Cheetah Users Guide

Release 0.9.6

The Cheetah Development Team

June 12, 2001

cheetahtemplate-devel@lists.sourceforge.net

Contents

©Copyright 2001, The Cheetah Development Team. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

1 Introduction

1.1 What is Cheetah?

Cheetah is a Python-based template engine and code-generator. It aims:

- **to make it easy to separate content, graphic design, and program code.**

  Program code should not pollute HTML and HTML should not pollute program code. Nor should content pollute the structure of complex HTML designs and vice versa.

  There should be no need for a designer to work through a programmer to change a website's design or to use dynamic components that have already been coded. Likewise, content-providers should not have to work through a 'webmaster' to add new content to a site. All members of the team should be able to work independently and in parallel.

  A clean separation makes it easier for a team of content-providers, designers, and programmers to work together without stepping on each other's toes and polluting each other's work. Other advantages include faster development time; HTML and program code that are easier to understand and maintain; content that can be displayed in a variety of non-HTML formats such as PDF; and highly modular, flexible, and reusable site architectures.

- **to make it easy to integrate content, graphic design, and program code.**

  While it should be easy to develop content, graphic design, and program code separately, it should NOT be difficult to integrate them as part of a website. There should be no difficult hoops to jump through.

  It should be easy:

  - for programmers to create reusable components and functions that are accessible and understandable to designers.
  - for designers to mark out placeholders for content and dynamic components in their templates.
  - for designers to soft-code aspects of their design that are either repeated in several places or are subject to change.
  - for designers to extend and customize existing templates and thus minimize duplication of effort and code.
  - and, of course, for content-providers to use the templates that designers have created.

- **to provide template designers with a small set of 'Display Logic' programming structures such as conditional blocks and for loops**

  Graphic designers often do tasks that would be easier, faster, and less error prone if they had access to **conditional blocks** and **for loops**. However, a full programming language would be overkill for these simple tasks and most designers don't have the time or desire to learn one.

- **to be equally well-suited for HTML, SGML, XML, SQL, Postscript, form email, LaTeX, or any other text-based format.**

  Although it was designed with dynamic websites and web applications in mind, Cheetah is not HTML-specific.

- **to achieve all these aims in a manner that is efficient, flexible, and extendable.**

Cheetah achieves these aims by:

- blending the power and flexibility of Python with the simplicity of a small Template Definition language that non-programmers can understand.
- giving template designers a simple way of accessing Python variables, objects, and functions in their templates.
- providing a modular, object-orientated framework that makes it easy to create and maintain large websites.

- compiling 'Template Definitions' into native Python code at startup. Thereafter this code is executed for each request. This approach is dramatically faster than the string substitution approach used by many templating engines.
- providing a very simple, yet powerful, caching mechanism that can significantly increase the responsiveness of a dynamic website.

1.2 Why is it called Cheetah?

Cheetah is fast, flexible, agile and graceful - like its namesake.

1.3 Who developed Cheetah?

Cheetah is one of several templating frameworks that grew out of a 'templates' thread on the 'Webware For Python' email list. Tavis Rudd, Mike Orr, Chuck Esterbrook, Ian Bicking and Tom Schwaller are the core developers.

1.4 How mature is Cheetah?

Cheetah is alpha/beta software as this User's Guide is incomplete and several aspects of the design are still subject to change. However, it has been tested extensively and has few known issues. We are hoping to release production version 1.0 in the summer of 2001.

Here's a summary of known issues and aspects of the design that are in flux.

- The #include directive is not working with relative path file includes when used with Webware. This should be resolved soon.
- The #include directive might be reworked to monitor for changes in the included file at run-time. It currently does the include once-off at compile-time.
- The implementation of $placeholders(WithArgstrings) needs to be fleshed out to handle nesting.

