The Stencil Templating System

The Stencil Template System is used to produce text to represent data, such as a web page or report. It is primarily intended to be very simple to learn.

A stencil comprises the output text with embedded simple processing instructions.

All data for a stencil must be prepared in advance as a stencil cannot contain embedded code.

## Example

Input:

appliance: “lamp”

isTurnedOn: true

Stencil:

The {appliance} is [if isTurnedOn]on[else]off[end].

Output:

The lamp is on.

# The input data

The data is presented to the stencil as a map of name-value pairs. The value may be any data type including structures such as collections and maps.

The value identifier should be limited to the normal characters legal in identifiers.

As a guide alpha-numeric characters are fine but braces, brackets, and commas must be avoided.

Specifically, an identifier can contain:

* Any Unicode alphabetic character
* Any Unicode decimal number character
* Any Unicode connector punctuation
* Any Unicode marks (i.e. accents and decorations)
* Any Unicode currency symbol
* The solidus character (“/”)
* The period character (“.”) which has a special interpretation.

## Meaning of period (“.”)

The data is assumed to contain structures equivalent to dictionaries and arrays. The period is used as a path separator to traverse the input data structures.

For example, given this input JSON:

{

"a": [

{

"b": 1,

"c": false

},

true,

{

"d": 2

}

]

}

Then:

* “a.0.b” == 1
* “a.1” == true
* “a.2” == { “d”:2 }

## Inheritance

Every level of the input structure inherits from its container. This means that the following identities also hold:

* “a.0.1” == true (“a.0” does not have a property “1”, so it inherits it)
* “a.2.0.d” == 2 (“a.2” does not have a property “0”, so it inherits it from “a”, and “a.0” does not have a property “d”, so it inherits if from “a.2”)

## Handling of Maps

If part of the input data is a map, then its contents will be accessed using the relevant section of the input identifier to select a value from the map.

If the map does not specify a value for “isEmpty”, then this value returns a Boolean indicating if the map is empty or not.

If the map does not specify a value for “size”, then this value returns the number of entries in the map.

## Handling of Lists

If the input is a list, array or an iterable, then it is values can be accessed using a zero-based index. A “size” and an “isEmpty” properties are also available.

## Everything Else

If the input is not a map, array, list, nor iterable, then it is treated as Java Bean, and its properties can be access according to the normal conventions for such.

# Literals, Values and Directives

A stencil contains literal text, which is simply copied to the output, values which are replaced with data from the input, and directives which control processing.

A value is always enclosed in braces, the “{“ and “}” characters.

A directive is always enclosed in brackets, the “[“ and “]” characters.

## Whitespace

Whitespace is ignored around the parameters inside a value.

Whitespace is ignored around the keywords and parameters of a directive.

Whitespace is ignored between two directives if it contains a new-line.

This last rule is to enable formatting of stencils to improve readability without compromising the output and is best explained by an example. Compare the following two equivalent stencils:

The nicely formatted version that ignores whitespace that contains a new-line between directives:

Headers:  
--------  
[loop headers]  
 [loop value]  
 [apply F.index]  
 [if isFirst]  
 [\*\*]{key,format,%-15s} : {value}  
[\*\*]  
 [else]  
 [\*\*] : {value}  
[\*\*]  
 [end]  
 [end]  
[end]

Note the use of directive comments “[\*\*]” to separate ignorable whitespace from required whitespace.

The not nicely formatted version that does not include ignorable white space:

Headers:  
--------  
[loop headers][loop value][apply F.index][if isFirst]{key,format,%-15s} : {value}  
[else] : {value}  
[end][end][end]

# Comments

Stencils may include comments. There are two kinds.

[\* This is a directive comment. \*]

{\* This is a value comment. \*}

The only difference between the two comment styles is how they affect surrounding whitespace. Remember that whitespace containing a new-line is ignored between two directives:

[\* This is a directive comment. \*]

[\* The preceding whitespace was ignored. \*]

{\* This is a value comment. \*}

[\* The preceding whitespace was not ignored. \*]

This can be useful as in the example above to make formatting easier, or to preserve necessary line breaks. For example, this “if” directive never outputs a new-line:

[if newLine]

[end]

But this one does:

[if newLine]

{\*\*}[end]

# Directives

## The “if” directive

[if value] … value is true … [end]

[if value] … value is true … [else] … value is false … [end]

The “IF” directive is used to process part of the stencil only if a value is true, and optionally process part of the stencil only if a value is false.

Value is not an expression, it is a value in the input data. The value is equivalent to true if it is **not** any of the following:

* Boolean False
* Null
* The empty string
* A string that equals “false”, ignoring case
* A floating point Not-a-number (NaN)
* Zero

For example, all the following are equivalent to true:

* Boolean.TRUE
* 86
* new HashMap<>()
* “calculator”

## The “loop” directive

This is used to process every entry in a collection or map. When processing a map, the values are the contents of the map’s entry set.

