tomcatmanager Documentation

Release 1.0.0

Jared Crapo

Feb 01, 2020

Contents

tomcatmanager is a command line tool and python library for managing a Tomcat server.

CHAPTER 1

What Can It Do?

This package installs a command line utility called `tomcat-manager`. It's easily scriptable using your favorite shell:

```
$ tomcat-manager --user=ace --password=newenglandclamchowder \
http://localhost:8080/manager deploy local sample.war /sampleapp
$ echo $?
0
```

There is also an interactive mode:

```
$ tomcat-manager
tomcat-manager>connect http://localhost:8080/manager ace newenglandclamchowder
--connected to http://localhost:8080/manager as ace
tomcat-manager>list
Path                    Status  Sessions Directory
----------------------- ------- -------- -----------------------------------
/                       running        0 ROOT
/sampleapp              stopped        0 sampleapp##9
/sampleapp              running        0 sampleapp##8
/host-manager           running        0 /usr/share/tomcat8-admin/host-manage
/manager                running        0 /usr/share/tomcat8-admin/manager
```

And for the ultimate in flexibility, you can use the python package directly:

```
>>> import tomcatmanager as tm
>>> tomcat = tm.TomcatManager()
>>> r = tomcat.connect(url='http://localhost:8080/manager',
... user='ace', password='newenglandclamchowder')
>>> tomcat.is_connected
True
>>> r = tomcat.stop('/someapp')
>>> r.status_code == tm.status_codes.ok
False
>>> r.status_message
'No context exists named /someapp'
```

CHAPTER 2

System Requirements

You'll need Python 3.5 or higher on macOS, Windows, or Linux.

The following Tomcat versions are supported:

- 7.0.x
- 8.0.x
- 8.5.x
- 9.0.x

CHAPTER 3

Table of Contents

3.1 Installation

You need Python 3.5 or higher. Install using `pip`:

```
pip install tomcatmanager
```

Works on Windows, macOS, and Linux.

Now what?

First you will need to *Configure Tomcat*.

If you are in a hurry to *get started with the command line tool*, type:

```
tomcat-manager -h
```

Or, you can start *writing your own python code which imports the package*.

3.2 Configure Tomcat

3.2.1 Supported Tomcat Versions

The following Tomcat versions are supported:

- 7.0.x
- 8.0.x
- 8.5.x
- 9.0.x

The operating system and Java Virtual Machine don't matter as long as Tomcat runs on it.

3.2.2 Authentication

This library and associated tools do their work via the Tomcat Manager web application included in the Tomcat distribution.

You will need the URL where the Tomcat Manager application is available. You can use the URL that points directly to the container, or the URL of a proxy like nginx or Apache HTTP Server you have deployed in front of Tomcat. TLS is recommended, but it works without if you must.

You will also need to configure authentication for a user, and grant that user permission to access the Tomcat Manager application. The full details of this procedure can be found in the Tomcat Manager Howto. A short summary is included here.

Configure a user in `tomcat-users.xml` and grant that user the `manager-script` role. The username and password can be anything of your choosing. Here's how to configure the user we will use as an example throughout this documentation:

```
<tomcat-users>
  ...
  <role rolename="manager-script"/>
  <user username="ace" password="newenglandclamchowder" roles="manager-script"/>
  ...
</tomcat-users>
```

3.3 Interactive Use

After installation, you will have a new tool called `tomcat-manager`. Run this with no command line arguments to invoke an interactive, line-oriented command interpreter:

```
$ tomcat-manager
tomcat-manager> connect http://localhost:8080/manager admin newenglandclamchowder
--connected to http://localhost:8080/manager as ace
tomcat-manager> list
Path                    Status  Sessions Directory
----------------------- ------- -------- -----------------------------------
/                       running        0 ROOT
/manager                running       14 /usr/share/tomcat7-admin/manager
/host-manager           running        0 /usr/share/tomcat7-admin/host-manager
tomcat-manager> exit
```

Use the `exit` or `quit` command to exit the interpreter and return to your operating system shell.

3.3.1 Built In Help

The interactive shell has a built-in list of all available commands:

```
tomcat-manager> help
tomcat-manager is a command line tool for managing a Tomcat server

Connecting to a Tomcat server
=============================================================
connect   Connect to a tomcat manager instance.
which     Show the url of the tomcat server you are connected to.

```

(continues on next page)

(continued from previous page)

```
Managing applications
============================================================
list            Show all installed applications.
deploy local    Deploy a local war file to the tomcat server.
deploy server   Deploy a war file to the tomcat server.
deploy context  Deploy a context xml file to the tomcat server.
redeploy        Undeploy an existing app and deploy a new one in its place.
undeploy        Remove an application at a given path from the tomcat server.
start           Start a deployed tomcat application that isn't running.
stop            Stop a tomcat application and leave it deployed on the server.
restart         Start and stop a tomcat application. Synonym for reload.
  reload        Synonym for 'restart'.
sessions        Show active sessions for a tomcat application.
expire          Expire idle sessions.

Server information
============================================================
findleakers          Show tomcat applications that leak memory.
resources            Show global JNDI resources configured in Tomcat.
serverinfo           Show information about the tomcat server.
sslconnectorciphers  Show SSL/TLS ciphers configured for each connector.
status               Show server status information in xml format.
threaddump           Show a jvm thread dump.
vminfo               Show diagnostic information about the jvm.

Settings, configuration, and tools
============================================================
config       Edit or show the location of the user configuration file.
edit         Edit a file in the preferred text editor.
exit_code    Show a number indicating the status of the previous command.
history      View, run, edit, and save previously entered commands.
py           Execute python commands.
pyscript     Run a file containing a python script.
set          Change program settings.
show         Show all settings or a specific setting.
  settings   Synonym for 'show'.
shell        Execute a command in the operating system shell.
shortcuts    Show shortcuts for other commands.

Other
============================================================
exit     Exit this program.
  quit   Synonym for 'exit'.
help     Show available commands, or help on a specific command.
version  Show the version number of this program.
license  Show the MIT license.
```

As well as help for each command:

```
tomcat-manager> help stop
usage: stop [-h] [-v VERSION] path

Stop a running tomcat application and leave it deployed on the server.

positional arguments:
  path                  The path part of the URL where the application is
                        deployed.
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  -v VERSION, --version VERSION
                        Optional version string of the application to stop. If
                        the application was deployed with a version string, it
                        must be specified in order to stop the application.
```

This document does not include detailed explanations of every command. It does show how to connect to a Tomcat server and deploy a war file, since there are quite a few options for both of those commands. For everything else, the built-in help should be sufficient.

3.3.2 Connect To A Tomcat Server

Before you can do anything to a Tomcat server, you need to enter the connection information, including the url and the authentication credentials. You can pass the connection information on the command line:

```
$ tomcat-manager --user=ace http://localhost:8080/manager
Password: {you type your password here}
```

Or:

```
$ tomcat-manager --user=ace --password=newenglandclamchowder \
http://localhost:8080/manager
```

You can also enter this information into the interactive prompt:

```
$ tomcat-manager
tomcat-manager> connect http://localhost:8080/manager ace newenglandclamchowder
```

Or:

```
$ tomcat-manager
tomcat-manager> connect http://localhost:8080/manager ace
Password: {type your password here}
```

3.3.3 Deploy applications

Tomcat applications are usually packaged as a WAR file, which is really just a zip file with a different extension. The `deploy` command sends a WAR file to the Tomcat server and tells it which URL to deploy that application at.

The WAR file can be located in one of two places: some path on the computer that is running Tomcat, or some path on the computer where the command line `tomcat-manager` program is running.

If the WAR file is located on the same server as Tomcat, we call that `server`. If the WAR file is located where `tomcat-manager` is running, we call that `local`. If the file is already on the server, then we have to tell Tomcat where to go find it. If it's `local`, then we have to send the WAR file over the network so Tomcat can deploy it.

For all of these examples, lets assume I have a Tomcat server running far away in a data center somewhere, accessible at `https://www.example.com`. I'm running the command line `tomcat-manager` program on my laptop. We'll also assume that we have already connected to the Tomcat server, using one of the methods just described in *Connect To A Tomcat Server*.

For our first example, let's assume we have a WAR file already on our server, in `/tmp/fancyapp.war`. To deploy this WAR file to `https://www.example.com/fancy`:

```
tomcat-manager> deploy server /tmp/myfancyapp.war /fancy
```

Now let's say I just compiled a WAR file on my laptop for an app called `shiny`. It's saved at `~/src/shiny/dist/shinyv2.0.5.war`. I'd like to deploy it to `https://www.example.com/shiny`:

```
tomcat-manager> deploy local ~/src/shiny/dist/shiny2.0.5.war /shiny
```

Sometimes when you deploy a WAR you want to specify additional configuration information. You can do so by using a context file. The context file must reside on the same server where Tomcat is running.

```
tomcat-manager> deploy context /tmp/context.xml /sample
```

This command will deploy the WAR file specified in the `docBase` attribute of the `Context` element so it's available at `https://www.example.com/sample`.

Note: When deploying via context files, be aware of the following:

- The `path` attribute of the `Context` element is ignored by the Tomcat Server when deploying from a context file.
- If the `Context` element specifies a `docBase` attribute, it will be used even if you specify a war file on the command line.

3.3.4 Parallel Deployment

Tomcat supports a parallel deployment feature which allows multiple versions of the same WAR to be deployed simultaneously at the same URL. To utilize this feature, you need to deploy an application with a version string. The combination of path and version string uniquely identify the application.

