Cheetah Template Portable

**Overview**

Cheetah Template Portable is an open source Python-based template engine and code generation framework. It's packaged with a PortableApps.com launcher so you can use it from a cloud folder, portable drive, or local folder without needing to install it on each PC. Learn more about [Cheetah Template](https://github.com/CheetahTemplate3/cheetah3) …

Components in this release (3.2.6.post1 Dev Test 1):

Cheetah Template 3.2.6.post1 (64 bit) (https://cheetahtemplate.org/)

Python 3.8.10 (64 bit) (https://github.com/winpython)

PortableAppsFormat 3.5 (https://portableapps.com/)

Bootstrap 4.6.1 (https://github.com/twbs/bootstrap)

jQuery JavaScript Library v2.2.4 (https://jquery.com/)

tether 2.0.0 (https://github.com/shipshapecode/tether)

theme\_styles.html (modified) from https://github.com/bofeng/

All of the above components have been integrated and packaged with a PortableApps.com launcher with the appropriate PortableAppsFormat as listed above.

Bootstrap is included with theme\_styles.html along with jQuery(jquery.js) and tether(tether.min.js).

Cheetah Template works with Python distribution as mentioned above. As with Cheetah Template Python is also in PortableAppsFormat.

**Cheetah Template: Python PortableAppsFormat**

Although the Python distribution is derived from WinPython, only Python itself is used. By using the PortableApps.com launcher, a Python shell launches with environment variables configured for Cheetah Template but Python is actually at its core. Here is a snippet of the shell:

\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_

\ \/ \/ /

\/ \* \* \/ CHEETAH 3.2.6.post1 Command-Line Tool

\ | /

\ ==----== / by Tavis Rudd <tavis@damnsimple.com>

\\_\_\_\_\_\_\_\_\_\_/ and Mike Orr <sluggoster@gmail.com>

USAGE:

------

cheetah compile [options] [FILES ...] : Compile template definitions

cheetah fill [options] [FILES ...] : Fill template definitions

cheetah help : Print this help message

cheetah options : Print options help message

cheetah test [options] : Run Cheetah's regression tests

: (same as for unittest)

cheetah version : Print Cheetah version number

You may abbreviate the command to the first letter; e.g., 'h' == 'help'.

If FILES is a single "-", read standard input and write standard output.

Run "cheetah options" for the list of valid options.

python –h:

usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...

Options and arguments (and corresponding environment variables):

-b : issue warnings about str(bytes\_instance), str(bytearray\_instance)

and comparing bytes/bytearray with str. (-bb: issue errors)

-B : don't write .pyc files on import; also PYTHONDONTWRITEBYTECODE=x

-c cmd : program passed in as string (terminates option list)

-d : debug output from parser; also PYTHONDEBUG=x

-E : ignore PYTHON\* environment variables (such as PYTHONPATH)

-h : print this help message and exit (also --help)

-i : inspect interactively after running script; forces a prompt even

if stdin does not appear to be a terminal; also PYTHONINSPECT=x

-I : isolate Python from the user's environment (implies -E and -s)

-m mod : run library module as a script (terminates option list)

-O : remove assert and \_\_debug\_\_-dependent statements; add .opt-1 before

.pyc extension; also PYTHONOPTIMIZE=x

-OO : do -O changes and also discard docstrings; add .opt-2 before

.pyc extension

-q : don't print version and copyright messages on interactive startup

-s : don't add user site directory to sys.path; also PYTHONNOUSERSITE

-S : don't imply 'import site' on initialization

-u : force the stdout and stderr streams to be unbuffered;

this option has no effect on stdin; also PYTHONUNBUFFERED=x

-v : verbose (trace import statements); also PYTHONVERBOSE=x

can be supplied multiple times to increase verbosity

-V : print the Python version number and exit (also --version)

when given twice, print more information about the build

-W arg : warning control; arg is action:message:category:module:lineno

also PYTHONWARNINGS=arg

-x : skip first line of source, allowing use of non-Unix forms of #!cmd

-X opt : set implementation-specific option. The following options are available

:

