager App How-To

Table of Contents

Introduction

In many production environments, it is very useful to have the capability to deploy a new web application, or undeploy an existing one, without having to shut down and restart the entire container. In addition, you can request an existing application to reload itself, even if you have not declared it to be `reloadable` in the Tomcat server configuration file.

To support these capabilities, Tomcat includes a web application (installed by default on context path `/manager`) that supports the following functions:

- Deploy a new web application from the uploaded contents of a WAR file.
- Deploy a new web application, on a specified context path, from the server file system.
- List the currently deployed web applications, as well as the sessions that are currently active for those web apps.
- Reload an existing web application, to reflect changes in the contents of `/WEB-INF/classes` or `/WEB-INF/lib`.
- List the OS and JVM property values.
- List the available global JNDI resources, for use in deployment tools that are preparing `<ResourceLink>` elements nested in a `<Context>` deployment description.
- Start a stopped application (thus making it available again).
- Stop an existing application (so that it becomes unavailable), but do not undeploy it.
- Undeploy a deployed web application and delete its document base directory (unless it was deployed from file system).

A default Tomcat installation includes an instance of the Manager application configured for the default virtual host. If you create additional virtual hosts, you may wish to add an instance of the Manager application to one or more of those Hosts. To add an instance of the Manager web application `Context` to a new host install the `manager.xml` context configuration file in the `$CATALINA_BASE/conf/[enginename]/[hostname]` folder. Here is an example:

```
<Context privileged="true" antiResourceLocking="false"
         docBase="${catalina.home}/webapps/manager">
  <CookieProcessor className="org.apache.tomcat.util.http.Rfc6265CookieProcessor"
                   sameSiteCookies="strict" />
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
  <Manager sessionAttributeValueClassNameFilter="java\.lang\.(?:Boolean|Integer|Long|Numbe
</Context>
```

There are three ways to use the **Manager** web application.

- As an application with a user interface you use in your browser. Here is an example URL where you can replace `localhost` with your website host name: `http://localhost:8080/manager/html` .
- A minimal version using HTTP requests only which is suitable for use by scripts setup by system administrators. Commands are given as part of the request URI, and responses are in the form of simple text that can be easily parsed and processed. See Supported Manager Commands for more information.
- A convenient set of task definitions for the *Ant* (version 1.4 or later) build tool. See Executing Manager Commands With Ant for more information.

Configuring Manager Application Access

The description below uses the variable name $CATALINA_BASE to refer the base directory against which most relative paths are resolved. If you have not configured Tomcat for multiple instances by setting a CATALINA_BASE

directory, then $CATALINA_BASE will be set to the value of $CATALINA_HOME, the directory into which you have installed Tomcat.

It would be quite unsafe to ship Tomcat with default settings that allowed anyone on the Internet to execute the Manager application on your server. Therefore, the Manager application is shipped with the requirement that anyone who attempts to use it must authenticate themselves, using a username and password that have one of **manager-xxx** roles associated with them (the role name depends on what functionality is required). Further, there is no username in the default users file (`$CATALINA_BASE/conf/tomcat-users.xml`) that is assigned to those roles. Therefore, access to the Manager application is completely disabled by default.

You can find the role names in the `web.xml` file of the Manager web application. The available roles are:

- **manager-gui —** Access to the HTML interface.
- **manager-status —** Access to the "Server Status" page only.
- **manager-script —** Access to the tools-friendly plain text interface that is described in this document, and to the "Server Status" page.
- **manager-jmx —** Access to JMX proxy interface and to the "Server Status" page.

The HTML interface is protected against CSRF (Cross-Site Request Forgery) attacks, but the text and JMX interfaces cannot be protected. It means that users who are allowed access to the text and JMX interfaces have to be cautious when accessing the Manager application with a web browser. To maintain the CSRF protection:

- If you use web browser to access the Manager application using a user that has either **manager-script** or **manager-jmx** roles (for example for testing the plain text or JMX interfaces), you MUST close all windows of the browser afterwards to terminate the session. If you do not close the browser and visit other sites, you may become victim of a CSRF attack.
- It is recommended to never grant the **manager-script** or **manager-jmx** roles to users that have the **manager-gui** role.

Note that JMX proxy interface is effectively low-level root-like administrative interface of Tomcat. One can do a lot, if one knows what commands to call. You should be cautious when enabling the **manager-jmx** role.

To enable access to the Manager web application, you must either create a new username/password combination and associate one of the **manager-xxx** roles with it, or add a **manager-xxx** role to some existing username/password combination. As the majority of this document describes the using the text interface, this example will use the role name **manager-script**. Exactly how the usernames/passwords are configured depends on which Realm implementation you are using:

- *UserDatabaseRealm* plus *MemoryUserDatabase*, or *MemoryRealm* **—** The *UserDatabaseRealm* and *MemoryUserDatabase* are configured in the default `$CATALINA_BASE/conf/server.xml`. Both *MemoryUserDatabase* and *MemoryRealm* read an XML-format file by default stored at `$CATALINA_BASE/conf/tomcat-users.xml`, which can be edited with any text editor. This file contains an XML `<user>` for each individual user, which might look something like this:

  ```
  <user username="craigmcc" password="secret" roles="standard,manager-script" />
  ```

  which defines the username and password used by this individual to log on, and the role names they are associated with. You can add the **manager-script** role to the comma-delimited `roles` attribute for one or more existing users, and/or create new users with that assigned role.