Let's revisit our `shiny` app. This time we will deploy with a version string:

```
tomcat-manager>deploy local ~/src/shiny/dist/shiny2.0.5.war /shiny -v v2.0.5
tomcat-manager>list
Path                    Status  Sessions Directory
----------------------- ------- -------- -----------------------------------
/                       running        0 ROOT
/manager                running        0 manager
/shiny                  running        0 shiny##v2.0.5
```

Later today, I make a bug fix to 'shiny', and build version 2.0.6 of the app. Parallel deployment allows me to deploy two versions of that app at the same path, and Tomcat will migrate users to the new version over time as their sessions expire in version 2.0.5.

```
tomcat-manager>deploy local ~/src/shiny/dist/shiny2.0.6.war /shiny -v v2.0.6
tomcat-manager>list
Path                    Status  Sessions Directory
----------------------- ------- -------- -----------------------------------
/                       running        0 ROOT
/manager                running        0 manager
/shiny                  running       12 shiny##v2.0.5
/shiny                  running        0 shiny##v2.0.6
```

Once all the sessions have been migrated to version 2.0.6, I can undeploy version 2.0.5:

```
tomcat-manager>undeploy /shiny --version v2.0.5
tomcat-manager>list
Path                     Status  Sessions Directory
------------------------ ------- -------- -------------------------------------
/                        running        0 ROOT
/manager                 running        0 manager
/shiny.                  running        9 shiny##v2.0.6
```

The following commands support the `-v` or `--version` option, which makes parallel deployment possible:

- deploy
- undeploy
- start
- stop
- reload
- sessions
- expire

3.3.5 Readline Editing

You can edit current or previous commands using standard `readline` editing keys. If you aren't familiar with `readline`, just know that you can use your arrow keys, `home` to move to the beginning of the line, `end` to move to the end of the line, and `delete` to forward delete characters.

3.3.6 Command History

Interactive mode keeps a command history, which you can navigate using the up and down arrow keys. and search the history of your commands with `<control>+r`.

You can view the list of previously issued commands:

```
tomcat-manager> history
```

And run a previous command by string search:

```
tomcat-manager> history -r undeploy
```

Or by number:

```
tomcat-manager> history -r 10
```

The `history` command has many other options, including the ability to save commands to a file and load commands from a file. Use `help history` to get the details.

3.3.7 Settings

The `show` or `settings` (they do exactly the same thing) commands display a list of settings which control the behavior of `tomcat-manager`:

```
tomcat-manager> show
autorun_on_edit=False       # Automatically run files after editing
colors=True                 # Colorized output (*nix only)
debug=False                 # Show stack trace for exceptions
echo=False                  # For piped input, echo command to output
editor=/usr/local/bin/zile  # Program used to edit files
locals_in_py=True           # Allow access to your application in py via self
prompt='tomcat-manager> '   # The prompt issued to solicit input
quiet=False                 # Don't print nonessential feedback
status_prefix=--            # String to prepend to all status output
status_to_stdout=False      # Status information to stdout instead of stderr
timeout=10                  # Seconds to wait for HTTP connections
timing=False                # Report execution times
```

You can change any of these settings using the `set` command:

```
tomcat-manager> set prompt='tm> '
tm>
```

Quotes around values are not required unless they contain spaces or other quotes.

3.3.8 Configuration File

`tomcat-manager` reads a user configuration file on startup. This file allows you to:

- change settings on startup
- define shortcuts for connecting to Tomcat servers

The location of the configuration file is different depending on your operating system. To see the location of the file:

```
tomcat-manager> config file
/Users/kotfu/Library/Application Support/tomcat-manager/tomcat-manager.ini
```

You can edit the file from within `tomcat-manager` too. Well, it really just launches the editor of your choice, you know, the one specified in the `editor` setting. Do that by typing:

```
tomcat-manager> config edit
```

This file uses the INI file format. If you create a section called `settings`, you can set the values of any of the available settings. My config file contains:

```
[settings]
prompt='tm> '
debug=True
editor=/usr/local/bin/zile
```

3.3.9 Server Shortcuts

You can also use the configuration file to set up shortcuts to various Tomcat servers. Define a section named the shortcut, and then include a property for `url`, `user`, and `password`. Here's a simple example:

```
[localhost]
url=http://localhost:8080/manager
user=ace
password=newenglandclamchowder
```

With this defined in your configuration file, you can now connect using the name of the shortcut:

```
tomcat-manager> connect localhost
```

If you define a `user`, but omit `password`, you will be prompted for it when you use the shortcut in the `connect` command.

3.3.10 Shell-style Output Redirection

Save the output of the `list` command to a file:

```
tomcat-manager> list > /tmp/tomcat-apps.txt
```

Search the output of the `vminfo` command:

```
tomcat-manager> vminfo | grep user.timezone
  user.timezone: US/Mountain
```

Or the particularly useful:

```
tomcat-manager> threaddump | less
```

3.3.11 Clipboard Integration

You can copy output to the clipboard by redirecting but not giving a filename:

```
tomcat-manager> list >
```

You can also append output to the clipboard using a similar method:

```
tomcat-manager> serverinfo >>
```

3.3.12 Run shell commands

Use the `shell` or `!` commands to execute operating system commands (how meta):

```
tomcat-manager> !ls
```

Of course tab completion works on shell commands.

3.3.13 Python Interpreter

You can launch a python interpreter:

```
tomcat-manager> py
Python 3.6.1 (default, Apr  4 2017, 09:40:51)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveTomcatManager)

    py <command>: Executes a Python command.
    py: Enters interactive Python mode.
```

(continues on next page)

(continued from previous page)

```
    End with ``Ctrl-D`` (Unix) / ``Ctrl-Z`` (Windows), ``quit()``, ``exit()``.
    Non-python commands can be issued with ``cmd("your command")``.
    Run python code from external script files with ``run("script.py")``

>>> self.tomcat
<tomcatmanager.tomcat_manager.TomcatManager object at 0x10f353550>
>>> self.tomcat.is_connected
True
>>> exit()
```

As you can see, if you have connected to a Tomcat server, then you will have a `self.tomcat` object available. See *Use from Python* for more information about what you can do with this object.

3.4 Command Line

You've already read about *Interactive Use* right? If not, this part will feel kind of hollow.

Say you want to find out how many active sessions there are in the oldest version of our `shiny` app (told you it would feel kind of hollow). You could use interactive mode:

```
$ tomcat-manager
tomcat-manager>connect https://www.example.com/manager ace newenglandclamchowder
--connected to https://www.example.com/manager as ace
tomcat-manager>list
Path                    Status  Sessions Directory
----------------------- ------- -------- -----------------------------------
/                       running        0 ROOT
/manager                running        0 manager
/shiny                  running       17 shiny##v2.0.6
/shiny                  running        6 shiny##v2.0.5
```

3.4.1 Using Shell Scripts

If you need to automate a more complex sequence of commands or parse the output, you might choose to use `tomcat-manager` from within a shell script:

```
#!/usr/bin/env bash
#
URL=https://www.example.com/manager
USERID=ace
PASSWD=newenglandclamchowder
COMMAND='list --raw'
TOMCAT="tomcat-manager --quiet --user=$USERID --password=$PASSWD $URL $COMMAND"

# get the output of the list into a shell variable
LIST=$($TOMCAT)

# if the tomcat command completed successfully
TOMCAT_EXIT=$?
if [ "$TOMCAT_EXIT" -eq 0 ]; then
    echo "$LIST" | grep '^/shiny' | awk -F ':' '{ print $4":"$3}' | \
    sort | head -1 | awk -F ':' '{ print #2 }'
else
```

(continues on next page)

(continued from previous page)

```
    # list has an error message, not the list of tomcat apps
    echo -n "$LIST"
    exit $TOMCAT_EXIT
fi
```

Save this script as `~/bin/oldshiners.sh`, and then run it:

```
$ ~/bin/oldshiners.sh
6
```

This script builds a `tomcat-manager` command which includes authentication credentials, the url where the Tomcat Manager web app is deployed, as well as the command from *Interactive Use*. In this example, we used `list` as our command. Any command that works in the interactive mode works on the command line.

Note how we check the exit code in the shell. `tomcat-manager` knows whether the command to the tomcat server completed successfully or not, and sets the shell exit code appropriately. The exit codes are:

0 = command completed succesfully
1 = command had an error
2 = incorrect usage
127 = unknown command

3.4.2 Server Shortcuts

You can use *Server Shortcuts* from the command line with or without commands:

```
$ tomcat-manager localhost
--connected to http://localhost:8080/manager as ace
tomcat-manager>list
Path                    Status  Sessions Directory
----------------------- ------- -------- -------------------------------------
/                       running        0 ROOT
/manager                running        0 manager
```

Or:

```
$ tomcat-manager localhost list
--connected to http://localhost:8080/manager as ace
Path                    Status  Sessions Directory
----------------------- ------- -------- -------------------------------------
/                       running        0 ROOT
/manager                running        0 manager
```

This mechanism allows you to keep all authentication credentials out of your scripts. Simply define shortcut(s) with credentials for the server(s) you want to manage, and reference the shortcuts in your scripts. Instead of this:

```
TOMCAT="tomcat-manager --user=$USERID --password=$PASSWD $URL $COMMAND"
```

you might use this:

```
TOMCAT="tomcat-manager example $COMMAND"
```

with the following in your configuration file:

```
[example]
url=https://www.example.com
user=ace
password=newenglandclamchowder
```

3.4.3 Piped Input

`tomcat-manager` will process lines from standard input as though they were entered at the interactive prompt. There is no mechanism to check for errors this way, the commands are blindly run until the pipe is closed. The shell exit code of `tomcat-manager` will be the exit code of the last command run.