-X faulthandler: enable faulthandler

-X showrefcount: output the total reference count and number of used

memory blocks when the program finishes or after each statement in

the

interactive interpreter. This only works on debug builds

-X tracemalloc: start tracing Python memory allocations using the

tracemalloc module. By default, only the most recent frame is store

d in a

traceback of a trace. Use -X tracemalloc=NFRAME to start tracing wi

th a

traceback limit of NFRAME frames

-X showalloccount: output the total count of allocated objects for each

type when the program finishes. This only works when Python was bui

lt with

COUNT\_ALLOCS defined

-X importtime: show how long each import takes. It shows module name,

cumulative time (including nested imports) and self time (excluding

nested imports). Note that its output may be broken in multi-thread

ed

application. Typical usage is python3 -X importtime -c 'import asyn

cio'

-X dev: enable CPython's "development mode", introducing additional run

time

checks which are too expensive to be enabled by default. Effect of

the

developer mode:

\* Add default warning filter, as -W default

\* Install debug hooks on memory allocators: see the PyMem\_SetupD

ebugHooks() C function

\* Enable the faulthandler module to dump the Python traceback on

a crash

\* Enable asyncio debug mode

\* Set the dev\_mode attribute of sys.flags to True

\* io.IOBase destructor logs close() exceptions

-X utf8: enable UTF-8 mode for operating system interfaces, overriding

the default

locale-aware mode. -X utf8=0 explicitly disables UTF-8 mode (even w

hen it would

otherwise activate automatically)

-X pycache\_prefix=PATH: enable writing .pyc files to a parallel tree ro

oted at the

given directory instead of to the code tree

--check-hash-based-pycs always|default|never:

control how Python invalidates hash-based .pyc files

file : program read from script file

- : program read from stdin (default; interactive mode if a tty)

arg ...: arguments passed to program in sys.argv[1:]

Other environment variables:

PYTHONSTARTUP: file executed on interactive startup (no default)

PYTHONPATH : ';'-separated list of directories prefixed to the

default module search path. The result is sys.path.

PYTHONHOME : alternate <prefix> directory (or <prefix>;<exec\_prefix>).

The default module search path uses <prefix>\python{major}{minor}

.

PYTHONCASEOK : ignore case in 'import' statements (Windows).

PYTHONUTF8: if set to 1, enable the UTF-8 mode.

PYTHONIOENCODING: Encoding[:errors] used for stdin/stdout/stderr.

PYTHONFAULTHANDLER: dump the Python traceback on fatal errors.

PYTHONHASHSEED: if this variable is set to 'random', a random value is used

to seed the hashes of str and bytes objects. It can also be set to an

integer in the range [0,4294967295] to get hash values with a

predictable seed.

PYTHONMALLOC: set the Python memory allocators and/or install debug hooks

on Python memory allocators. Use PYTHONMALLOC=debug to install debug

hooks.

PYTHONCOERCECLOCALE: if this variable is set to 0, it disables the locale

coercion behavior. Use PYTHONCOERCECLOCALE=warn to request display of

locale coercion and locale compatibility warnings on stderr.

PYTHONBREAKPOINT: if this variable is set to 0, it disables the default

debugger. It can be set to the callable of your debugger of choice.

PYTHONDEVMODE: enable the development mode.

PYTHONPYCACHEPREFIX: root directory for bytecode cache (pyc) files.

**Cheetah Template: Cheetah Fill & Cheetah Compile**

**theme\_styles.html**

The theme\_styles.html exposes the most commonly used Bootstrap components which makes it useful to further customizations.

To create a Cheetah template file theme\_styles.tmpl cheetah fill is used. The cheetah command is actually a subcommand of Python (i.e. python cheetah).

**cheetah fill**

So, the syntax of cheetah fill in the current working directory (without changing directory):

cheetah fill --idir=C:\alpha\_beta\beta\CheetahTemplatePortable\Other\bootstrap --odir=C:\alpha\_beta\beta\CheetahTemplatePortable\Other\bootstrap --iext=.html --oext=.tmpl theme\_styles.html

As an example above the directories specified are provided as samples. Your directory structure might be different as you can always move portable app around as you see fit.

The above automatically creates theme\_styles.tmpl in the output directory specified. Note that the encoding is utf-8.