- *DataSourceRealm* or *JDBCRealm* — Your user and role information is stored in a database accessed via JDBC. Add the **manager-script** role to one or more existing users, and/or create one or more new users with this role assigned, following the standard procedures for your environment.
- *JNDIRealm* — Your user and role information is stored in a directory server accessed via LDAP. Add the **manager-script** role to one or more existing users, and/or create one or more new users with this role assigned, following the standard procedures for your environment.

The first time you attempt to issue one of the Manager commands described in the next section, you will be challenged to log on using BASIC authentication. The username and password you enter do not matter, as long as they identify a valid user in the users database who possesses the role **manager-script**.

In addition to the password restrictions, access to the Manager web application can be restricted by the **remote IP address** or host by adding a `RemoteAddrValve` or `RemoteHostValve`. See valves documentation for details. Here is an example of restricting access to the localhost by IP address:

```
<Context privileged="true">
         <Valve className="org.apache.catalina.valves.RemoteAddrValve"
                allow="127\.0\.0\.1"/>
</Context>
```

HTML User-friendly Interface

The user-friendly HTML interface of Manager web application is located at

```
http://{host}:{port}/manager/html
```

As has already been mentioned above, you need **manager-gui** role to be allowed to access it. There is a separate document that provides help on this interface. See:

- HTML Manager documentation

The HTML interface is protected against CSRF (Cross-Site Request Forgery) attacks. Each access to the HTML pages generates a random token, which is stored in your session and is included in all links on the page. If your next action does not have correct value of the token, the action will be denied. If the token has expired you can start again from the main page or *List Applications* page of Manager.

Supported Manager Commands

All commands that the Manager application knows how to process are specified in a single request URI like this:

```
http://{host}:{port}/manager/text/{command}?{parameters}
```

where `{host}` and `{port}` represent the hostname and port number on which Tomcat is running, `{command}` represents the Manager command you wish to execute, and `{parameters}` represents the query parameters that are specific to that command. In the illustrations below, customize the host and port appropriately for your installation.

The commands are usually executed by HTTP GET requests. The `/deploy` command has a form that is executed by an HTTP PUT request.

Common Parameters

Most commands accept one or more of the following query parameters:

- **path -** The context path (including the leading slash) of the web application you are dealing with. To select the ROOT web application, specify "/".
  NOTE: It is not possible to perform administrative commands on the Manager application itself.
  NOTE: If the path parameter is not explicitly specified then the path and the version will be derived using the standard Context naming rules from the config parameter or, if the config parameter is not present, the war parameter.
- **version -** The version of this web application as used by the parallel deployment feature. If you use parallel deployment wherever a path is required you must specify a version in addition to the path and it is the combination of path and version that must be unique rather than just the path.
  NOTE: If the path is not explicitly specified, the version parameter is ignored.
- **war -** URL of a web application archive (WAR) file, or pathname of a directory which contains the web application, or a Context configuration ".xml" file. You can use URLs in any of the following formats:
  - **file:/absolute/path/to/a/directory -** The absolute path of a directory that contains the unpacked version of a web application. This directory will be attached to the context path you specify without any changes.
  - **file:/absolute/path/to/a/webapp.war -** The absolute path of a web application archive (WAR) file. This is valid **only** for the `/deploy` command, and is the only acceptable format to that command.
  - **file:/absolute/path/to/a/context.xml** - The absolute path of a web application Context configuration ".xml" file which contains the Context configuration element.
  - **directory -** The directory name for the web application context in the Host's application base directory.
  - **webapp.war -** The name of a web application war file located in the Host's application base directory.

Each command will return a response in `text/plain` format (i.e. plain ASCII with no HTML markup), making it easy for both humans and programs to read). The first line of the response will begin with either `OK` or `FAIL`, indicating whether the requested command was successful or not. In the case of failure, the rest of the first line will contain a description of the problem that was encountered. Some commands include additional lines of information as described below.

Internationalization Note **-** The Manager application looks up its message strings in resource bundles, so it is possible that the strings have been translated for your platform. The examples below show the English version of the messages.

Deploy A New Application Archive (WAR) Remotely

```
http://localhost:8080/manager/text/deploy?path=/foo
```

Upload the web application archive (WAR) file that is specified as the request data in this HTTP PUT request, install it into the `appBase` directory of our corresponding virtual host, and start, deriving the name for the WAR file added to the `appBase` from the specified path. The application can later be undeployed (and the corresponding WAR file removed) by use of the `/undeploy` command.

This command is executed by an HTTP PUT request.

The .WAR file may include Tomcat specific deployment configuration, by including a Context configuration XML file in `/META-INF/context.xml`.

URL parameters include:

- `update`: When set to true, any existing update will be undeployed first. The default value is set to false.
- `tag`: Specifying a tag name, this allows associating the deployed webapp with a tag or label. If the web application is undeployed, it can be later redeployed when needed using only the tag.
- `config` : URL of a Context configuration ".xml" file in the format **file:/absolute/path/to/a/context.xml**. This must be the absolute path of a web application Context configuration ".xml" file which contains the Context configuration element.

NOTE - This command is the logical opposite of the `/undeploy` command.