If you want to see what the exit codes are, you can either use `$?` in your shell, or you can use the interactive command `exit_code` (`$?` works too) to see the result.

If you want more sophisticated error checking, then you should probably write a shell script and invoke `tomcat-manager` seperately for each command you want to execute. That will allow you to use the shell script for checking exit codes, logic branching, looping, etc.

3.4.4 Controlling Output

When using `tomcat-manager` from the command line, you have fine grained control of what you want included in the output. As a well-behaved shell program it sends output to `stdout` and errors to `stderr`. If you are using `bash` or one of the other `sh` variants, you can easily co-mingle them into a single stream:

```
$ tomcat-manager localhost list > myapps.txt 2>&1
```

In addition to redirecting with the shell, there are several command line switches that change what's included in the output. These options correspond to *Settings* you can change in *Interactive Use*. All of the settings default to `False`, but be aware that you may have altered them your *Configuration File*, which is read on startup.

Option	Setting	Description
`-e, --echo`	`echo`	Add the command to the output stream.
`-q, --quiet`	`quiet`	Don't show non-essential feedback.
`-s, --status-to-stdout`	`status_to_stdout`	Send status information to `stdout` instead of `stderr`.
`-d, --debug`	`debug`	Show detailed exception and stack trace, even if `quiet` is True.

Some commands show additional status information during their execution which is not part of the output. If `quiet=True` then all status output is suppressed. If `quiet=False` then status information is sent to `stderr`. If `status_to_stdout=True` then status information is sent to `stdout`, as long as `quiet=False`.

Here's a couple of examples to demonstrate, using a *server shortcut* of `localhost`, which we assume gets you authenticated to a Tomcat Server web application:

These two commands yield the same output, but by different mechanisms: the first one uses the shell to redirect status messages to the bitbucket, the second one uses the `--quiet` switch to instruct `tomcat-manager` to suppress status messages.

```
$ tomcat-manager localhost list 2>/dev/null
Path                   Status  Sessions Directory
---------------------- ------- -------- -----------------------------------
/                      running        0 ROOT
```

(continues on next page)

(continued from previous page)

```
/manager                  running        0 manager
$ tomcat-manager --quiet localhost list 2>/dev/null
Path                      Status  Sessions Directory
------------------------ ------- -------- -----------------------------------
/                         running        0 ROOT
/manager                  running        0 manager
```

If you pipe commands into `tomcat-manager` instead of providing them as arguments, the `--echo` command line switch can be included which will print the prompt and command to the output:

```
$ echo list | tomcat-manager --echo localhost
--connected to https://home.kotfu.net/manager as ace
tomcat-manager> list
Path                      Status  Sessions Directory
------------------------ ------- -------- -----------------------------------
/                         running        0 ROOT
/manager                  running        0 manager
```

For most common errors, like failed authorization, connection timeouts, and DNS lookup failures, `tomcat-manager` catches the exceptions raised by those errors, and outputs a terse message describing the problem. For example, if my Tomcat container is not currently running, or if the HTTP request fails for any other reason, you will see something like this:

```
$ tm vm list
connection error
```

If you want all the gory detail, give the `--debug` command line switch or set `debug=True`. Then you'll see something like this (stack trace truncated with '...'):

```
$ tm --debug vm list
Traceback (most recent call last):
  File "/Users/kotfu/.pyenv/versions/3.6.2/envs/tomcatmanager-3.6/lib/python3.6/site-
↪packages/urllib3/connection.py", line 141, in _new_conn
    (self.host, self.port), self.timeout, **extra_kw)
  File "/Users/kotfu/.pyenv/versions/3.6.2/envs/tomcatmanager-3.6/lib/python3.6/site-
↪packages/urllib3/util/connection.py", line 83, in create_connection
    raise err
  File "/Users/kotfu/.pyenv/versions/3.6.2/envs/tomcatmanager-3.6/lib/python3.6/site-
↪packages/urllib3/util/connection.py", line 73, in create_connection
    sock.connect(sa)
socket.timeout: timed out
...
requests.exceptions.ConnectTimeout: HTTPConnectionPool(host='192.168.13.66',␣
↪port=8080): Max retries exceeded with url: /manager/text/serverinfo (Caused by␣
↪ConnectTimeoutError(<urllib3.connection.HTTPConnection object at 0x103180a20>,
↪'Connection to 192.168.13.66 timed out. (connect timeout=2)'))
```

3.5 Use from Python

3.5.1 Connect to the server

Before you can do anything useful, you need to create a *TomcatManager* object and connect to a server.

TomcatManager.**connect**(*url: str, user: str = '', password: str = ''*) → tomcatmanager.models.TomcatManagerResponse

Connect to a Tomcat Manager server.

Parameters **url** – url where the Tomcat Manager web application is deployed :param user: (optional) user to authenticate with :param password: (optional) password to authenticate with :return: *TomcatManagerResponse()* object

You don't have to connect before using any other commands. If you initialized the object with credentials you can call any other method. This method:

- give you a way to change the credentials on an existing object
- provide a convenient mechanism to validate you can actually connect to the server
- allow you to inspect the response so you can see why you can't connect

Usage:

```
>>> import tomcatmanager as tm
>>> url = 'http://localhost:8080/manager'
>>> user = 'ace'
>>> password = 'newenglandclamchowder'
>>> tomcat = tm.TomcatManager()
>>> try:
...     r = tomcat.connect(url, user, password)
...     if r.ok:
...         print('connected')
...     else:
...         print('not connected')
... except Exception as err:
...    # handle exception
...    print('not connected')
not connected
```

The only way to validate whether we are connected is to make an HTTP request to the server and see if it returns successfully. Internally this method tries to retrieve `/manager/text/serverinfo`.

Requesting url's via http can raise all kinds of exceptions. For example, if you give a URL where no web server is listening, you'll get a `requests.connections.ConnectionError`. However, this method won't raise exceptions for everything. If the credentials are incorrect, you won't get an exception unless you ask for it.

Requesting url's via http can also result in redirection to another url. If that occurs, the new url, not the one you passed, will be stored in the url attribute.

You can also use `TomcatManager.is_connected()` to check if you are connected.

If you want to raise more exceptions see *TomcatManagerResponse.raise_for_status()*.

3.5.2 Responses from the server

All the methods of *TomcatManager* which interact with the server return a response in the form of a *TomcatManagerResponse* object. Use this object to check whether the command completed successfully, and to get any results generated by the command.

class tomcatmanager.models.**TomcatManagerResponse**(*response=None*)

Returned as the response for *TomcatManager* commands.

After running a command, it's a good idea to check and make sure that the command completed succesfully before relying on the results:

```
>>> import tomcatmanager as tm
>>> tomcat = getfixture('tomcat')
>>> try:
...     r = tomcat.server_info()
...     r.raise_for_status()
...     if r.ok:
...         print(r.server_info.os_name)
...     else:
...         print('Error: {}'.format(r.status_message))
... except Exception as err:
...     # handle exception
...     pass
Linux
```

ok

Returns True if the request completed with no errors.

For this property to return True:

- The HTTP request must return a status code of `200 OK`
- The first line of the response from the Tomcat Server must begin with `OK`.

raise_for_status()
Raise exceptions for server errors.

First this method calls `requests.Response.raise_for_status()` which raises exceptions if a 4xx or 5xx response is received from the server.

If that doesn't raise anything, then it raises a *TomcatError* if there is not an `OK` response from the first line of text back from the Tomcat Manager web app.

status_code
Status of the Tomcat Manager command from the first line of text.

The preferred way to check for success is to use the `ok()` method, because it checks for http errors as well as tomcat errors. However, if you want specific access to the status of the tomcat command, use this method.

There are three status codes:

- `OK`
- `FAIL`
- `NOTFOUND`

`tomcatmanager.status_codes` is a dictionary which makes it easy to check this code against known values. It also has attributes with friendly names, as shown here:

```
>>> import tomcatmanager as tm
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.server_info()
>>> r.status_code == tm.status_codes.ok
True
```

status_message
The message on the first line of the response from the Tomcat Server.

result
The text of the response from the Tomcat server, without the first line (which contains the status code and message).

response
The server's response to an HTTP request.

TomcatManager uses the excellent Requests package for HTTP communication. This property returns the `requests.models.Response` object which contains the server's response to the HTTP request.

Of particular use is `requests.models.Response.text` which contains the content of the response in unicode. If you want raw access to the content returned by the Tomcat Server, this is where you can get it.

3.5.3 Deploying applications

There are three methods you can use to deploy applications to a Tomcat server.

`TomcatManager.`**`deploy_localwar`**(*path: str*, *warfile: str*, *version: str = None*, *update: bool = False*) → tomcatmanager.models.TomcatManagerResponse

Deploy a warfile on the local file system to the Tomcat server.

Parameters

- **path** – The path on the server to deploy this war to, i.e. /sampleapp
- **warfile** – The path (specified using your local operating system convention) to a war file on the local file system. You can also pass a stream or file-like object. This will be sent to the server for deployment.
- **version** – (optional) For tomcat parallel deployments, the version string to associate with this deployment
- **update** – (optional) Whether to undeploy the existing path first (default False)

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is specified; if no warfile is specified

`TomcatManager.`**`deploy_serverwar`**(*path: str*, *warfile: str*, *version: str = None*, *update: bool = False*) → tomcatmanager.models.TomcatManagerResponse

Deploy a warfile on the local file system to the Tomcat server.