[loop map]

{index} : {key} is mapped to {value}

[end]

There is no map, or it is empty.

[else]

Within a loop there are only four values available (though others are inherited):

1. “value” – the current value of the list, or the value of the map entry
2. “key” – only present for maps where it is the current map key
3. “index” – the zero based index
4. “size” – the number of entries in this loop

The “else” block is invoked if the value is missing or empty.

## The “include” directive

Code re-use is good. The “include” directive allows another stencil to be included in the current stencil.

[include ../path/general]

The path is a literal. It may be absolute or relative.

Dynamic includes are not supported.

### Resource Bundles

Code can also be included from resource bundles. The syntax is:

[{ *key* }]  
 [{ *bundle*, *key* }]

The first form retrieves an entry identified by *key* from the current resource bundle. The current resource bundle may be set with the “set” directive.

The second form specifies the resource bundle to use as well as the key.

The text retrieved from the resource bundle is parsed as a stencil and processed in place.

Stencils drawn from resource bundles cannot use the include directive, but can include other resource bundle entries.

## The “use” directive

Specifying the full path to a value can be tiresome, or limit code re-use. The “use” directive switches the root to the specified datum.

[use mapValue] … [else] … [end]

[use otherValue] … [end]

An optional “[else]” block may be given. This optional block is processed if the datum is missing or null.

## The “set” directive

The “set” directive is used to specify one or more processing parameters.

[set bundle=api\_messages, escape=html\_strict]

### The “bundle” parameter

This is used to specify the default resource bundle from which messages may be drawn. Use of resource bundles will be explained later.

### The “escape” parameter

Sets the current character escaping system for values. Possible escape systems are explained later.

## The “apply” directive

The input data may contain functions which can be invoked during processing. The apply directive makes this possible.

[apply isLess = F.is( a, 'LT', b )]

[if isLess] ‘a’ is less than ‘b’ [end]

A function does not have to return a single value. It can also manipulate the current data directly.

The parameters to the function are either references to input data, or literals enclosed in single quotes. Numerical literals are still enclosed in single quotes.

Creating functions is an advanced topic, not covered here.

See “Common Functions” for descriptions of commonly available functions.

# Values

As stencil templates are for displaying data, there are obviously many ways to display and format data.

The table explains the basics:

|  |  |
| --- | --- |
| Syntax and example | Purpose |
| {value}  {value} | Outputs the value by invoking the Java toString method on it. Nulls are output as an empty string. |
| {value, date, style}  {value, date, iso-ordinal} | Accepts a date value and formats it according to the specified style. Date-time formats listed in Appendix 1.  The second form allows for formats that include |
| {value, time, *style*}  {value, time, short} | Accepts a time value and formats it according to the specified style. Date and time formats in Appendix 1. |
| {value, datetime, *style*}  {value, datetime, rfc822} | Accepts a date and time value and formats it according to the specified style. Date and time formats in Appendix 1. |
| {value, datetime, *date-style*, *time-style*}  {value, datetime, medium, short} | Accepts a date and time and formats it according to the specified date style and the specified time style. Only the basic “short”, “medium”, “long”, and “full” styles are supported here. |
| {value, format, *style*}  {value, format, %15s} | Apply a “String.format” style format to the value. If the value is null, an empty string is returned. |
| {value, number, *style*}  {value, number, ¤#,##0.00} | Apply a number format to the value. Available number formats are listed in Appendix 2. |

## “Here Documents”

A “here document” takes the form:

{>> *marker*} … any text … {*marker*}

Where *marker* is any identifier that such that “{*marker*}” does not appear in the text. The text inside the document is outputted exactly as it is present.

For example:

{ >> !}<% {value} %>{!}

Produces the literal output, with HTML escaping:

&lt;% {value} %&gt;

“Here documents” may also be used to specify a style that contains troublesome patterns. For example:

{value, datetime, >> X} {YYYY}-{MM}-{DD} {X}

Produces the output:

{2020}-{05}-{23}

## Escaping special characters

The escape style is applied to all values and “here documents”. Literal text in the stencil is not escaped. There are many built-in escape styles and advanced users can add others.

To specify an escape style one can either use the set directive:

[set escape = *style*]

Or one can specify it just for a single value:

{*style* : value,format,%15s}

All values take this optional escape style specifier.

Built-in escape styles are listed in Appendix 3

# Stencil Compendia

Stencils are stored in compendium files. They typically have a “.comp” suffix.

A compendium is a text file that can be loaded from the class path. Each compendium can contain multiple stencils. Each stencil in the file is preceded by its identifying path.

Normally any line starting with a '/' is assumed to be a path specification.

Whitespace after a stencil is normally ignored.