If installation and startup is successful, you will receive a response like this:

```
OK - Deployed application at context path /foo
```

Otherwise, the response will start with `FAIL` and include an error message. Possible causes for problems include:

- *Application already exists at path /foo*

  The context paths for all currently running web applications must be unique. Therefore, you must undeploy the existing web application using this context path, or choose a different context path for the new one. The `update` parameter may be specified as a parameter on the URL, with a value of `true` to avoid this error. In that case, an undeploy will be performed on an existing application before performing the deployment.

- *Encountered exception*

  An exception was encountered trying to start the new web application. Check the Tomcat logs for the details, but likely explanations include problems parsing your `/WEB-INF/web.xml` file, or missing classes encountered when initializing application event listeners and filters.

Deploy A New Application from a Local Path

Deploy and start a new web application, attached to the specified context `path` (which must not be in use by any other web application). This command is the logical opposite of the `/undeploy` command.

This command is executed by an HTTP GET request. There are a number of different ways the deploy command can be used.

Deploy a previously deployed webapp

```
http://localhost:8080/manager/text/deploy?path=/footoo&tag=footag
```

This can be used to deploy a previously deployed web application, which has been deployed using the `tag` attribute. Note that the work directory of the Manager webapp will contain the previously deployed WARs; removing it would make the deployment fail.

Deploy a Directory or WAR by URL

Deploy a web application directory or ".war" file located on the Tomcat server. If no `path` is specified, the path and version are derived from the directory name or the war file name. The `war` parameter specifies a URL (including the `file:` scheme) for either a directory or a web application archive (WAR)

file. The supported syntax for a URL referring to a WAR file is described on the Javadocs page for the `java.net.JarURLConnection` class. Use only URLs that refer to the entire WAR file.

In this example the web application located in the directory `/path/to/foo` on the Tomcat server is deployed as the web application context named `/footoo`.

```
http://localhost:8080/manager/text/deploy?path=/footoo&war=file:/path/to/foo
```

In this example the ".war" file `/path/to/bar.war` on the Tomcat server is deployed as the web application context named `/bar`. Notice that there is no `path` parameter so the context path defaults to the name of the web application archive file without the ".war" extension.

```
http://localhost:8080/manager/text/deploy?war=file:/path/to/bar.war
```

Deploy a Directory or War from the Host appBase

Deploy a web application directory or ".war" file located in your Host appBase directory. The path and optional version are derived from the directory or war file name.

In this example the web application located in a sub directory named `foo` in the Host appBase directory of the Tomcat server is deployed as the web application context named `/foo`. Notice that the context path used is the name of the web application directory.

```
http://localhost:8080/manager/text/deploy?war=foo
```

In this example the ".war" file `bar.war` located in your Host appBase directory on the Tomcat server is deployed as the web application context named `/bar`.

```
http://localhost:8080/manager/text/deploy?war=bar.war
```

Deploy using a Context configuration ".xml" file

If the Host deployXML flag is set to true you can deploy a web application using a Context configuration ".xml" file and an optional ".war" file or web application directory. The context `path` is not used when deploying a web application using a context ".xml" configuration file.

A Context configuration ".xml" file can contain valid XML for a web application Context just as if it were configured in your Tomcat `server.xml` configuration file. Here is an example:

```
<Context path="/foobar" docBase="/path/to/application/foobar">
</Context>
```

When the optional `war` parameter is set to the URL for a web application ".war" file or directory it overrides any docBase configured in the context configuration ".xml" file.

Here is an example of deploying an application using a Context configuration ".xml" file.

```
http://localhost:8080/manager/text/deploy?config=file:/path/context.xml
```

Here is an example of deploying an application using a Context configuration ".xml" file and a web application ".war" file located on the server.

```
http://localhost:8080/manager/text/deploy
```

```
?config=file:/path/context.xml&war=file:/path/bar.war
```

Deployment Notes

If the Host is configured with unpackWARs=true and you deploy a war file, the war will be unpacked into a directory in your Host appBase directory.

If the application war or directory is installed in your Host appBase directory and either the Host is configured with autoDeploy=true or the Context path must match the directory name or war file name without the ".war" extension.

For security when untrusted users can manage web applications, the Host deployXML flag can be set to false. This prevents untrusted users from deploying web applications using a configuration XML file and also prevents them from deploying application directories or ".war" files located outside of their Host appBase.

Deploy Response

If installation and startup is successful, you will receive a response like this:

```
OK - Deployed application at context path /foo
```

Otherwise, the response will start with `FAIL` and include an error message. Possible causes for problems include:

- *Application already exists at path /foo*

  The context paths for all currently running web applications must be unique. Therefore, you must undeploy the existing web application using this context path, or choose a different context path for the new one. The `update` parameter may be specified as a parameter on the URL, with a value of `true` to avoid this error. In that case, an undeploy will be performed on an existing application before performing the deployment.

- *Document base does not exist or is not a readable directory*

  The URL specified by the `war` parameter must identify a directory on this server that contains the "unpacked" version of a web application, or the absolute URL of a web application archive (WAR) file that contains this application. Correct the value specified by the `war` parameter.

- *Encountered exception*

  An exception was encountered trying to start the new web application. Check the Tomcat logs for the details, but likely explanations include problems parsing your `/WEB-INF/web.xml` file, or missing classes encountered when initializing application event listeners and filters.