Parameters

- **path** – The path on the server to deploy this war to, i.e. /sampleapp
- **warfile** – The java-style path (use slashes not backslashes) to the war file on the server. Don't include `file:` at the beginning.
- **version** – (optional) For tomcat parallel deployments, the version string to associate with this deployment
- **update** – (optional) Whether to undeploy the existing path first (default False)

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is given; if no warfile is given

`TomcatManager.`**`deploy_servercontext`**(*path: str*, *contextfile: str*, *warfile: str = None*, *version: str = None*, *update: bool = False*) → tomcatmanager.models.TomcatManagerResponse

Deploy a Tomcat application defined by a context file.

Parameters

- **path** – The path on the server to deploy this war to, i.e. /sampleapp

- **contextfile** – The java-style path (use slashes not backslashes) to the context file on the server. Don't include `file:` at the beginning.
- **warfile** – (optional) The java-style path (use slashes not backslashes) to the war file on the server. Don't include `file:` at the beginning.
- **version** – (optional) For tomcat parallel deployments, the version string to associate with this deployment
- **update** – (optional) Whether to undeploy the existing path first (default False)

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is given; if no contextfile is given

You can also undeploy applications. This removes the WAR file from the Tomcat server.

`TomcatManager.`**`undeploy`**(*path: str*, *version: str = None*) → tomcatmanager.models.TomcatManagerResponse

Undeploy the application at a given path.

Parameters

- **path** – The path of the application to undeploy
- **version** – (optional) The version string of the app to undeploy

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is specified

If the application was deployed with a version string, it must be specified in order to undeploy the application.

3.5.4 Other application commands

`TomcatManager.`**`start`**(*path: str*, *version: str = None*) → tomcatmanager.models.TomcatManagerResponse

Start the application at a given path.

Parameters

- **path** – The path of the application to start
- **version** – (optional) The version string of the app to start

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is specified

If the application was deployed with a version string, it must be specified in order to start the application.

`TomcatManager.`**`stop`**(*path: str*, *version: str = None*) → tomcatmanager.models.TomcatManagerResponse

Stop the application at a given path.

Parameters

- **path** – The path of the application to stop
- **version** – (optional) The version string of the app to stop

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is specified

If the application was deployed with a version string, it must be specified in order to stop the application.

TomcatManager.**reload**(*path: str, version: str = None*) → tomcatmanager.models.TomcatManagerResponse

Reload (stop and start) the application at a given path.

Parameters

- **path** – The path of the application to reload
- **version** – (optional) The version string of the app to reload

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is specified

If the application was deployed with a version string, it must be specified in order to reload the application.

TomcatManager.**sessions**(*path: str, version: str = None*) → tomcatmanager.models.TomcatManagerResponse

Get the age of the sessions in an application.

Parameters

- **path** – The path of the application to get session information about
- **version** – (optional) The version string of the app to get session information about

Returns *TomcatManagerResponse* object with the session summary in both the `result` attribute and the `sessions` attribute

Raises **ValueError** – if no path is specified

Usage:

```
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.sessions('/manager')
>>> if r.ok:
...     session_data = r.sessions
```

TomcatManager.**expire**(*path: str, version: str = None, idle: Any = None*) → tomcatmanager.models.TomcatManagerResponse

Expire sessions idle for longer than idle minutes.

Parameters

- **path** – the path to the app on the server whose sessions you want to expire
- **idle** – sessions idle for more than this number of minutes will be expired. Use idle=0 to expire all sessions.

Returns *TomcatManagerResponse* object with the session summary in both the `result` attribute and the `sessions` attribute

Raises **ValueError** – if no path is specified

Usage:

```
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.expire('/manager', idle=15)
>>> if r.ok:
...     expiration_data = r.sessions
```

TomcatManager.**list**() → tomcatmanager.models.TomcatManagerResponse

Get a list of all applications currently installed.

Returns *TomcatManagerResponse* object with an additional `apps` attribute which contains a list of *TomcatApplication* objects

Usage:

```
>>> import tomcatmanager as tm
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.list()
>>> if r.ok:
...     running = filter(lambda app: app.state == tm.application_states.running,␣
↪r.apps)
```

3.5.5 Parallel Deployment

Tomcat supports a parallel deployment feature which allows multiple versions of the same WAR to be deployed simultaneously at the same URL. To utilize this feature, you need to deploy an application with a version string. The combination of path and version string uniquely identify the application:

```
>>> tomcat = getfixture('tomcat')
>>> safe_path = getfixture('safe_path')
>>> localwar_file = getfixture('localwar_file')
>>> with open(localwar_file, 'rb') as localwar_fileobj:
...     r = tomcat.deploy_localwar(safe_path, localwar_fileobj, version='42')
...     r.ok
True
>>> with open(localwar_file, 'rb') as localwar_fileobj:
...     r = tomcat.deploy_localwar(safe_path, localwar_fileobj, version='43')
...     r.ok
True
```

We now have two instances of the same application, deployed at the same location, but with different version strings. To do anything to either of those applications, you must supply both the path and the version string:

```
>>> r = tomcat.stop(path=safe_path, version='42')
>>> r.ok
True
>>> r = tomcat.undeploy(path=safe_path, version='42')
>>> r.ok
True
>>> r = tomcat.undeploy(path=safe_path, version='43')
>>> r.ok
True
```

The following methods include an optional version parameter to support parallel deployments:

- *deploy_localwar()*
- *deploy_serverwar()*
- *deploy_servercontext()*
- *undeploy()*
- *start()*
- *stop()*
- *reload()*
- *sessions()*
- *expire()*

3.5.6 Information about Tomcat

There are a number of methods which just return information about the Tomcat server. With the exception of *find_leakers()* (which triggers garbage collection), these methods don't effect any change on the server.

TomcatManager.**server_info**() → tomcatmanager.models.TomcatManagerResponse
Get information about the Tomcat server.

Returns *TomcatManagerResponse* object with an additional server_info attribute

The server_info attribute contains a *ServerInfo* object, which is a dictionary with some added properties for well-known values returned from the Tomcat server.

Usage:

```
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.server_info()
>>> if r.ok:
...     r.server_info['OS Name'] == r.server_info.os_name
True
```

TomcatManager.**status_xml**() → tomcatmanager.models.TomcatManagerResponse
Get server status information in XML format.

Returns *TomcatManagerResponse* object with an additional status_xml attribute

Usage:

```
>>> import xml.etree.ElementTree as ET
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.status_xml()
>>> if r.ok:
...     root = ET.fromstring(r.status_xml)
...     mem = root.find('jvm/memory')
...     print('Free Memory = {}'.format(mem.attrib['free']))
Free Memory ...
```

Tomcat 8.0 doesn't include application info in the XML, even though the docs say it does.

TomcatManager.**vm_info**() → tomcatmanager.models.TomcatManagerResponse
Get diagnostic information about the JVM.

Returns *TomcatManagerResponse* object with an additional vm_info attribute

TomcatManager.**ssl_connector_ciphers**() → tomcatmanager.models.TomcatManagerResponse
Get SSL/TLS ciphers configured for each connector.

Returns *TomcatManagerResponse* object with an additional ssl_connector_ciphers attribute

TomcatManager.**thread_dump**() → tomcatmanager.models.TomcatManagerResponse
Get a jvm thread dump.

Returns *TomcatManagerResponse* object with an additional thread_dump attribute

TomcatManager.**resources**(*type_: str = None*) → tomcatmanager.models.TomcatManagerResponse
Get the global JNDI resources available for use in resource links for context config files

Parameters **type** – (optional) Fully qualified java class name of the resource type you are interested in. For example, pass javax.sql.DataSource to acquire the names of all available JDBC data sources.

Returns *TomcatManagerResponse* object with an additional resources attribute.

Usage:

```
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.resources()
>>> if r.ok:
...     print(r.resources)
{'UserDatabase': 'org.apache.catalina.users.MemoryUserDatabase'}
```

`resources` is a dictionary with the resource name as the key and the class name as the value.

`TomcatManager.`**`find_leakers`**`()` → tomcatmanager.models.TomcatManagerResponse

Get apps that leak memory.

Returns *`TomcatManagerResponse`* object with an additional `leakers` attribute

The `leakers` attribute contains a list of paths of applications which leak memory.

This command triggers a full garbage collection on the server. Use with extreme caution on production systems.

Explicity triggering a full garbage collection from code is documented to be unreliable. Furthermore, depending on the jvm, there are options to disable explicit GC triggering, like `-XX:+DisableExplicitGC`. If you want to make sure this command triggered a full GC, you will have to verify using something like GC logging or JConsole.

The Tomcat Manager documentation says the server can return duplicates in this list if the app has been reloaded and was leaking both before and after the reload. The list returned by the `leakers` attribute will have no duplicates in it.

Usage:

```
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.find_leakers()
>>> if r.ok:
...     cnt = len(r.leakers)
... else:
...     cnt = 0
```

3.6 API Documentation

Python API documentation for tomcatmanager 1.0.0

tomcatmanager is a command line tool and python library for managing a Tomcat server.

The most important class in the package is *`TomcatManager`*. This class connects to a Tomcat Manager web application, allows you to run various commands, and returns the responses to you as an instance of *`TomcatManagerResponse`*.

The interactive command line program `tomcat-manager` provided by this package is an instance of *`InteractiveTomcatManager`*.

Here's the classes defined by this package:

3.6.1 TomcatManager

`class` `tomcatmanager.tomcat_manager.`**`TomcatManager`**

A class for interacting with the Tomcat Manager web application.