1.5 Where can I get releases?

Cheetah releases can be downloaded from http://CheetahTemplate.sourceforge.net

1.6 Where can I get news?

News and updates can be obtained from the the Cheetah website: http://CheetahTemplate.sourceforge.net

Cheetah discussions take place on the list cheetahtemplate-discuss@lists.sourceforge.net.

If you encounter difficulties, or are unsure about how to do something, please post a detailed message to the list.

1.7 How can I contribute?

Cheetah is the work of many volunteers. If you use Cheetah please share your experiences, tricks, customizations, and frustrations.

Bug reports and patches

If you think there is a bug in Cheetah, send a message to the email list with the following information:

1. a description of what you were trying to do and what happened
2. all tracebacks and error output
3. your version of Cheetah
4. your version of Python
5. your operating system
6. whether you have changed anything in the Cheetah installation

Example sites and tutorials

If you're developing a website with Cheetah, please send a link to the email list so we can keep track of Cheetah sites. Also, if you discover new and interesting ways to use Cheetah please share your experience and write a quick tutorial about your technique.

Macro libraries

We hope to build up a framework of macros libraries (see section **??**) to distribute with Cheetah and would appreciate any contributions.

Test cases

Cheetah is packaged with a regression testing suite that is run with each new release to ensure that everything is working as expected and that recent changes haven't broken anything. The test cases are in the Cheetah.Tests module. If you find a reproduceable bug please consider writing a test case that will pass only when the bug is fixed. Send any new test cases to the email list with the subject-line "new test case for Cheetah."

Publicity

Help spread the word ... recommend it to others, write articles about it, etc.

1.8 Acknowledgements

We'd like to thank the following people for contributing valuable advice, code and encouragement: Geoff Talvola, Jay Love, Terrel Shumway, Sasa Zivkov, Arkaitz Bitorika, Jeremiah Bellomy, Baruch Even, Paul Boddie, Stephan Diehl, and Geir Magnusson.

The Velocity, WebMacro and Smarty projects provided inspiration and design ideas. Cheetah has benefited from the creativity and energy of their developers. Thank you.

1.9 License

Cheetah is released for unlimited distribution under the terms of the Python license.

2 Getting Started

2.1 Requirements

Cheetah requires Python release 2.0 or greater and should run on any operating system that Python 2.0 runs on.

2.2 Installation

To install Cheetah for a single user:

1. copy the 'src' sub-directory to a directory called 'Cheetah' that is in the user's PYTHON_PATH

To install Cheetah for system-wide use:

1. on POSIX systems (AIX, Solaris, Linux, IRIX, etc.) become the 'root' user and run: python ./setup.py install
2. On non-POSIX systems, such as Windows NT, login as an administrator and type this at the command-line: python setup.py install

On POSIX systems, the system-wide installation will also install the Cheetah's command-line compiler program, TScompile, to a system-wide executable path such as /usr/local/bin.

2.3 Testing your installation

You can run the test suite to insure that your installation is correct by following these steps:

1. CD into the directory ./Cheetah
2. type: `python Tests.py`

If any of the tests fail please send a message to the email list with a copy of the test output and the following details about your installation:

1. your version of Cheetah
2. your version of Python
3. your operating system
4. whether you have changed anything in the Cheetah installation

2.4 Quickstart tutorial

This tutorial briefly introduces the basic usage of Cheetah. See the following chapters for more detailed explanations.

This tutorial will be fleshed out further at later date.

The core of Cheetah is the `Template` class in the `Cheetah.Template` module. The following example shows how to use the `Template` class from an interactive Python session. Lines prefixed with `>>>` and `...` are user input. The remaining lines are Python output.

```
>>> from Cheetah.Template import Template
>>> templateDef = """
... <HTML>
... <HEAD><TITLE>$title</TITLE></HEAD>
... <BODY>
... $contents
... </BODY>
... </HTML>"""
>>> nameSpace = {'title': 'Hello World Example', 'contents': 'Hello World!'}
>>> templateObj = Template(templateDef, nameSpace)
>>> print templateObj

<HTML>
<HEAD><TITLE>Hello World Example</TITLE></HEAD>
<BODY>
Hello World!
</BODY>
</HTML>
>>> print templateObj  # templateObj can be printed as many times as you need

<HTML>
<HEAD><TITLE>Hello World Example</TITLE></HEAD>
<BODY>
Hello World!
</BODY>
</HTML>
```

3 Template Objects

3.1 The Template class

The Template class is the heart of Cheetah. It parses and compiles Template Definitions into python code and serves the filled template output to any client that requests it.