- *Invalid application URL was specified*

  The URL for the directory or web application that you specified was not valid. Such URLs must start with `file:`, and URLs for a WAR file must end in ".war".

- *Invalid context path was specified*

  The context path must start with a slash character. To reference the ROOT web application use "/".

- *Context path must match the directory or WAR file name:*

  If the application war or directory is installed in your Host appBase directory and either the Host is configured with autoDeploy=true the Context path must match the directory name or war file name without the ".war" extension.

- *Only web applications in the Host web application directory can be installed*

  If the Host deployXML flag is set to false this error will happen if an attempt is made to deploy a web application directory or ".war" file outside of the Host appBase directory.

List Currently Deployed Applications

```
http://localhost:8080/manager/text/list
```

List the context paths, current status (`running` or `stopped`), and number of active sessions for all currently deployed web applications. A typical response immediately after starting Tomcat might look like this:

```
OK - Listed applications for virtual host localhost
/webdav:running:0:webdav
/examples:running:0:examples
/manager:running:0:manager
/:running:0:ROOT
/test:running:0:test##2
/test:running:0:test##1
```

Reload An Existing Application

```
http://localhost:8080/manager/text/reload?path=/examples
```

Signal an existing application to shut itself down and reload. This can be useful when the web application context is not reloadable and you have updated classes or property files in the `/WEB-INF/classes` directory or when you have added or updated jar files in the `/WEB-INF/lib` directory.

If this command succeeds, you will see a response like this:

```
OK - Reloaded application at context path /examples
```

Otherwise, the response will start with `FAIL` and include an error message. Possible causes for problems include:

- *Encountered exception*

  An exception was encountered trying to restart the web application. Check the Tomcat logs for the details.

- *Invalid context path was specified*

  The context path must start with a slash character. To reference the ROOT web application use "/".

- *No context exists for path /foo*

  There is no deployed application on the context path that you specified.

- *No context path was specified*

  The path parameter is required.

- *Reload not supported on WAR deployed at path /foo*

  Currently, application reloading (to pick up changes to the classes or `web.xml` file) is not supported when a web application is deployed directly from a WAR file. It only works when the web application is deployed from an unpacked directory. If you are using a WAR file, you should `undeploy` and then `deploy` or `deploy` with the `update` parameter the application again to pick up your changes.

List OS and JVM Properties

```
http://localhost:8080/manager/text/serverinfo
```

Lists information about the Tomcat version, OS, and JVM properties.

If an error occurs, the response will start with `FAIL` and include an error message. Possible causes for problems include:

- *Encountered exception*

  An exception was encountered trying to enumerate the system properties. Check the Tomcat logs for the details.

List Available Global JNDI Resources

```
http://localhost:8080/manager/text/resources[?type=xxxxx]
```

List the global JNDI resources that are available for use in resource links for context configuration files. If you specify the `type` request parameter, the value must be the fully qualified Java class name of the resource type you are interested in (for example, you would specify `javax.sql.DataSource` to acquire the names of all available JDBC data sources). If you do not specify the `type` request parameter, resources of all types will be returned.

Depending on whether the `type` request parameter is specified or not, the first line of a normal response will be:

```
OK - Listed global resources of all types
```

or

```
OK - Listed global resources of type xxxxx
```

followed by one line for each resource. Each line is composed of fields delimited by colon characters (":"), as follows:

- *Global Resource Name* - The name of this global JNDI resource, which would be used in the `global` attribute of a `<ResourceLink>` element.
- *Global Resource Type* - The fully qualified Java class name of this global JNDI resource.

If an error occurs, the response will start with `FAIL` and include an error message. Possible causes for problems include:

- *Encountered exception*

  An exception was encountered trying to enumerate the global JNDI resources. Check the Tomcat logs for the details.

- *No global JNDI resources are available*

  The Tomcat server you are running has been configured without global JNDI resources.

Session Statistics

```
http://localhost:8080/manager/text/sessions?path=/examples
```

Display the default session timeout for a web application, and the number of currently active sessions that fall within one-minute ranges of their actual timeout times. For example, after restarting Tomcat and then executing one of the JSP samples in the `/examples` web app, you might get something like this:

```
OK - Session information for application at context path /examples
Default maximum session inactive interval 30 minutes
<1 minutes: 1 sessions
1 - <2 minutes: 1 sessions
```

Expire Sessions

```
http://localhost:8080/manager/text/expire?path=/examples&idle=num
```

Display the session statistics (like the above `/sessions` command) and expire sessions that are idle for longer than `num` minutes. To expire all sessions, use `&idle=0` .

```
OK - Session information for application at context path /examples
Default maximum session inactive interval 30 minutes
1 - <2 minutes: 1 sessions
3 - <4 minutes: 1 sessions
>0 minutes: 2 sessions were expired
```

Actually `/sessions` and `/expire` are synonyms for the same command. The difference is in the presence of `idle` parameter.

Start an Existing Application

```
http://localhost:8080/manager/text/start?path=/examples
```

Signal a stopped application to restart, and make itself available again. Stopping and starting is useful, for example, if the database required by your application becomes temporarily unavailable. It is usually better to stop the web application that relies on this database rather than letting users continuously encounter database exceptions.