Here's a summary of the recommended way to use this class with proper exception and error handling. For this example, we'll use the *server_info()* method.

```
>>> import tomcatmanager as tm
>>> url = 'http://localhost:8080/manager'
>>> user = 'ace'
>>> password = 'newenglandclamchowder'
>>> tomcat = tm.TomcatManager()
>>> try:
...     r = tomcat.connect(url, user, password)
...     if r.ok:
...         r = tomcat.server_info()
...         if r.ok:
...             print(r.server_info)
...         else:
...             print('Error: {}'.format(r.status_message))
...     else:
...         print('Error: not connected')
... except Exception as err:
...     # handle exception
...     print('Error: not connected')
Error: not connected
```

connect(*url: str, user: str = '', password: str = ''*) → tomcatmanager.models.TomcatManagerResponse

Connect to a Tomcat Manager server.

Parameters **url** – url where the Tomcat Manager web application is deployed :param user: (optional) user to authenticate with :param password: (optional) password to authenticate with :return: *TomcatManagerResponse()* object

You don't have to connect before using any other commands. If you initialized the object with credentials you can call any other method. This method:

- give you a way to change the credentials on an existing object
- provide a convenient mechanism to validate you can actually connect to the server
- allow you to inspect the response so you can see why you can't connect

Usage:

```
>>> import tomcatmanager as tm
>>> url = 'http://localhost:8080/manager'
>>> user = 'ace'
>>> password = 'newenglandclamchowder'
>>> tomcat = tm.TomcatManager()
>>> try:
...     r = tomcat.connect(url, user, password)
...     if r.ok:
...         print('connected')
...     else:
...         print('not connected')
... except Exception as err:
...    # handle exception
...    print('not connected')
not connected
```

The only way to validate whether we are connected is to make an HTTP request to the server and see if it returns successfully. Internally this method tries to retrieve `/manager/text/serverinfo`.

Requesting url's via http can raise all kinds of exceptions. For example, if you give a URL where no web server is listening, you'll get a `requests.connections.ConnectionError`. However, this method won't raise exceptions for everything. If the credentials are incorrect, you won't get an exception unless you ask for it.

Requesting url's via http can also result in redirection to another url. If that occurs, the new url, not the one you passed, will be stored in the url attribute.

You can also use `TomcatManager.is_connected()` to check if you are connected.

If you want to raise more exceptions see *TomcatManagerResponse.raise_for_status()*.

is_connected
Does the url point to an actual tomcat server and are the credentials valid?

Returns True if connected to a tomcat server, otherwise, False.

deploy_localwar(*path: str*, *warfile: str*, *version: str = None*, *update: bool = False*) → tomcatmanager.models.TomcatManagerResponse
Deploy a warfile on the local file system to the Tomcat server.

Parameters

- **path** – The path on the server to deploy this war to, i.e. /sampleapp
- **warfile** – The path (specified using your local operating system convention) to a war file on the local file system. You can also pass a stream or file-like object. This will be sent to the server for deployment.
- **version** – (optional) For tomcat parallel deployments, the version string to associate with this deployment
- **update** – (optional) Whether to undeploy the existing path first (default False)

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is specified; if no warfile is specified

deploy_serverwar(*path: str*, *warfile: str*, *version: str = None*, *update: bool = False*) → tomcatmanager.models.TomcatManagerResponse
Deploy a warfile on the local file system to the Tomcat server.

Parameters

- **path** – The path on the server to deploy this war to, i.e. /sampleapp
- **warfile** – The java-style path (use slashes not backslashes) to the war file on the server. Don't include `file:` at the beginning.
- **version** – (optional) For tomcat parallel deployments, the version string to associate with this deployment
- **update** – (optional) Whether to undeploy the existing path first (default False)

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is given; if no warfile is given

deploy_servercontext(*path: str*, *contextfile: str*, *warfile: str = None*, *version: str = None*, *update: bool = False*) → tomcatmanager.models.TomcatManagerResponse
Deploy a Tomcat application defined by a context file.

Parameters

- **path** – The path on the server to deploy this war to, i.e. /sampleapp

- **contextfile** – The java-style path (use slashes not backslashes) to the context file on the server. Don't include `file:` at the beginning.
- **warfile** – (optional) The java-style path (use slashes not backslashes) to the war file on the server. Don't include `file:` at the beginning.
- **version** – (optional) For tomcat parallel deployments, the version string to associate with this deployment
- **update** – (optional) Whether to undeploy the existing path first (default False)

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is given; if no contextfile is given

undeploy(*path: str*, *version: str = None*) → tomcatmanager.models.TomcatManagerResponse
Undeploy the application at a given path.

Parameters

- **path** – The path of the application to undeploy
- **version** – (optional) The version string of the app to undeploy

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is specified

If the application was deployed with a version string, it must be specified in order to undeploy the application.

start(*path: str*, *version: str = None*) → tomcatmanager.models.TomcatManagerResponse
Start the application at a given path.

Parameters

- **path** – The path of the application to start
- **version** – (optional) The version string of the app to start

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is specified

If the application was deployed with a version string, it must be specified in order to start the application.

stop(*path: str*, *version: str = None*) → tomcatmanager.models.TomcatManagerResponse
Stop the application at a given path.

Parameters

- **path** – The path of the application to stop
- **version** – (optional) The version string of the app to stop

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is specified

If the application was deployed with a version string, it must be specified in order to stop the application.

reload(*path: str*, *version: str = None*) → tomcatmanager.models.TomcatManagerResponse
Reload (stop and start) the application at a given path.

Parameters

- **path** – The path of the application to reload
- **version** – (optional) The version string of the app to reload

Returns *TomcatManagerResponse* object

Raises **ValueError** – if no path is specified

If the application was deployed with a version string, it must be specified in order to reload the application.

sessions(*path: str*, *version: str = None*) → tomcatmanager.models.TomcatManagerResponse
Get the age of the sessions in an application.

Parameters

- **path** – The path of the application to get session information about
- **version** – (optional) The version string of the app to get session information about

Returns *TomcatManagerResponse* object with the session summary in both the `result` attribute and the `sessions` attribute

Raises **ValueError** – if no path is specified

Usage:

```
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.sessions('/manager')
>>> if r.ok:
...     session_data = r.sessions
```

expire(*path: str*, *version: str = None*, *idle: Any = None*) → tomcatmanager.models.TomcatManagerResponse
Expire sessions idle for longer than idle minutes.

Parameters

- **path** – the path to the app on the server whose sessions you want to expire
- **idle** – sessions idle for more than this number of minutes will be expired. Use idle=0 to expire all sessions.

Returns *TomcatManagerResponse* object with the session summary in both the `result` attribute and the `sessions` attribute

Raises **ValueError** – if no path is specified

Usage:

```
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.expire('/manager', idle=15)
>>> if r.ok:
...     expiration_data = r.sessions
```

list() → tomcatmanager.models.TomcatManagerResponse
Get a list of all applications currently installed.

Returns *TomcatManagerResponse* object with an additional `apps` attribute which contains a list of *TomcatApplication* objects

Usage:

```
>>> import tomcatmanager as tm
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.list()
>>> if r.ok:
...     running = filter(lambda app: app.state == tm.application_states.
↪running, r.apps)
```

server_info() → tomcatmanager.models.TomcatManagerResponse
Get information about the Tomcat server.

Returns *TomcatManagerResponse* object with an additional `server_info` attribute

The `server_info` attribute contains a *ServerInfo* object, which is a dictionary with some added properties for well-known values returned from the Tomcat server.

Usage:

```
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.server_info()
>>> if r.ok:
...     r.server_info['OS Name'] == r.server_info.os_name
True
```

status_xml() → tomcatmanager.models.TomcatManagerResponse
Get server status information in XML format.

Returns *TomcatManagerResponse* object with an additional `status_xml` attribute

Usage:

```
>>> import xml.etree.ElementTree as ET
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.status_xml()
>>> if r.ok:
...     root = ET.fromstring(r.status_xml)
...     mem = root.find('jvm/memory')
...     print('Free Memory = {}'.format(mem.attrib['free']))
Free Memory ...
```

Tomcat 8.0 doesn't include application info in the XML, even though the docs say it does.

vm_info() → tomcatmanager.models.TomcatManagerResponse
Get diagnostic information about the JVM.

Returns *TomcatManagerResponse* object with an additional `vm_info` attribute

ssl_connector_ciphers() → tomcatmanager.models.TomcatManagerResponse
Get SSL/TLS ciphers configured for each connector.

Returns *TomcatManagerResponse* object with an additional `ssl_connector_ciphers` attribute

thread_dump() → tomcatmanager.models.TomcatManagerResponse
Get a jvm thread dump.

Returns *TomcatManagerResponse* object with an additional `thread_dump` attribute

resources(*type_: str = None*) → tomcatmanager.models.TomcatManagerResponse
Get the global JNDI resources available for use in resource links for context config files

Parameters **type** – (optional) Fully qualified java class name of the resource type you are interested in. For example, pass `javax.sql.DataSource` to acquire the names of all available JDBC data sources.

Returns *TomcatManagerResponse* object with an additional `resources` attribute.

Usage:

```
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.resources()
>>> if r.ok:
...     print(r.resources)
{'UserDatabase': 'org.apache.catalina.users.MemoryUserDatabase'}
```

`resources` is a dictionary with the resource name as the key and the class name as the value.

find_leakers() → tomcatmanager.models.TomcatManagerResponse
Get apps that leak memory.

Returns *TomcatManagerResponse* object with an additional `leakers` attribute

The `leakers` attribute contains a list of paths of applications which leak memory.

This command triggers a full garbage collection on the server. Use with extreme caution on production systems.

Explicity triggering a full garbage collection from code is documented to be unreliable. Furthermore, depending on the jvm, there are options to disable explicit GC triggering, like `-XX:+DisableExplicitGC`. If you want to make sure this command triggered a full GC, you will have to verify using something like GC logging or JConsole.