3.2 Constructing Template objects

3.3 Using Template objects

3.4 Modifying Template objects

4 The Template Definition Language

Template definitions are text strings, or files, that have been marked up with Cheetah tags for special processing. Cheetah has 2 types of tags:

1. **placeholders**: for marking areas of the template that should be replaced with something.

   Placeholders begin with a dollar sign (`$varName`).

2. **directives**: for everything else:

   (a) **raw text** for marking verbatim blocks should not be parsed for Cheetah tags.

   (b) **comments** that should not appear in the output

   (c) **includes** to include external text. The text can be included verbatim or with parsing for Cheetah tags.

   (d) **display logic** such as **conditional blocks** (if-blocks) and **for loops**

   (e) **blocks**, which are named sections of a template that can be redefined (overridden) in a subclass or by template users

   (f) etc.

   Directives begin with a hash character (#).

4.1 Placeholder tags

Cheetah uses placeholder tags in the form `$varName` to mark out areas of the template definition that should be replaced with something. Placeholders are equivalent to **fields** on a form. Placeholders can be replaced with plain content or variables, objects and function output from Python.

The example below demonstrates the use of placeholders in an HTML document.

```
<HTML>
<HEAD><TITLE>$title</TITLE></HEAD>
<BODY>
$contents
</BODY>
</HTML>
```

When this template is filled in, the placeholders `$title` and `$content` will be replaced with the values of the variables `title` and `content`.

Rules for placeholder names

- Cheetah ignores all dollar signs (`$`) that are not followed by a letter or an underscore. As a corollary, dollar amounts (`$2.50`)are not placeholders but are instead output literally as they should be. Cheetah also ignores any placeholder escaped by a backslash (`\$placeholderName`).
- The first character of a placeholder name must be either an underscore or a letter. Valid characters for the rest of the name are underscores, letters, numbers and periods. These names are valid: `$a`, `$_`, `$var`, `$_var`, `$var1`, `$_1var`, `$var2_`, `$dict.key`, `$list.item`, `$object.method`. These names are not: `$1`, `$var@2`, `$^var`.
- names are case-sensitive. `$var` does not equal `$Var` or `$vAr` or `$VAR`.

- Trailing periods are ignored. Cheetah will recognize that placeholder name in `$varName.` is `varName` and the period will be left alone in the filled template output.
- Placeholders can also be written in the form `${placeholderName}`. This is useful for cases where there is no whitespace between the placeholder and surrounding text (`surrounding${embeddedVar}text`).

The searchList

A **namespace** is a Python dictionary that links names to values. Each template definition that has been loaded into a `Template` objet is associated with an ordered list of namespaces in which values for placeholder variable names can be stored. This list is called the **searchList**.

The searchList can contain one or more namespaces. In most cases only one namespace will be in the searchList unless you explicitly load extra ones. When Cheetah fills in `$content` in previous example it searchs sequentially through the searchList until it finds a value for `$content`. Thus, if three namespaces are loaded and two of them contain a value for `$content`, the value for `content` from the namespace that is closest to the start of the searchList will be returned.

If you add a Python object to the searchList, its attributes and methods will be accessible as placeholder names. For example, `myObject` contains `myAttrib` and `myMethod`. If `myObject` is added to the searchList, `$myAttrib` and `$myMethod` can be used as placeholder names.