A path:

* Starts with a ‘/’.
* Does not end with a ‘/’, unless it is just “/”.
* Cannot contain these characters: ‘{‘, ‘}’, ‘<’, ‘>’, ‘[‘, ‘]’, ‘:’, ‘;’, ‘\’

Example:

/this/is/a/path

This is the text of a stencil.

It carries on until a line starts with a ‘/’.

/this/is/another/stencil >> END-OF

This is a second stencil, using a “here document”.

/this/is/NOT/A/PATH

The above line does not start a new stencil because we are still within the “here document”.

The following line ends the “here document”.

END-OF

/this/is/a/third/stencil

A third stencil in this compendium.

/this/has/trailing/whitespace >> END-OF

This stencil has preserved trailing whitespace because it is inside a “here document”.

END-OF

The end of a “here document” is indicated by putting the marker on its own on a line. Trailing whitespace before the marker is preserved. Trailing whitespace after the marker is ignored.

# Common Functions

## The “index” function

The index function is used to provide useful additional values inside the “[loop]” directive.

[loop data]

[apply index]

Write information about the data.

[end]

[loop data]

[apply index(,10)]

Write information about the data with a page size of 10.

[end]

[loop data]

[apply index(prefix,10)]

Write information about the data with a page size of 10.

The new values are prefixed with “prefix”.

[end]

[loop data]

[apply index(prefix)]

Write information about the data with no paging.

The new values are prefixed with “prefix”.

[end]

The following values are added to the loop’s data context:

|  |  |
| --- | --- |
| **Property** | **Description** |
| isFirst | True if this is the first entry in the loop |
| isLast | True if this is the last entry in the loop |
| index1 | The index plus 1, for a one-based index instead of a zero based index |
| isEven | True if the index is even |
| isOdd | True if the index is odd |

If a page size has been set, then the following additional values are also set:

|  |  |
| --- | --- |
| **Property** | **Description** |
| isFirstOnPage | True if this is the first entry on the current page |
| isLastOnPage | True if this is the last entry on the current page |
| pageRow | The row on the page (starting from 1) |
| pageNumber | The current page number (starting from 1) |
| pageCount | The number of pages required. |

## The “is” function

The “is” function is used to compare two values and returns a Boolean.

Test if the values are equal

[apply result = is(value1, 'EQ', value2)]

Test if the values are not equal

[apply result = is(value1, 'NE', value2)]

Test if value1 is less than value2

[apply result = is(value1, 'LT', value2)]

Test if value1 is less than or equal to value2

[apply result = is(value1, 'LE', value2)]

Test if value1 is greater than value2

[apply result = is(value1, 'GT', value2)]

Test if value1 is greater than or equal to value2

[apply result = is(value1, 'GE', value2)]

Test if a value is less than 10

[apply result = is(value1, 'LT', '10')

The result of the comparison can then be used in an [if] directive.

If the values are both numbers, then they are compared as numbers. Otherwise they are compared as Strings.

## The “for” function

The “for” function is used to generate a list of integers that can be used to trigger a fixed size loop.

[apply oneToTen = for('1','11')]

[loop oneToTen] {value} [end]

[apply count = for(countSize)]

[apply count = for(start,end,step)]

The "for" function takes one to three arguments.

With one argument, the generated integers count from zero up to one less than the value given. If the value is negative or zero the generated list is empty.

With two arguments, the generated integers count from the first value up to one less than the second value. If the second value is less than or equal to the first, the generated list is empty.

With three arguments, the generated integers count from the first value to the second value in steps as indicated by the third value. The step may be negative in size.

## The Stack Trace function

The stack trace returns the result of invoking the "printStackTrace" method on a Throwable.

[apply trace = stackTrace(throwable)]

# Working with the pychain-web-base module

The pychain-web-base module uses stencils to format error messages.

The input data to the stencil is normally specified via a ParameterisedException or a ParameterisedWebException.

The input data is augmented with a lot of data from the request:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| $.now | Current data and time |
| $.isDeveloperMode | If true, sensitive security related information may be revealed in the error message to assist in debugging. |
| $.template | The name of the template being processed |
| $.request | A map of information drawn from the HttpServletRequest instance. Not all of this data may be available. |
| $.request.authType | The name of the authentication scheme |
| $.request.contextPath | The portion of the request URI containing the context specifier |
| $.request.method | The HTTP method used |
| $.request.uri | The request URI |
| $.request.queryString | The request query string |
| $.request.remoteAddress | The IP address that made the request |
| $.request.remoteHost | The name of the machine that made the request, if known |
| $.request.remoteUser | The user that made the request, if known |
| $.request.headers | The HTTP request headers as a map of lists |
| $.request.parameters | The request parameters as a map |
| $.authentication | The request authentication if available |
| $.cause | An exception that caused the request to fail, if any |

The module also includes the standard functions for use with the [apply] directive under “F.”, e.g. “F.is”, “F.index”, “F.stackTrace”.