The Tomcat Manager documentation says the server can return duplicates in this list if the app has been reloaded and was leaking both before and after the reload. The list returned by the `leakers` attribute will have no duplicates in it.

Usage:

```
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.find_leakers()
>>> if r.ok:
...     cnt = len(r.leakers)
... else:
...     cnt = 0
```

3.6.2 TomcatManagerResponse

class tomcatmanager.models.**TomcatManagerResponse**(*response=None*)
Returned as the response for *TomcatManager* commands.

After running a command, it's a good idea to check and make sure that the command completed succesfully before relying on the results:

```
>>> import tomcatmanager as tm
>>> tomcat = getfixture('tomcat')
>>> try:
...     r = tomcat.server_info()
...     r.raise_for_status()
...     if r.ok:
...         print(r.server_info.os_name)
...     else:
...         print('Error: {}'.format(r.status_message))
... except Exception as err:
...     # handle exception
...     pass
Linux
```

ok

Returns True if the request completed with no errors.

For this property to return True:

- The HTTP request must return a status code of `200 OK`
- The first line of the response from the Tomcat Server must begin with `OK`.

raise_for_status()
Raise exceptions for server errors.

First this method calls `requests.Response.raise_for_status()` which raises exceptions if a 4xx or 5xx response is received from the server.

If that doesn't raise anything, then it raises a *TomcatError* if there is not an `OK` response from the first line of text back from the Tomcat Manager web app.

status_code
Status of the Tomcat Manager command from the first line of text.

The preferred way to check for success is to use the `ok()` method, because it checks for http errors as well as tomcat errors. However, if you want specific access to the status of the tomcat command, use this method.

There are three status codes:

- `OK`
- `FAIL`
- `NOTFOUND`

`tomcatmanager.status_codes` is a dictionary which makes it easy to check this code against known values. It also has attributes with friendly names, as shown here:

```
>>> import tomcatmanager as tm
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.server_info()
>>> r.status_code == tm.status_codes.ok
True
```

status_message
The message on the first line of the response from the Tomcat Server.

result
The text of the response from the Tomcat server, without the first line (which contains the status code and message).

response
The server's response to an HTTP request.

TomcatManager uses the excellent Requests package for HTTP communication. This property returns the `requests.models.Response` object which contains the server's response to the HTTP request.

Of particular use is `requests.models.Response.text` which contains the content of the response in unicode. If you want raw access to the content returned by the Tomcat Server, this is where you can get it.

3.6.3 TomcatApplication

class tomcatmanager.models.**TomcatApplication**
Discrete data about an application running inside a Tomcat Server.

A list of these objects is returned by *TomcatManager.list()*.

classmethod sort_by_state_by_path_by_version(*app: TA*)
Function to create a key usable by `sort` to sort by state, by path, by version.

classmethod sort_by_path_by_version_by_state(*app: TA*)
Function to create a key usable by `sort` to sort by path, by version, by state

parse(*line: str*)
Parse a line from the server into this object.

Param line - the line of text from Tomcat Manager describing a deployed application

Tomcat Manager outputs a line like this for each application:

```
/shiny:running:0:shiny##v2.0.6
```

The data elements in this line can be described as:

```
{path}:{state}:{sessions}:{directory}##{version}
```

Where version and the two hash marks that precede it are optional.

path
The context path, or relative URL, where this app is available on the server.

state
The current state of the application.

`tomcatmanager.application_states` is a dictionary of all the valid values for this property. In addition to being a dictionary, it also has attributes for each possible state:

```
>>> import tomcatmanager as tm
>>> tm.application_states['stopped']
'stopped'
>>> tm.application_states.running
'running'
```

sessions
The number of currently active sessions.

directory
The directory on the server where this application resides.

version
The version of the application given when it was deployed.

If deployed without a version, this property returns None.

directory_and_version
Combine directory and version together.

Tomcat provides this information as `{directory}` if there was no version specified when the application was deployed, or `{directory}##{version}` if the version was specified.

This method has the logic to determine if version was specified or not.

3.6.4 ServerInfo

class tomcatmanager.models.**ServerInfo**(**args*, ***kwargs*)

Discrete data about the Tomcat server.

This object is a dictionary of keys and values as returned from the Tomcat server. It also has properties for well-known values.

Usage:

```
>>> tomcat = getfixture('tomcat')
>>> r = tomcat.server_info()
>>> r.server_info['OS Architecture']
'...'
>>> r.server_info.jvm_vendor
'...'
```

tomcat_version

The tomcat version string.

os_name

The operating system name.

os_version

The operating system version.

os_architecture

The operating system architecture.

jvm_version

The java virtual machine version string.

jvm_vendor

The java virtual machine vendor.

3.6.5 InteractiveTomcatManager

class tomcatmanager.interactive_tomcat_manager.**InteractiveTomcatManager**

An interactive command line tool for the Tomcat Manager web application.

each command sets the value of the instance variable exit_code, which mirrors bash standard values for exit codes (available via $?)

EXIT_CODES = {0: 'success', 1: 'error', 2: 'usage', 127: 'command_not_found'}

BOOLEAN_VALUES = {'0': False, '1': True, 'f': False, 'false': False, 'n': False,

exit_codes = {'command_not_found': 127, 'error': 1, 'success': 0, 'usage': 2}

app_name = 'tomcat-manager'

app_author = 'tomcatmanager'

config = None

timeout = 10

status_prefix = '--'

status_to_stdout

Proxy property for feedback_to_output.

poutput(*msg: Any = ''*, *, *end: str = '\n'*) → None
Convenient shortcut for self.stdout.write(); by default adds newline to end if not already present.

Also handles BrokenPipeError exceptions for when a commands's output has been piped to another process and that process terminates before the cmd2 command is finished executing.

Parameters

- **msg** – str - message to print to current stdout - anyting convertible to a str with '{}'.format() is OK
- **end** – str - string appended after the end of the message if not already present, default a newline

perror(*msg: Any = ''*, *, *end: str = '\n'*) → None
Print an error message or an exception.

Param errmsg The error message to print. If None, then print information about the exception currently beging handled.

Param exception_type From superclass. Ignored here.

Param traceback_war From superclass. Ignored here.

If debug=True, you will get a full stack trace, otherwise just the exception.

pfeedback(*msg: Any*, *, *end: str = '\n'*) → None
Print nonessential feedback.

Set quiet=True to supress all feedback. If feedback_to_output=True, then feedback will be included in the output stream. Otherwise, it will be sent to sys.stderr.

emptyline()
Do nothing on an empty line

default(*statement: cmd2.parsing.Statement*)
what to do if we don't recognize the command the user entered

docmd(*func: Callable*, **args*, ***kwargs*) → Any
Call a function and return, printing any exceptions that occur

Sets exit_code to 0 and calls {func}. If func throws a TomcatError, set exit_code to 1 and print the exception

show_help_from(*argparser: argparse.ArgumentParser*)
Set exit code and output help from an argparser.

parse_args(*parser: argparse.ArgumentParser, argv: List[T]*) → argparse.Namespace
Use argparse to parse a list of arguments a-la sys.argv

do_help(*args: str*)
Show available commands, or help on a specific command.

config_parser = ArgumentParser(prog='config', usage=None, description='Edit or show th

do_config(*cmdline: cmd2.parsing.Statement*)
Edit or show the location of the user configuration file.

help_config()
Show help for the 'config' command.

show_parser = ArgumentParser(prog='show', usage=None, description='Show all settings o

do_show(*cmdline: cmd2.parsing.Statement*)
Show all settings or a specific setting.

help_show()
Show help for the 'show' command.

settings_parser = ArgumentParser(prog='settings', usage=None, description="Show all se

do_settings(*cmdline: cmd2.parsing.Statement*)
Synonym for 'show' command.

help_settings()
Show help for the 'settings' command.

do_set(*args: cmd2.parsing.Statement*)
Change program settings.

help_set()
Show help for the 'set' command.

config_file
The location of the user configuration file.

Returns The full path to the user configuration file, or None if self.appdirs has not been defined.

history_file
The location of the command history file.

Returns The full path to the file where command history will be saved and loaded, or None if self.appdirs has not been defined.

load_config()
Open and parse the user config file and set self.config.

convert_to_boolean(*value: Any*)
Return a boolean value translating from other types if necessary.

connect_parser = ArgumentParser(prog='connect', usage='%(prog)s [-h] config_name\n %(p

do_connect(*cmdline: cmd2.parsing.Statement*)
Connect to a tomcat manager instance.

help_connect()
Show help for the connect command.

which_parser = ArgumentParser(prog='which', usage=None, description='show the url of t

do_which(**args*, ***kwargs*)
Show the url of the tomcat server you are connected to.

help_which()
Show help for the 'which' command.

deploy_local(*args: argparse.Namespace*, *update: bool = False*)
Deploy a local war file to the tomcat server.

deploy_server(*args: argparse.Namespace*, *update: bool = False*)
Deploy a war file to the tomcat server.

deploy_context(*args: argparse.Namespace*, *update: bool = False*)
Deploy a context xml file to the tomcat server.

deploy_parser = ArgumentParser(prog='deploy', usage=None, description='deploy an appli

do_deploy(**args*, ***kwargs*)
Deploy an application to the tomcat server.

help_deploy()
Show help for the deploy command.

redeploy_parser = ArgumentParser(prog='redeploy', usage=None, description='deploy an ap

do_redeploy(**args*, ***kwargs*)
Redeploy an application to the tomcat server.

help_redeploy()
Show help for the redeploy command.