The default namespace in every searchList is the `Template` object itself. This means that any attributes or methods that are added to classes that inherit from `Template` can be accessed in templates via `$placeholders`. New namespaces can be added to the searchList at any time using the `Template.addToSearchList()` method. See section **??** for more information on how to use namespaces and the searchList.

Placeholder values

Placeholder names can map to Python text strings, numbers, dictionaries, lists (arrays), functions, objects, or even nested Cheetah templates. If the value is not a String, Cheetah will call `str()` on it to obtain a string representation that can be inserted into the template. But if the value is a callable object (e.g., a function or a method), Cheetah will first call it and then call `str()` on the result. You may include or omit the () after callable placeholders: Cheetah will call it (without arguments) in either case.

```
<HTML>
<HEAD><TITLE>$title</TITLE></HEAD>
<BODY>

$aString                        ## these names are chosen to indicate what
$aNumber                        ## type of value they map to.  They could be
$aList                          ## any valid placeholder name.
$aDictionary
$aFunction
$anObject

</BODY>
</HTML>
```

Note that you don't need to include () after `$aFunction`. Cheetah will recognize callable variables like functions and methods, call them and interpolate their return value in the filled template output.

You cannot pass arguments to a function or method called this way. (Actually, you can, but this feature may be removed soon, because it introduces too much complexity that is better done in Python outside the template.) The

function or method must either accept no arguments, or all its arguments must have default values. Macro calls (see the Macros section) can accept arguments, but we aren't talking about macros here.

Templates can be nested. It is valid to embed a placeholder name that maps to another `Template` object. The filled output of the nested template will be interpolated into the top-level template's output. There is no limit on the depth of template nesting.

Placeholders with no value defined

If there is no value defined in the searchList for a placeholder name, Cheetah will search for an leading underscore version of the name. For example, if it can't find `$varName` it will attempt to find `$_varName`. If that fails, Cheetah will include the placeholder tag verbatim in the filled template output.

This behaviour can be customized and a default value can be set for names that are not found. See section **??** for more details.

Dotted notation

Placeholder names can also use **dotted notation** to access entries in dictionaries, items in lists, and the attributes and methods of objects. Cheetah uses a consistent dotted notation syntax to access the contents of all types of containers. This is unlike Python, C++, Java, and other languages where dotted notation can only be used to access the attributes and methods of objects. [1]

```
$aDictionary.keyName            ## must be a valid key of the dictionary
$aList.3                        ## must be a valid index (0-based like in Python)
$anObject.attributeName         ## must be a valid attribute name
$anObject.method                ## must be a valid method name,
                                ##   leave off the parentheses on method()
```

Dotted notation can be used on nested containers of any depth.

```
$dict1.dict2.dict3.keyName      ## nested dictionaries
$anObject.nestedDict.keyName    ## if the object contains a dictionary
```

Caching

By default the value of each `$placeholder` is updated for each request. If you want to statically cache the value of the `$placeholder` upon startup, add an asterisk after the $ sign. `$var` becomes `$*var`. See the section on the `#cache` directive below for more information.

If you only need to update the value of the `$placeholders` at specific time intervals use this form: `$variable` becomes `$*15*variable` , where 15 is the time interval in minutes. The time interval can also be specified in fractions of a minute such as `$*0.5*variable`.

[1] Cheetah uses a Python module called NameMapper to handle this style of dotted notation. NameMapper is distributed as part of the Cheetah package and can be used as a stand-alone tool. NameMapper was inspired by Chuck Esterbrook's NamedValueAccess module.

```
<HTML>
<HEAD><TITLE>$title</TITLE></HEAD>
<BODY>

$var                ## dynamic - will be reinterpolated for each request
$*var2              ## static - will be interpolated only once at start-up
$*5*var3            ## timed refresh - will be updated every 5 minutes.

</BODY>
</HTML>
```

4.2 Directive tags

Directives tags are used for all functionality that cannot be handled with simple placeholders and are enclosed in # and /#. Cheetah does not use HTML/XML style tags because they would be hard to distinguish from real HTML tags and would not be visible in rendered HTML when something goes wrong.