**theme\_styles.tmpl**

With the template definition at hand, it can be altered to store constant values and use inheritance to arrange the template in a hierarchy. Depending on your purpose, you can go with the containment approach.

**cheetah compile**

To compile theme\_styles.tmpl into theme\_styles.py, cheetah compile is used. The syntax for cheetah compile in the current working directory (without changing directory):

cheetah compile --idir=C:\alpha\_beta\beta\CheetahTemplatePortable\Other\bootstrap --odir=C:\alpha\_beta\beta\CheetahTemplatePortable\Other\bootstrap theme\_styles.tmpl

As it is the case again, the above directory structure maybe very different from yours. Again, note the encoding is utf-8.

**theme\_styles.py**

With success in compiling to a Python module and a class, it is very useful that the Python module can be modified to your hearts desired. It is very flexible so you can even integrated at this point to other frameworks for many purposes.

Here is a snippet of the class:

##################################################

## CLASSES

class theme\_styles(Template):

##################################################

## CHEETAH GENERATED METHODS

def \_\_init\_\_(self, \*args, \*\*KWs):

super(theme\_styles, self).\_\_init\_\_(\*args, \*\*KWs)

if not self.\_CHEETAH\_\_instanceInitialized:

cheetahKWArgs = {}

allowedKWs = 'searchList namespaces filter filtersLib errorCatcher'.split()

for k,v in KWs.items():

if k in allowedKWs: cheetahKWArgs[k] = v

self.\_initCheetahInstance(\*\*cheetahKWArgs)

**Cheetah Template: Cheetah Test**

Executing cheetah test runs 2177 tests. Most of the tests are very useful to determine the state of the Cheetah Template feature set. With some caveats, the Cheetah Template in PortableAppsFormat passed all tests. The completion of tests is display as:

.................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................

----------------------------------------------------------------------

Ran 2177 tests in 126.242s

OK

What are the caveats? The next sections will cover all those Cheetah Template and Python configurations, workarounds, and code changes for this release.

**Cheetah Template: Python & Cheetah Configurations, Workarounds, Code Changes, Et Cetera**

**Python & Cheetah Configurations**

To start, Cheetah.pth file was placed in site-packages which takes care of cheetah cheetah-compile Scripts path for python.

The cheetah and python shell is configured properly when the PortableApps.com launcher runs. When everything is up you land on the working directory. Of course as long as you are in the shell environment variables are present. You can even launch python interactive mode and all its environment variables are also present.

**Workarounds**

It is when you change directory in the same shell when things do not work as reliable as it should be. Even though all the environment variables are still valid the cheetah command no longer work and error messages like “Fatal error in launcher: Unable to create process using '"python" or some similar error messages. Recall in previous sections that was mentioned that cheetah is actually a subcommand (i.e. python cheetah). Out of roughly ninety-nine times, one time the shell work as it should but only when manually repeating the shell configurations again even though it was fine. Of course changing back to the working directory everything works as before. A lot of research was put into this issue so some stuff came out from it.

There were many cases of similar and different platforms having similar error issues and so on. Python has so many features at its core including the ability to run multiple versions of Python on the same platform/OS using many methods (i.e. directly executing each Python version, using some virtual environments venv/pyvenv and more). So, one of the workaround is to stay at the current working directory as launched and then specifying the input and out directory as cheetah supports that. As mentioned previously and here again,

cheetah fill --idir=C:\alpha\_beta\beta\CheetahTemplatePortable\Other\bootstrap --odir=C:\alpha\_beta\beta\CheetahTemplatePortable\Other\bootstrap --iext=.html --oext=.tmpl theme\_styles.html

cheetah compile --idir=C:\alpha\_beta\beta\CheetahTemplatePortable\Other\bootstrap --odir=C:\alpha\_beta\beta\CheetahTemplatePortable\Other\bootstrap theme\_styles.tmpl

As you see, --idir and --odir, would allow sidestepping all that changing working directory stuff. Another workaround is to feed all the Cheetah Template sources to another framework and work from there but that is for another topic entirely. It is not necessary as the cheetah subcommand in a shell works fine if stay within its current limits for this release.