If this command succeeds, you will see a response like this:

```
OK - Started application at context path /examples
```

Otherwise, the response will start with `FAIL` and include an error message. Possible causes for problems include:

- *Encountered exception*

  An exception was encountered trying to start the web application. Check the Tomcat logs for the details.

- *Invalid context path was specified*

  The context path must start with a slash character. To reference the ROOT web application use "/".

- *No context exists for path /foo*

  There is no deployed application on the context path that you specified.

- *No context path was specified*

  The `path` parameter is required.

Stop an Existing Application

```
http://localhost:8080/manager/text/stop?path=/examples
```

Signal an existing application to make itself unavailable, but leave it deployed. Any request that comes in while an application is stopped will see an HTTP error 404, and this application will show as "stopped" on a list applications command.

If this command succeeds, you will see a response like this:

```
OK - Stopped application at context path /examples
```

Otherwise, the response will start with `FAIL` and include an error message. Possible causes for problems include:

- *Encountered exception*

  An exception was encountered trying to stop the web application. Check the Tomcat logs for the details.

- *Invalid context path was specified*

  The context path must start with a slash character. To reference the ROOT web application use "/".

- *No context exists for path /foo*

  There is no deployed application on the context path that you specified.

- *No context path was specified* The `path` parameter is required.

Undeploy an Existing Application

```
http://localhost:8080/manager/text/undeploy?path=/examples
```

WARNING - This command will delete any web application artifacts that exist within `appBase` directory (typically "webapps") for this virtual host. This will delete the application .WAR, if present, the application directory resulting either from a deploy in unpacked form or from .WAR expansion as well as the XML Context definition from `$CATALINA_BASE/conf/[enginename]/[hostname]/` directory. If you simply want to take an application out of service, you should use the `/stop` command instead.

Signal an existing application to gracefully shut itself down, and remove it from Tomcat (which also makes this context path available for reuse later). In addition, the document root directory is removed, if it exists in the `appBase` directory (typically "webapps") for this virtual host. This command is the logical opposite of the `/deploy` command.

If this command succeeds, you will see a response like this:

```
OK - Undeployed application at context path /examples
```

Otherwise, the response will start with `FAIL` and include an error message. Possible causes for problems include:

- *Encountered exception*

  An exception was encountered trying to undeploy the web application. Check the Tomcat logs for the details.

- *Invalid context path was specified*

  The context path must start with a slash character. To reference the ROOT web application use "/".

- *No context exists named /foo*

  There is no deployed application with the name that you specified.

- *No context path was specified* The `path` parameter is required.

Finding memory leaks

```
http://localhost:8080/manager/text/findleaks[?statusLine=[true|false]]
```

The find leaks diagnostic triggers a full garbage collection. It should be used with extreme caution on production systems.

The find leaks diagnostic attempts to identify web applications that have caused memory leaks when they were stopped, reloaded or undeployed. Results should always be confirmed with a profiler. The diagnostic uses additional functionality provided by the StandardHost implementation. It will not work if a custom host is used that does not extend StandardHost.

Explicitly triggering a full garbage collection from Java code is documented to be unreliable. Furthermore, depending on the JVM used, there are options to disable explicit GC triggering, like `-XX:+DisableExplicitGC`. If you want to make sure, that the diagnostics were successfully running a full GC, you will need to check using tools like GC logging, JConsole or similar.

If this command succeeds, you will see a response like this:

```
/leaking-webapp
```

If you wish to see a status line included in the response then include the `statusLine` query parameter in the request with a value of `true`.

Each context path for a web application that was stopped, reloaded or undeployed, but which classes from the previous runs are still loaded in memory, thus causing a memory leak, will be listed on a new line. If an application has been reloaded several times, it may be listed several times.

If the command does not succeed, the response will start with `FAIL` and include an error message.

Connector SSL/TLS cipher information

```
http://localhost:8080/manager/text/sslConnectorCiphers
```

The SSL Connector/Ciphers diagnostic lists the SSL/TLS ciphers that are currently configured for each connector. For NIO and NIO2, the names of the individual cipher suites are listed. For APR, the value of SSLCipherSuite is returned.

The response will look something like this:

```
OK - Connector / SSL Cipher information
Connector[HTTP/1.1-8080]
  SSL is not enabled for this connector
Connector[HTTP/1.1-8443]
  TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
  TLS_DHE_RSA_WITH_AES_128_CBC_SHA
  TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
  TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
  ...
```

Connector SSL/TLS certificate chain information

```
http://localhost:8080/manager/text/sslConnectorCerts
```

The SSL Connector/Certs diagnostic lists the certificate chain that is currently configured for each virtual host.

The response will look something like this:

```
OK - Connector / Certificate Chain information
Connector[HTTP/1.1-8080]
SSL is not enabled for this connector
Connector[HTTP/1.1-8443]-_default_-RSA
[
[
  Version: V3
  Subject: CN=localhost, OU=Apache Tomcat PMC, O=The Apache Software Foundation, L=Wakef

  Signature Algorithm: SHA256withRSA, OID = 1.2.840.113549.1.1.11
  ...
```

Connector SSL/TLS trusted certificate information

```
http://localhost:8080/manager/text/sslConnectorTrustedCerts
```

The SSL Connector/Certs diagnostic lists the trusted certificates that are currently configured for each virtual host.