# Appendix 1 – List of supported date and time formats.

Date and time format specifiers are case insensitive. Hyphens may be removed, or replaced by whitespace, periods, or underscores. The “iso” prefix is optional.

|  |  |  |  |
| --- | --- | --- | --- |
| Iso-basic | Iso-basic-date | Iso-local-date | Iso-offset-date |
| Iso-date | Iso-local-time | Iso-offset-time | Iso-time |
| Iso-zoned | Iso-zoned-date-time | Iso-date-time | Iso-ordinal |
| Iso-ordinal-date | Iso-week-date | Iso-instant |  |
| Rfc | Rfc-1123 | Rfc-822 | Rfc-1123-date-time |
| Short | Medium | Long | Full |

For explanations of the ISO and RFC styles, see the Java DateTimeFormatter documentation.

For explanations of the short, medium, long, and full styles, see Java FormatStyle.

The style may also be any pattern acceptable to the Java DateTimeFormatter.ofPattern method.

# Appendix 2 – List of supported number formats

The following standard number formats are supported:

Currency,  
Percent,  
Integer,  
Number

One can also specify any pattern acceptable to Java DecimalFormat.

# Appendix 3 – List of supported escape styles.

Escape styles are case insensitive. The hyphen may be removed, or replaced with a period, underscore, or whitespace.

|  |  |
| --- | --- |
| **Style** | **Description** |
| No-Escape No None | Output the text with no changes |
| HTML-Safe | Ensures there are no dangerous characters in the text but leaves valid SGML character entities untouched.  Example: <&amp;> becomes &lt;&amp;&gt; |
| HTML-Strict | Replaces all special characters with the appropriate SGML character entities.  Example: <&amp;> becomes &lt;&amp;amp;&gt; |
| ECMA | Escape the text as a ECMA Script (JavaScript, Jscript, ES6+, etc.) literal. This does not add surrounding quotes.  Example: "Hello\n" becomes \"Hello\\n\" |
| ECMA-ASCII | As ECMA, but also replaces all non-ASCII characters with numerical escapes |
| JAVA | Escape the text as a Java String literal. This does not add surrounding quotes. |
| JAVA-ASCII | As JAVA, but also replaces all non-ASCII characters with numerical escapes. |
| JSON | Escapes the text as a JSON String literal. This does not add surrounding quotes. |
| JSON-ASCII | As JSON, but also replaces all non-ASCII characters with numerical escapes. |
| URL | Escapes the text as a URL query parameter assuming UTF-8 encoding. |
| LOG | Escapes the text as a safe log message (see below) |

The difference between ECMA, Java and JSON encoding is subtle, and shown in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Character** | **ECMA** | **Java** | **JSON** |
| Backspace (0x8) | \b | \b | \b |
| Tab (0x9) | \t | \t | \t |
| Line feed (0xA) | \n | \n | \n |
| Vertical tab (0xB) | \v |  |  |
| Form Feed (0xC) | \f | \f | \f |
| Carriage Return (0xD) | \r | \r | \r |
| Double Quote (0x22) | \" | \" | \" |
| Single Quote (0x27) | \’ | \’ |  |
| Reverse Solidus (Back-slash) (0x5C) | \\ | \\ | \\ |
| Back-tick (0x60) | \` |  |  |

In ASCII mode, non-ASCII characters are replaced with numeric escapes.

If the character is within the Unicode Basic Multilingual Plane, the character is represented by a sequence like \u*xxxx* where *xxxx* represents a four digit hexadecimal value.

For example the Euro symol has Unicode code point 0x20AC and so is represented as \u20ac.

If a character is outside the Basic Multilingual Plane and the style is Java or JSON, it is represented by an escaped surrogate pair.

For example, the clown-face emoji has Unicode code point 0x1F921 and is hence represented by \ud83e\udd21.

If a character is outside the Basic Multilingual Plane and the style is ECMA, it is represented by \u{*xxxxx*}, where *xxxxx* is the appropriate hexadecimal value.

For example, the clown-face emoji is represented as \u{1f921}.

## “Log” escaping

Log escaping is used to sanitise untrusted text with minimal alteration. If the text contains no new-lines, tabs, ISO control characters, surrogates, private-use nor unassigned Unicode characters, then it is deemed safe and left unaltered.

If any form of new-line is found (CR, LF, CRLF, or LFCR) it is replaced with system line separator and all subsequent lines are prefix with a vertical bar and some white space to make it clear that it is a continuation.

If a tab character is found, it is expanded to space characters with tab-stops at every 8th character position.

Orphaned surrogates are replaced with the Unicode replacement character. Valid surrogate pairs are left untouched.

All other control, private use, and unassigned characters are replaced with the Unicode replacement character.