Some directives consist of a single tag while others have **start** and **end** tags that surround a chunk of text. End tags are written in the form `#end nameOfTheDirective/#`.

Escaping directives

Directives can be escaped by placing a backslash (\) before them. Escaped directives will be printed verbatim.

Tag closures: explicit and implicit

Directive tags can closed explicitly with /# or implicitly with the end of the line if you're feeling lazy.

```
#block /#
Text in the contents area of the
block directive
#end block /#
```

or

```
#block
Text in the contents area of the
block directive
#end block
```

Whitespace handling

4.3 Comment directives

Comment directives are used to mark notes, explanations, and decorative text that should not appear in the output. There are two forms of the comment directive: single-line and multi-line.

All text in a template definition that lies between 2 hash characters (##) and the end of the line is treated as a single-line comment and will not show up in the output, unless the 2 hash characters are escaped with a backslash.

```
<HTML>
<HEAD><TITLE>$title</TITLE></HEAD>
<BODY>
##====================================  a decorative comment
$contents                              ## an end-of-line comment
##====================================
</BODY>
</HTML>
```

Any text between `#*` and `*#` will be treated as a multi-line comment.

```
<HTML>
<HEAD><TITLE>$title</TITLE></HEAD>
<BODY>
#*
   Here is some multiline
   comment text
*#
##====================================  a decorative comment
$contents                              ## an end-of-line comment
##====================================
</BODY>
</HTML>
```

4.4 #raw directives

Any section of a template definition that is delimeted by `#raw` and `#end raw` will be printed verbatim without any parsing of $placeholders or other directives. This can be very useful for debugging or writing Cheetah examples and tutorials.

4.5 #include directives

`#include` directives are used to include text from outside the template definition. The text can come from `$placeholder` variables or from external files. The example below demonstrates use with `$placeholder` variables.

```
#include $myParseText
```

This example demonstrates its use with external files.

```
#include "includeFileName.txt"
```

By default, included text will be parsed for Cheetah tags. The keyword **raw** can be used to mark the text for verbatim inclusion without any tag parsing.

```
#include raw $myParseText
#include raw "includeFileName.txt"
```

`Template` uses its .getFileContents(fileName) method to locate the file to be included. This method can be overriden in subclasses if you want to modify or extend its behaviour. It is possible to implement the logic for getting remote files such as `http://myserver.com/file.txt`.

4.6 #cache directives

4.7 Display logic directives

Conditional blocks

For loops

4.8 #block directives

#redefine directives

5 Macros

6 Using Cheetah with Webware

6.1 Background

Webware is a 'Python-Powered Internet Platform' that uses servlets in a manner similar to Java servlets. 'WebKit' is the name of Webware's application server. For more details please visit http://webware.sourceforge.net.

As Cheetah's core is flexible there are many ways to use it with Webware servlets. There are two broad categories: the Inheritance approach and the Containment approach. In the Inheritance approach a servlet is created that subclasses both the Template class and Webware's HTTPServlet class. The Template instance IS the servlet and its .respond() method is automatically called by WebKit for each request. All pre-request processing is handled via Cheetah.

In the Containment approach an instance of the Template class is wrapped up inside of a Webware servlet class. Instances of the servlet class must explicitly call the Template instance's .respond(), or .__str__(), method for each request. In this case the servlet class can handle whatever per-request processing needs to be done before it calls Cheetah.respond().

The Inheritance approach is the simplest and is best suited for building sites from scratch.

The Containment approach is slightly more complex and is best suited for use with existing Webware servlets. It is also ideal for cases where you wish to use Cheetah for only a portion of the servlet's output, such as a discussion-forum table at the bottom of a webpage.