The response will look something like this:

```
OK - Connector / Trusted Certificate information
Connector[HTTP/1.1-8080]
SSL is not enabled for this connector
Connector[HTTP/1.1-8443]-_default_
[
[
  Version: V3
  Subject: CN=Apache Tomcat Test CA, OU=Apache Tomcat PMC, O=The Apache Software Foundat
  ...
```

Reload TLS configuration

```
http://localhost:8080/manager/text/sslReload?tlsHostName=name
```

Reload the TLS configuration files (the certificate and key files, this does not trigger a re-parsing of server.xml). To reload the files for all hosts don't specify the `tlsHostName` parameter.

```
OK - Reloaded TLS configuration for [_default_]
```

Thread Dump

```
http://localhost:8080/manager/text/threaddump
```

Write a JVM thread dump.

The response will look something like this:

```
OK - JVM thread dump
2014-12-08 07:24:40.080
Full thread dump Java HotSpot(TM) Client VM (25.25-b02 mixed mode):

"http-nio-8080-exec-2" Id=26 cpu=46800300 ns usr=46800300 ns blocked 0 for -1 ms waited
   java.lang.Thread.State: RUNNABLE
        locks java.util.concurrent.ThreadPoolExecutor$Worker@1738ad4
        at sun.management.ThreadImpl.dumpThreads0(Native Method)
        at sun.management.ThreadImpl.dumpAllThreads(ThreadImpl.java:446)
        at org.apache.tomcat.util.Diagnostics.getThreadDump(Diagnostics.java:440)
        at org.apache.tomcat.util.Diagnostics.getThreadDump(Diagnostics.java:409)
        at org.apache.catalina.manager.ManagerServlet.threadDump(ManagerServlet.java:557
        at org.apache.catalina.manager.ManagerServlet.doGet(ManagerServlet.java:371)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:618)
        at javax.servlet.http.HttpServlet.service(HttpServlet.java:725)
...
```

VM Info

```
http://localhost:8080/manager/text/vminfo
```

Write some diagnostic information about Java Virtual Machine.

The response will look something like this:

```
OK - VM info
2014-12-08 07:27:32.578
Runtime information:
  vmName: Java HotSpot(TM) Client VM
  vmVersion: 25.25-b02
  vmVendor: Oracle Corporation
  specName: Java Virtual Machine Specification
  specVersion: 1.8
  specVendor: Oracle Corporation
  managementSpecVersion: 1.2
  name: ...
  startTime: 1418012458849
  uptime: 393855
  isBootClassPathSupported: true

OS information:
...
```

Save Configuration

```
http://localhost:8080/manager/text/save
```

If specified without any parameters, this command saves the current configuration of the server to server.xml. The existing file will be renamed as a backup if required.

If specified with a `path` parameter that matches the path of a deployed web application then the configuration for that web application will be saved to an appropriately named context.xml file in the `xmlBase` for the current Host.

To use the command a StoreConfig MBean must be present. Typically this is configured using the StoreConfigLifecycleListener.

If the command does not succeed, the response will start with `FAIL` and include an error message.

Server Status

From the following links you can view Status information about the server. Any one of **manager-xxx** roles allows access to this page.

```
http://localhost:8080/manager/status
http://localhost:8080/manager/status/all
```

Displays server status information in HTML format.

```
http://localhost:8080/manager/status?XML=true
http://localhost:8080/manager/status/all?XML=true
```

Displays server status information in XML format.

First, you have the server and JVM version number, JVM provider, OS name and number followed by the architecture type.

Second, there is information about the memory usage of the JVM.

Then, there is information about the Tomcat AJP and HTTP connectors. The same information is available for both of them :

- Threads information : Max threads, min and max spare threads, current thread count and current thread busy.
- Request information : Max processing time and processing time, request and error count, bytes received and sent.
- A table showing Stage, Time, Bytes Sent, Bytes Receive, Client, VHost and Request. All existing threads are listed in the table. Here is the list of the possible thread stages :
    - *"Parse and Prepare Request"* : The request headers are being parsed or the necessary preparation to read the request body (if a transfer encoding has been specified) is taking place.
    - *"Service"* : The thread is processing a request and generating the response. This stage follows the "Parse and Prepare Request" stage and precedes the "Finishing" stage. There is always at least one thread in this stage (the server-status page).
    - *"Finishing"* : The end of the request processing. Any remainder of the response still in the output buffers is sent to the client. This stage is followed by "Keep-Alive" if it is appropriate to keep the connection alive or "Ready" if "Keep-Alive" is not appropriate.
    - *"Keep-Alive"* : The thread keeps the connection open to the client in case the client sends another request. If another request is received, the next stage will be "Parse and Prepare Request". If no request is received before the keep alive times out, the connection will be closed and the next stage will be "Ready".
    - *"Ready"* : The thread is at rest and ready to be used.

If you are using `/status/all` command, additional information on each of deployed web applications will be available.

Using the JMX Proxy Servlet

What is JMX Proxy Servlet

The JMX Proxy Servlet is a lightweight proxy to get and set the tomcat internals. (Or any class that has been exposed via an MBean) Its usage is not very user friendly but the UI is extremely helpful for integrating command line scripts for monitoring and changing the internals of tomcat. You can do two things with the proxy: get information and set information. For you to really understand the JMX Proxy Servlet, you should have a general understanding of JMX. If you don't know what JMX is, then prepare to be confused.