undeploy_parser = ArgumentParser(prog='undeploy', usage=None, description='Remove an ap

do_undeploy(**args*, ***kwargs*)
Remove an application from the tomcat server.

help_undeploy()
Help for the 'undeploy' command.

start_parser = ArgumentParser(prog='start', usage=None, description="Start a tomcat app

do_start(**args*, ***kwargs*)
Start a deployed tomcat application that isn't running.

help_start()
Help for the 'start' command.

stop_parser = ArgumentParser(prog='stop', usage=None, description='Stop a running tomca

do_stop(**args*, ***kwargs*)
Stop a tomcat application and leave it deployed on the server.

help_stop()
Help for the 'stop' command.

reload_parser = ArgumentParser(prog='reload', usage=None, description="Start and stop a

do_reload(**args*, ***kwargs*)
Start and stop a tomcat application.

help_reload()
Help for the 'reload' command.

restart_parser = ArgumentParser(prog='restart', usage=None, description='Start and stop

do_restart(**args*, ***kwargs*)
Start and stop a tomcat application.

help_restart()
Show help for the 'restart' command.

sessions_parser = ArgumentParser(prog='sessions', usage=None, description='Show active

do_sessions(**args*, ***kwargs*)
Show active sessions for a tomcat application.

help_sessions()
Help for the 'sessions' command.

expire_parser = ArgumentParser(prog='expire', usage=None, description='expire idle sess

do_expire(**args*, ***kwargs*)
Expire idle sessions.

help_expire()
Help for the 'expire' command.

list_parser = ArgumentParser(prog='list', usage=None, description='Show all installed

do_list(**args*, ***kwargs*)
Show all installed applications.

help_list()
Show help for the 'list' command.

serverinfo_parser = ArgumentParser(prog='serverinfo', usage=None, description='show in

do_serverinfo(**args*, ***kwargs*)
Show information about the tomcat server.

help_serverinfo()
Show help for the 'serverinfo' command.

status_parser = ArgumentParser(prog='status', usage=None, description='show server sta

do_status(**args*, ***kwargs*)
Show server status information in xml format.

help_status()
Show help for the 'status' command.

vminfo_parser = ArgumentParser(prog='vminfo', usage=None, description='show diagnostic

do_vminfo(**args*, ***kwargs*)
Show diagnostic information about the jvm.

help_vminfo()
Show help for the 'vminfo' command.

sslconnectorciphers_parser = ArgumentParser(prog='sslconnectorciphers', usage=None, de

do_sslconnectorciphers(**args*, ***kwargs*)
Show SSL/TLS ciphers configured for each connector.

help_sslconnectorciphers()
Show help for the 'sslconnectorciphers' command.

threaddump_parser = ArgumentParser(prog='threaddump', usage=None, description='show a

do_threaddump(**args*, ***kwargs*)
Show a jvm thread dump.

help_threaddump()
Show help for the 'threaddump' command.

resources_parser = ArgumentParser(prog='resources', usage=None, description='show globa

do_resources(**args*, ***kwargs*)
Show global JNDI resources configured in Tomcat.

help_resources()
Show help for the 'resources' command.

findleakers_parser = ArgumentParser(prog='findleakers', usage=None, description='show

do_findleakers(**args*, ***kwargs*)
Show tomcat applications that leak memory.

help_findleakers()
Show help for the 'findleakers' command.

do_exit(*_*)
Exit the interactive command prompt.

do_quit(*cmdline: cmd2.parsing.Statement*)
Synonym for the 'exit' command.

do_eof(*cmdline: cmd2.parsing.Statement*)
Exit on the end-of-file character.

version_parser = ArgumentParser(prog='version', usage=None, description='show the vers

do_version(*cmdline: cmd2.parsing.Statement*)
Show the version number of this program.

help_version()
Show help for the 'version' command.

exit_code_epilog = ['The codes have the following meanings:', ' 0 success', ' 1 erro

exit_code_parser = ArgumentParser(prog='exit_code', usage=None, description='show a nu

do_exit_code(_)
Show a number indicating the status of the previous command.

help_exit_code()
Show help for the 'exit_code' command.

license_parser = ArgumentParser(prog='license', usage=None, description='show the soft

do_license(*cmdline: cmd2.parsing.Statement*)
Show the software license for this program.

help_license()
Show help for the 'license' command.

name = 'command_not_found'

number = 127

3.6.6 TomcatError

class tomcatmanager.models.**TomcatError**
Raised when the Tomcat Server responds with an error.

3.7 Contributing

3.7.1 Get Source Code

Clone the repo from github:

```
$ git clone git@github.com:tomcatmanager/tomcatmanager.git
```

3.7.2 Create Python Environments

tomcatamanger uses tox to run the test suite against multiple python versions. I recommend using pyenv with the pyenv-virtualenv plugin to manage these various versions. If you are a Windows user, `pyenv` won't work for you, you'll probably have to use conda.

This distribution includes a shell script `build-pyenvs.sh` which automates the creation of these environments.

If you prefer to create these virtual envs by hand, do the following:

```
$ cd tomcatmanager
$ pyenv install 3.8.0
$ pyenv virtualenv -p python3.8 3.8.0 tomcatmanager-3.8
$ pyenv install 3.7.5
$ pyenv virtualenv -p python3.7 3.7.5 tomcatmanager-3.7
$ pyenv install 3.6.9
$ pyenv virtualenv -p python3.6 3.6.9 tomcatmanager-3.6
$ pyenv install 3.5.8
$ pyenv virtualenv -p python3.5 3.5.8 tomcatmanager-3.5
```

Now set pyenv to make all four of those available at the same time:

```
$ pyenv local tomcatmanager-3.8 tomcatmanager-3.7 tomcatmanager-3.6 tomcatmanager-3.5
```

Whether you ran the script, or did it by hand, you now have isolated virtualenvs for each of the minor python versions. This table shows various python commands, the version of python which will be executed, and the virtualenv it will utilize.

Command	python	virtualenv
`python`	3.8.0	tomcatmanager-3.8
`python3`	3.8.0	tomcatmanager-3.8
`python3.8`	3.8.0	tomcatmanager-3.8
`python3.7`	3.7.5	tomcatmanager-3.7
`python3.6`	3.6.9	tomcatmanager-3.6
`python3.5`	3.5.8	tomcatmanager-3.5
`pip`	3.8.0	tomcatmanager-3.8
`pip3`	3.8.0	tomcatmanager-3.8
`pip3.8`	3.8.0	tomcatmanager-3.8
`pip3.7`	3.7.5	tomcatmanager-3.7
`pip3.6`	3.6.9	tomcatmanager-3.6
`pip3.5`	3.5.8	tomcatmanager-3.5

3.7.3 Install Dependencies

Now install all the development dependencies:

```
$ pip install -e .[dev]
```

This installs the tomcatmanager package "in-place", so the package points to the source code instead of copying files to the python `site-packages` folder.

All the dependencies now have been installed in the `tomcatmanager-3.8` virtualenv. If you want to work in other virtualenvs, you'll need to manually select it, and install again:

```
$ pyenv shell tomcatmanager-3.6
$ pip install -e .[dev]
```

3.7.4 Branches, Tags, and Versions

This project uses a simplified version of the git flow branching strategy. We don't use release branches, and we generally don't do hotfixes, so we don't have any of those branches either. The master branch always contains the latest release of the code uploaded to PyPI, with a tag for the version number of that release.

The develop branch is where all the action occurs. Feature branches are welcome. When it's time for a release, we merge develop into master.

This project uses semantic versioning.

3.7.5 Invoking Common Development Tasks

This project uses many other python modules for various development tasks, including testing, rendering documentation, and building and distributing releases. These modules can be configured many different ways, which can make it difficult to learn the specific incantations required for each project you are familiar with.

This project uses invoke to provide a clean, high level interface for these development tasks. To see the full list of functions available:

```
$ invoke -l
```

You can run multiple tasks in a single invocation, for example:

```
$ invoke clean docs sdist wheel
```

That one command will remove all superflous cache, testing, and build files, render the documentation, and build a source distribution and a wheel distribution.

You probably won't need to read further in this document unless you want more information about the specific tools used.

3.7.6 Testing

To ensure the tests can run without an external dependencies, `tests/mock_server80.py` contains a HTTP server which emulates the behavior of Tomcat Manager 8.0. There is a test fixture to start this server, and all the tests run against this fixture. I created this fixture to speed up testing time. It doesn't do everything a real Tomcat server does, but it's close enough for the tests to run, and it allows you to parallelize the test suite using `python-xdist`.

You can run the tests against all the supported versions of python using tox:

```
$ tox
```

tox expects that when it runs `python3.4` it will actually get a python from the 3.4.x series. That's why we set up the various python environments earlier.

If you just want to run the tests in your current python environment, use pytest:

```
$ pytest
```

This runs all the test in `tests/` and also runs doctests in `tomcatmanager/` and `docs/`.

You can speed up the test suite by using `pytest-xdist` to parallelize the tests across the number of cores you have:

```
$ pip install pytest-xdist
$ pytest -n8
```

In many of the doctests you'll see something like:

```
>>> tomcat = getfixture('tomcat')
```

This `getfixture()` helper imports fixtures defined in `conftest.py`, which has several benefits:

- reduces the amount of redundant code in doctests which shows connecting to a tomcat server and handling exceptions
- allows doctests to execute against a mock tomcat server

3.7.7 Testing Against A Real Server

If you wish, you can run the test suite against a real Tomcat Server instead of against the mock server included in this distribution. Running the test suite will deploy and undeploy an app hundreds of times, and will definitely trigger garbage collection, so you might not want to run it against a production server.