6.2 Using the Inheritance approach

Cheetah.Servlet provides a servlet class that can be subclassed to create Webware servlets, as in the trivial example below.

```
## FILE: hello_world.py ##
template = """
<HTML>
<HEAD><TITLE>'Hello World - Test Servlet</TITLE></HEAD>
<BODY>
Hello World!
</BODY>
</HTML>
"""
from Cheetah.Servlet import TemplateServlet
class hello_world(TemplateServlet):
    def __init__(self):
        TemplateServlet.__init__(self, template)
```

TemplateServlet's constructor method (`TemplateServlet.__init__()`) adds the attribute dictionary of the servlet to the searchList that `$placeholder` variables can extracted from. Thus, attributes and methods of the servlet object can be interpolated into the template like this:

```
## FILE: hello_world.py ##
template = """
<HTML>
<HEAD><TITLE>$title</TITLE></HEAD>
<BODY>
$contents
</BODY>
</HTML>
"""
from Cheetah.Servlet import TemplateServlet
class hello_world(TemplateServlet):
    title = 'Hello World - Test Servlet'
    def __init__(self):
        TemplateServlet.__init__(self, template)

    def contents(self):
        return 'Hello World!'
```

This process can be simplified for non-programmers. All the Python wrapper code in these examples can be generated automatically by `cheetah-compile`, a small program that is installed with Cheetah. `cheetah-compile` parses Template Definitions written in files with the .tmpl extension and generates a Webware servlet file with the .py extension for each .tmpl file. Type `cheetah-compile` after installing Cheetah to get information on how to use it.

Here's the first example as a .tmpl file:

```
## FILE: hello_world.tmpl ##
<HTML>
<HEAD><TITLE>Hello World - Test Servlet</TITLE></HEAD>
<BODY>
Hello World!
</BODY>
</HTML>
```

Here's the second example as a .tmpl file. Note that all lines that begin with ## are comment lines.

```
## FILE: hello_world.tmpl ##
##====================================
#data
title = 'Hello World - Test Servlet'
def contents():
    return 'Hello World!'
#/data
##====================================
<HTML>
<HEAD><TITLE>$title</TITLE></HEAD>
<BODY>
$contents
</BODY>
</HTML>
```

The #data directive

The previous example introduced the `#data` directive, a shortcut means of adding data into the servlet's attribute dictionary and thus into the Template's **searchList**. The `#data` directive can be used in any Template Definition, regardless of whether the definition is in a .py or .tmpl file.

`#data` directives can contain any valid Python code

The SkeletonPage framework

PlateKit

Forget what you knew about PlateKit. It's undergoing some changes and is out of action for the time-being.

6.3 Using the Containment approach

6.4 User interaction in either approach

6.5 Components

What are components?

Using components

Building components

7 Customizing and extending Cheetah

7.1 Custom handling of unknown placeholder names

7.2 Plugins

The PSP plugin

7.3 Custom variable-tags

7.4 Custom directives

7.5 Custom error handlers

7.6 Safe delegation

Safe Delegation, as provided by Zope and Allaire's Spectra, is not a core aim of Cheetah. However, several hooks were built into Cheetah so that Safe Delegation can be implemented at a later date.

8 Examples

The Cheetah distribution comes with an 'examples' directory. Browse the files in this directory and its subdirectories for examples of how Cheetah can be used.

8.1 Syntax examples

Cheetah's `Tests` module contains a large number of test cases that can double as examples of how the Template Definition Language works. To view these cases go to the base directory of your Cheetah distribution and open the file Cheetah/Tests.py in a text editor.

8.2 Webware Examples

The 'examples' directory has a subdirectory called 'webware_examples'. It contains example servlets that use Webware.

A subdirectory titled 'webwareSite' contains a complete website example. This site is my proposal for the new Webware website. The site demonstrates the advanced Cheetah features such as the `#data` and `#redefine` directives. It also demonstrates how the TScompile program can be used to generate Webware .py servlet files from .tmpl Template Definition files.