**Code Changes**

Before getting into code changes, have a read under the headline “Testing your installation” at https://cheetahtemplate.org/users\_guide/gettingStarted.html. Remember, “Certain test failures are insignificant.” After examining the source and in the interest of getting cheetah test to pass all tests, code changes were made.

In site-packages\Cheetah\Tests\SyntaxAndOutput.py,

'includeFileName': '..\\..\\Data\\CheetahTemplatePortable64bitTemp\\parseTest.txt',

fp = open('..\\..\\Data\\CheetahTemplatePortable64bitTemp\\backslashes.txt', 'w')

Or do searches for “parseTest.txt” and “backslashes.txt” and you will find them all.

Otherwise, there are many error messages and error codes and not included here.

In site-packages\Cheetah\Tests\CheetahWrapper.py:

##################################################

# TEST CASE CLASSES

class OneFile(CFBase):

"""

substitute 'cheetah' with complete direct path to python.exe and complete direct path to cheetah.exe

at the present time as of january 1, 2022

'cheetah' subcommand = python.exe cheetah.exe

until function calls replace subcommands the crude method of substituting 'cheetah' as mentioned above

works

"""

The above comments say it all. In this release, function calls have not replace subcommands yet. Again, lots of error messages and error codes if the comments above were not followed and again those error messages and error codes are not included here.

**Cheetah Template: Compile Codecs Encoding**

This is a case of encoding issue with available codecs use by cheetah compile. For some reason every time this is executed,

cheetah compile --idir=C:\alpha\_beta\beta\CheetahTemplatePortable\Other\bootstrap --odir=C:\alpha\_beta\beta\CheetahTemplatePortable\Other\bootstrap --encoding=utf-8 theme\_styles.tmpl

This came up,

UnicodeEncodeError: 'ascii' codec can't encode characters in position 37265-3726

6: ordinal not in range(128)

Eventually it was determined that the encoding compiler setting was not picked up from commandline so a forced utf-8 default setting in the source still did not work. Finally, a code change in site-packages\Cheetah\Compiler.py (self.\_moduleEncoding) was necessary.

def \_setupCompilerState(self):

self.\_activeClassesList = []

self.\_finishedClassesList = [] # listed by ordered

self.\_finishedClassIndex = {} # listed by name

self.\_moduleDef = None

self.\_moduleShBang = '#!/usr/bin/env python'

self.\_moduleEncoding = 'utf-8'

According to many sources including official Python documentations, utf-8 was default for Python beginning with version 3. Again, if another framework were to just take the Cheetah Template source and work with it then the encoding setting is already set elsewhere. Again, this is another topic entirely.

**Cheetah Template: PortableAppsFormat Portability Notes**

Cheetah Template (and Python) uses %APPDATA% to store its settings and data. Specifically,

\Users\<username>\AppData\Local\Temp\CheetahTemplatePortable64bitTemp

and

\Users\<username>\AppData\Local\pip

are use.

To make the app portable, the PortableApps.com launcher App\AppInfo\Launcher\CheetahTemplatePortable64bit.ini has the

[DirectoriesMove]

CheetahTemplatePortable64bitTemp=%APPDATA%\CheetahTemplatePortable64bitTemp

pip=%APPDATA%\pip

which puts the settings and data in the Data\CheetahTemplatePortable64bitTemp and Data\pip inside the portable app directory.

A word about cheetah test: As long as you do not modify site-packages\Cheetah\Tests\CheetahWrapper.py,

DELETE = True # True to clean up after ourselves, False for debugging.

OUTPUT = False # Normally False, True for debugging.

and turn on debugging, then the temporary directories such as tmp\_hr9hz5b, tmp\_hr9hz5b\SRC, and so on will be cleaned up after completion of tests.

Besides meeting the portable app definition at https://portableapps.com/about/what\_is\_a\_portable\_app, it is very workable in other OS/platform such as Linux, UNIX, BSD, etc. via Wine (winehq.org) & Mac OS X via CrossOver, Wineskin, WineBottler, PlayOnMac.

Finally, “Why PortableApps.com Format and a PortableApps.com Installer?” Perhaps, a read at <https://portableapps.com/about/what_is_a_portable_app#whypaf> would suffice.