It's also slow (which is why the tests normally run against a mock server). When I run the test suite against a stock Tomcat on a Linode with 2 cores and 4GB of memory it takes approximately 3 minutes to complete. I don't think throwing more CPU at this would make it any faster: during the run of the test suite the Tomcat Server never consumes more than a few percent of the CPU capacity.

You must prepare some files on the server in order for the test suite to run successfully. Some of the tests instruct the Tomcat Server to deploy an application from a warfile stored on the server. I suggest you use the minimal application included in this distribution at `tomcatmanager/tests/war/sample.war`, but you can use any valid war file. Put this file in some directory on the server; I typically put it in `/tmp/sample.war`.

You must also construct a minimal context file on the server. You can see an example of such a context file in `tomcatmanager/tests/war/context.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Context configuration file for my web application -->
<Context path='/ignored' docBase='/tmp/sample.war'>
</Context>
```

The `docBase` attribute must point to a valid war file or the tests will fail. It can be the same minimal war file you already put on the server. The `path` attribute is ignored for context files that are not visible to Tomcat when it starts up, so it doesn't matter what you have there. I typically put this context file at `/tmp/context.xml`.

You will also need:

- the url where the manager app of your Tomcat Server is available
- a user with the `manager-script` role
- the password for the aforementioned user

With all these prerequisites ready, you can feed them to `pytest` as shown:

```
$ pytest --url=http://localhost:8080/manager --user=ace \
--password=newenglandclamchowder --warfile=/tmp/sample.war \
--contextfile=/tmp/context.xml
```

Warning: If you test against a real Tomcat server, you should not use the `pytest-xdist` plugin to parallelize testing across multiple CPUs or many platforms. Many of the tests depend on deploying and undeploying an app at a specific path, and that path is shared across the entire test suite. It wouldn't help much anyway because the testing is constrained by the speed of the Tomcat server.

If you kill the test suite in the middle of a run, you may leave the test application deployed in your tomcat server. If this happens, you must undeploy it before rerunning the test suite or you will get lots of errors.

When the test suite deploys applications, it will be at the path returned by the `safe_path` fixture in `conftest.py`. You can modify that fixture if for some reason you need to deploy at a different path.

3.7.8 Code Quality

Use `pylint` to check code quality. There is a pylint config file for the tests and for the main module:

```
$ pylint --rcfile=tests/pylintrc tests
$ pylint --rcfile=tomcatmanager/pylintrc tomcatmanager
```

You are welcome to use the pylint comment directives to disable certain messages in the code, but pull requests containing these directives will be carefully scrutinized.

As allowed by PEP 8 this project uses a nominal line length of 100 characters.

3.7.9 Documentation

The documentation is written in reStructured Test, and turned into HTML using Sphinx:

```
$ cd docs
$ make html
```

The output will be in `docs/build/html`.

If you are doing a lot of documentation work, the sphinx-autobuild module has been integrated. Type:

```
$ cd docs
$ make livehtml
```

Then point your browser at http://localhost:8000 to see the documentation automatically rebuilt as you save your changes.

Note: The `sphinx-autobuild` module has some limitations. Much of the documentation produced in this project is contained in the source code, and is incorporated via the Sphinx `autodoc` module. In order for `autodoc` to work, it must import the source code, and it's not very good about noticing and reloading source code modules as they change. If you change the source code and want to make sure you are seeing the current changes in your browser, best to kill the webserver and start it back up again.

Use `doc8` to check documentation quality:

```
$ invoke doc8
```

3.7.10 Make a Release

To make a release and deploy it to PyPI, do the following:

1. Merge everything to be included in the release into the **develop** branch.
2. Run `tox` to make sure the tests pass in all the supported python versions.
3. Review and update `CHANGELOG.rst`.
4. Update the milestone corresponding to the release at https://github.com/tomcatmanager/tomcatmanager/milestones
5. Push the **develop** branch to github.
6. Create a pull request on github to merge the **develop** branch into **master**. Wait for the checks to pass.
7. Merge the **develop** branch into the **master** branch and close the pull request.

8. Tag the **master** branch with the new version number, and push the tag.
9. Build source distribution, wheel distribution, and upload them to pypi staging:

```
$ invoke pypi-test
```

10. Build source distribution, wheel distribution, and upload them to pypi:

```
$ invoke pypi
```

11. Docs are automatically deployed to http://tomcatmanager.readthedocs.io/en/stable/. Make sure they look good.
12. Switch back to the **develop** branch. Add an **Unreleased** section to the top of `CHANGELOG.rst`. Push the change to github.

3.8 Changelog

All notable changes to tomcatmanager will be documented in this file.

The format is based on Keep a Changelog and this project uses Semantic Versioning.

3.8.1 1.0.0 (2020-02-01)

Changed

- Switch documentation theme from 'alabaster' to 'sphinx_rtd_theme'

Added

- Already have a setting to control network timeouts. Added a command line option '–timeout' to do the same, making it easier for modify for command-line only use.
- Adjustments for upstream changes in cmd2. No longer pinned to cmd2=0.9.4, but require cmd2>=0.9.14.
- Add support for Python 3.8.
- Add documentation style checking using `doc8`.

Removed

- Drop support for Python 3.4, which reached end-of-life on Mar 18, 2019.

3.8.2 0.14.0 (2019-05-16)

Changed

- `invoke clean.pycache` is now `invoke clean.bytecode`
- Run tests using python 3.7 on Appveyor and Travis
- Source code has been moved inside of `src` directory
- Pin cmd2 to version 0.9.4; newer versions break us badly

3.8.3 0.13.0 (2018-07-06)

Added

- In the interactive `tomcat-manager` tool, the history of previously executed commands is now persistent across invocations of the program.
- Added common developer tasks to `tasks.py`. To run these tasks, use the `invoke` command provided by pyinvoke.
- Tomcat 9.0.x officially supported. No material changes were required to gain this support, just validation via the test suite.
- Type hinting added for enhanced developer productivity in most IDE's
- Full support for Python 3.7

Changed

- `ServerInfo.__init__()` no longer accepts the result as a positional argument: it must be a keyword argument.

Fixed

- Test suite now runs several orders of magnitude faster. The upstream cmd2 used pyparsing which was very slow. `cmd2` versions >= 0.9.0 use `shlex` to parse commands.

3.8.4 0.12.0 (2018-02-23)

Added

- You can now deploy applications via a context xml file. A new interactive command `deploy context` and a new method `deploy_servercontext()` provide this capability.

Changed

- Better help messages in the interactive `tomcat-manager` tool
- `deploy()` has been replaced by three new methods: `deploy_localwar()`, `deploy_serverwar()`, and `deploy_servercontext()`.
- Commands which use an optional version parameter now use a `-v` option to specify the version
- Most commands now have `-h`/`--help` options

3.8.5 0.11.0 (2017-09-06)

Added

- New command line switches for `tomcat-manager`: `--quiet`, `--echo`, `--status_to_stdout`
- New setting `status_prefix` contains the string to emit prior to all status messages
- New class `TomcatApplication`

Changed

- If we get an http redirect during `TomcatManager.connect()`, save the new url so we don't have to re-traverse the redirect on every command.
- Interactive *`list`* command now can filter by application state, and has two sort options.
- `TomcatManager._user` is now `TomcatManager.user`
- `TomcatManager._url` is now `TomcatManager.url`
- `TomcatManager.list()` now returns a list of `TomcatApplication` objects
- Renamed `tm.codes` to `tm.status_codes` to clarify the purpose

3.8.6 0.10.0 (2017-08-24)

Added

- CHANGELOG.rst
- documentation for interactive mode
- documentation for use from the shell command line
- read settings from a config file
- add `config` command which allows user to edit config file
- server shortcuts: save url, user, and password in config file
- `which` command to show which tomcat server you are connected to
- `timeout` setting for HTTP timeouts
- `restart` command as synonym for `reload`
- Add tox for testing against multiple versions of python

Changed

- `status` command now pretty prints the xml response
- `TomcatManager.__init__` no long accepts paramemeters: use `connect` instead
- **`TomcatManager` methods which act on apps (`deploy`, `sessions`,** `stop`, etc.) now throw exceptions if no path is specified. Previously they returned a response with `r.ok == False`

3.8.7 0.9.2 (2017-08-16)

Added

- new TomcatManager.connect() method
- lots more documentation
- pytest now runs doctests

Changed

- version numbers now provided by `setuptools_scm`

3.8.8 0.9.1 (2017-08-10)

Changed

- New release to practice packaging and distribution

3.8.9 0.9.0 (2017-08-10)

Added

- Converted from a single script to an installable python package
- Remove documentation for tomcat 6, which is no longer supported
- Add `expire` command
- Add `vminro` command
- Add `sslconnectorciphers` command
- Add `threaddump` command
- Add `findleaks` command
- Add `status` command
- Unit tests using pytest
- Support Tomcat parallel deployment
- Real documentation using Sphinx
- Packaged to PyPI

Changed

- Switch from getopt to argparse
- Use `cmd2`, if available, instead of `cmd`
- Switch from `urllib` to `requests`

Removed

- Drop support for Python 3.3

3.8.10 Changes in 2014 and 2015

- Remove methods deprecated in Python 3.4
- Add documentation to support Tomcat 7

3.8.11 0.4 (2013-07-07)

Added

- Port to python 3
- New `resources` command

Removed

- Drop support for python 2

3.8.12 0.3 (2013-01-02)

Added

- Add code from private repo

Python Module Index

t

Index

E

F

H

I

J

L

N

O

P

R

S

T

U

V

W