JMX Query command

This takes the form:

```
http://webserver/manager/jmxproxy/?qry=STUFF
```

Where `STUFF` is the JMX query you wish to perform. For example, here are some queries you might wish to run:

- `qry=*%3Atype%3DRequestProcessor%2C* --> type=RequestProcessor` which will locate all workers which can process requests and report their state.
- `qry=*%3Aj2eeType=Servlet%2c* --> j2eeType=Servlet` which return all loaded servlets.
- `qry=Catalina%3Atype%3DEnvironment%2Cresourcetype%3DGlobal%2Cname%3DsimpleValue --> Catalina:type=Environment,resourcetype=Global,name=simpleValue` which look for a specific MBean by the given name.

You'll need to experiment with this to really understand its capabilities If you provide no `qry` parameter, then all of the MBeans will be displayed. We really recommend looking at the tomcat source code and understand the JMX spec to get a better understanding of all the queries you may run.

JMX Get command

The JXMProxyServlet also supports a "get" command that you can use to fetch the value of a specific MBean's attribute. The general form of the `get` command is:

```
http://webserver/manager/jmxproxy/?get=BEANNAME&att=MYATTRIBUTE&key=MYKEY
```

You must provide the following parameters:

1. `get`: The full bean name
2. `att`: The attribute you wish to fetch
3. `key`: (optional) The key into a CompositeData MBean attribute

If all goes well, then it will say OK, otherwise an error message will be shown. For example, let's say we wish to fetch the current heap memory data:

```
http://webserver/manager/jmxproxy/?get=java.lang:type=Memory&att=HeapMemoryUsage
```

Or, if you only want the "used" key:

```
http://webserver/manager/jmxproxy/
 ?get=java.lang:type=Memory&att=HeapMemoryUsage&key=used
```

JMX Set command

Now that you can query an MBean, its time to muck with Tomcat's internals! The general form of the set command is :

```
http://webserver/manager/jmxproxy/?set=BEANNAME&att=MYATTRIBUTE&val=NEWVALUE
```

So you need to provide 3 request parameters:

1. `set`: The full bean name
2. `att`: The attribute you wish to alter
3. `val`: The new value

If all goes ok, then it will say OK, otherwise an error message will be shown. For example, lets say we wish to turn up debugging on the fly for the `ErrorReportValve`. The following will set debugging to 10.

```
http://localhost:8080/manager/jmxproxy/
 ?set=Catalina%3Atype%3DValve%2Cname%3DErrorReportValve%2Chost%3Dlocalhost
 &att=debug&val=10
```

and my result is (YMMV):

```
Result: ok
```

Here is what I see if I pass in a bad value. Here is the URL I used, I try set debugging equal to 'cow':

```
http://localhost:8080/manager/jmxproxy/
 ?set=Catalina%3Atype%3DValve%2Cname%3DErrorReportValve%2Chost%3Dlocalhost
 &att=debug&val=cow
```

When I try that, my result is

```
Error: java.lang.NumberFormatException: For input string: "cow"
```

JMX Invoke command

The `invoke` command enables methods to be called on MBeans. The general form of the command is:

```
http://webserver/manager/jmxproxy/
 ?invoke=BEANNAME&op=METHODNAME&ps=COMMASEPARATEDPARAMETERS
```

For example, to call the `findConnectors()` method of the **Service** use:

```
http://localhost:8080/manager/jmxproxy/
 ?invoke=Catalina%3Atype%3DService&op=findConnectors&ps=
```

Executing Manager Commands With Ant

In addition to the ability to execute Manager commands via HTTP requests, as documented above, Tomcat includes a convenient set of Task definitions for the *Ant* (version 1.4 or later) build tool. In order to use these commands, you must perform the following setup operations:

- Download the binary distribution of Ant from https://ant.apache.org. You must use version **1.4** or later.
- Install the Ant distribution in a convenient directory (called ANT_HOME in the remainder of these instructions).
- Add the `$ANT_HOME/bin` directory to your `PATH` environment variable.
- Configure at least one username/password combination in your Tomcat user database that includes the `manager-script` role.

To use custom tasks within Ant, you must declare them first with an `<import>` element. Therefore, your `build.xml` file might look something like this:

```
<project name="My Application" default="compile" basedir=".">

  <!-- Configure the directory into which the web application is built -->
```

```
  <property name="build"    value="${basedir}/build"/>

  <!-- Configure the context path for this application -->
  <property name="path"     value="/myapp"/>

  <!-- Configure properties to access the Manager application -->
  <property name="url"      value="http://localhost:8080/manager/text"/>
  <property name="username" value="myusername"/>
  <property name="password" value="mypassword"/>

  <!-- Configure the path to the Tomcat installation -->
  <property name="catalina.home" value="/usr/local/apache-tomcat"/>

  <!-- Configure the custom Ant tasks for the Manager application -->
  <import file="${catalina.home}/bin/catalina-tasks.xml"/>

  <!-- Executable Targets -->
  <target name="compile" description="Compile web application">
    <!-- ... construct web application in ${build} subdirectory, and
             generated a ${path}.war ... -->
  </target>

  <target name="deploy" description="Install web application"
          depends="compile">
    <deploy url="${url}" username="${username}" password="${password}"
            path="${path}" war="file:${build}${path}.war"/>
  </target>

  <target name="reload" description="Reload web application"
          depends="compile">
    <reload  url="${url}" username="${username}" password="${password}"
            path="${path}"/>
  </target>

  <target name="undeploy" description="Remove web application">
    <undeploy url="${url}" username="${username}" password="${password}"
            path="${path}"/>
  </target>

</project>
```

Note: The definition of the resources task via the import above will override the resources datatype added in Ant 1.7. If you wish to use the resources datatype you will need to use Ant's namespace support to modify `catalina-tasks.xml` to assign the Tomcat tasks to their own namespace.

Now, you can execute commands like `ant deploy` to deploy the application to a running instance of Tomcat, or `ant reload` to tell Tomcat to reload it. Note also that most of the interesting values in this `build.xml` file are defined as replaceable properties, so you can override their values from the command line. For example, you might consider it a security risk to include the real manager password in your `build.xml` file's source code. To avoid this, omit the password property, and specify it from the command line:

```
ant -Dpassword=secret deploy
```

Tasks output capture

Using *Ant* version **1.6.2** or later, the Catalina tasks offer the option to capture their output in properties or external files. They support directly the following subset of the `<redirector>` type attributes:

Attribute	Description	Required
output	Name of a file to which to write the output. If the error stream is not also redirected to a file or property, it will appear in this output.	No
error	The file to which the standard error of the command should be redirected.	No
logError	This attribute is used when you wish to see error output in Ant's log and you are redirecting output to a file/property. The error output will not be included in the output file/property. If you redirect error with the *error* or *errorProperty* attributes, this will have no effect.	No
append	Whether output and error files should be appended to or overwritten. Defaults to `false`.	No
createemptyfiles	Whether output and error files should be created even when empty. Defaults to `true`.	No
outputproperty	The name of a property in which the output of the command should be stored. Unless the error stream is redirected to a separate file or stream, this property will include the error output.	No
errorproperty	The name of a property in which the standard error of the command should be stored.	No

A couple of additional attributes can also be specified:

Attribute	Description	Required
alwaysLog	This attribute is used when you wish to see the output you are capturing, appearing also in the Ant's log. It must not be used unless you are capturing task output. Defaults to `false`. *This attribute will be supported directly by `<redirector>` in Ant 1.6.3*	No
failonerror	This attribute is used when you wish to avoid that any manager command processing error terminates the ant execution. Defaults to `true`. It must be set to `false`, if you want to capture error output, otherwise execution will terminate before anything can be captured. This attribute acts only on manager command execution, any wrong or missing command attribute will still cause Ant execution termination.	No

They also support the embedded `<redirector>` element in which you can specify its full set of attributes, but `input`, `inputstring` and `inputencoding` that, even if accepted, are not used because they have no meaning in this context. Refer to ant manual for details on `<redirector>` element attributes.

Here is a sample build file extract that shows how this output redirection support can be used:

```
    <target name="manager.deploy"
        depends="context.status"
        if="context.notInstalled">
```

```
    <deploy url="${mgr.url}"
        username="${mgr.username}"
        password="${mgr.password}"
        path="${mgr.context.path}"
        config="${mgr.context.descriptor}"/>
</target>

<target name="manager.deploy.war"
    depends="context.status"
    if="context.deployable">
    <deploy url="${mgr.url}"
        username="${mgr.username}"
        password="${mgr.password}"
        update="${mgr.update}"
        path="${mgr.context.path}"
        war="${mgr.war.file}"/>
</target>

<target name="context.status">
    <property name="running" value="${mgr.context.path}:running"/>
    <property name="stopped" value="${mgr.context.path}:stopped"/>

    <list url="${mgr.url}"
        outputproperty="ctx.status"
        username="${mgr.username}"
        password="${mgr.password}">
    </list>

    <condition property="context.running">
        <contains string="${ctx.status}" substring="${running}"/>
    </condition>
    <condition property="context.stopped">
        <contains string="${ctx.status}" substring="${stopped}"/>
    </condition>
    <condition property="context.notInstalled">
        <and>
            <isfalse value="${context.running}"/>
            <isfalse value="${context.stopped}"/>
        </and>
    </condition>
    <condition property="context.deployable">
        <or>
            <istrue value="${context.notInstalled}"/>
            <and>
                <istrue value="${context.running}"/>
                <istrue value="${mgr.update}"/>
            </and>
            <and>
                <istrue value="${context.stopped}"/>
                <istrue value="${mgr.update}"/>
            </and>
        </or>
    </condition>
    <condition property="context.undeployable">
        <or>
            <istrue value="${context.running}"/>
            <istrue value="${context.stopped}"/>
```

```
            </or>
        </condition>
    </target>
```

WARNING: even if it doesn't make many sense, and is always a bad idea, calling a Catalina task more than once, badly set Ant tasks depends chains may cause that a task be called more than once in the same Ant run, even if not intended to. A bit of caution should be exercised when you are capturing output from that task, because this could lead to something unexpected:

- when capturing in a property you will find in it only the output from the *first* call, because Ant properties are immutable and once set they cannot be changed,
- when capturing in a file, each run will overwrite it and you will find in it only the *last* call output, unless you are using the `append="true"` attribute, in which case you will see the output of each task call appended to the file.
