STS Template Engine for Cocoa

A Universal Template Engine for Cocoa with support for Conditional Template Expansion

Licensing

The STS Template Engine is licensed under the GNU General Public License Version 2. For projects and software products for which the terms of the GPL license are not suitable, alternative licensing can be obtained directly from Sunrise Telephone Systems Ltd.

Prerequisites

The STS Template Engine will work with MacOS X version 10.3.x "Panther" and MacOS X version 10.4.x "Tiger". It uses the Cocoa framework and does not require any other frameworks or libraries. With a bit of effort it could be adapted to work with earlier versions of MacOS X or with GNUstep.

Download

The STS Template Engine is available as Objective-C source code for download at

http://www.sunrise-tel.com/downloads/STSTemplateEngine.zip

Documentation

The STS Template Engine is implemented as a category of NSString and it provides four additional class methods which return NSString objects by expanding templates. Two of the provided methods create NSString objects from templates passed as NSString objects and the other two methods create NSString objects from template files passed as POSIX pathnames. Template files can be in any encoding supported by the installed system software.

Overview of Provided Methods

stringByExpandingTemplate:usingDictionary:errorsReturned:

```
+ (id)stringByExpandingTemplate:(NSString *)templateString
                 usingDictionary:(NSDictionary *)dictionary
                  errorsReturned:(NSArray **)errorLog;
```

stringByExpandingTemplate:withStartTag:andEndTag:usingDictionary:errorsReturned:

```
+ (id)stringByExpandingTemplate:(NSString *)templateString
                    withStartTag:(NSString *)startTag
                       andEndTag:(NSString *)endTag
                 usingDictionary:(NSDictionary *)dictionary
                  errorsReturned:(NSArray **)errorLog;
```

stringByExpandingTemplateAtPath:usingDictionary:encoding:errorsReturned:

```
+ (id)stringByExpandingTemplateAtPath:(NSString *)path
                         usingDictionary:(NSDictionary *)dictionary
                                encoding:(NSStringEncoding)enc
                          errorsReturned:(NSArray **)errorLog;
```

stringByExpandingTemplateAtPath:withStartTag:andEndTag:usingDictionary:encoding:errorsReturned:

```
+ (id)stringByExpandingTemplateAtPath:(NSString *)path
                            withStartTag:(NSString *)startTag
                               andEndTag:(NSString *)endTag
                         usingDictionary:(NSDictionary *)dictionary
                                encoding:(NSStringEncoding)enc
                          errorsReturned:(NSArray **)errorLog;
```

Detailed Descriptions

---

stringByExpandingTemplate:usingDictionary:errorsReturned:

```
+ (id)stringByExpandingTemplate:(NSString *)templateString
                  usingDictionary:(NSDictionary *)dictionary
                   errorsReturned:(NSArray **)errorLog;
```

General Description

Invokes method **stringByExpandingTemplate:withStartTag:andEndTag:usingDictionary:errorsReturned:** with the STS Template Engine's default tags for placeholders: `startTag` "%«" (percent sign followed by opening chevrons) and `endTag` "»" (closing chevrons). This method is source file encoding-safe.

For further information see
stringByExpandingTemplate:withStartTag:andEndTag:usingDictionary:errorsReturned:

---

stringByExpandingTemplate:withStartTag:andEndTag:usingDictionary:errorsReturned:

```
+ (id)stringByExpandingTemplate:(NSString *)templateString
                     withStartTag:(NSString *)startTag
                        andEndTag:(NSString *)endTag
                  usingDictionary:(NSDictionary *)dictionary
                   errorsReturned:(NSArray **)errorLog;
```

General Description

Returns a new NSString made by expanding `templateString`. Placeholders to be expanded by the template engine are identified by tags passed in `startTag` and `endTag`. Placeholders are passed in `dictionary` as key/value pairs, where the keys represent the placeholders and their values represent the replacement strings. If any errors ocurr during expansion `errorLog` will contain a pointer to a list of error descriptions when the method completes.

Directives

Lines starting with a % character are interpreted as comments or directives for the template engine. Directives are %IF, %IFNOT, %IFEQ, %IFNEQ, %IFDEF, %IFNDEF, %ELSIF, %ELSIFNOT, %ELSIFEQ, %ELSIFNEQ, %ELSIFDEF, %ELSIFNDEF, %ELSE, %ENDIF, %DEFINE, %UNDEF, %LOG, %ECHO and %DEBUG. Any line starting with a % character that is not part of a valid directive nor part of a start tag is treated as a comment. Comment lines are ignored, that is they will not be processed for expansion nor will they be copied to the result returned.

Conditional Expansion

The %IF, %IFNOT, %IFEQ, %IFNEQ, %IFDEF, %IFNDEF, %ELSIF, %ELSEIFNOT, %ELSIFEQ, %ELSIFNEQ, %ELSIFDEF, %ELSIFNDEF, %ELSE and %ENDIF directives are for conditional template expansion. Any %IF, %IFNOT, %IFEQ, %IFNEQ, %IFDEF or %IFNDEF directive opens a new if-block and a new if-branch within the new if-block. Any %ELSIF, %ELSIFNOT, %ELSIFEQ, %ELSIFNEQ, %ELSIFDEF or %ELSEIFNDEF directive opens a new else-if branch in the current if-block. An %ELSE directive opens an else-branch in the current if-block. An %ENDIF directive closes the current if-block. If-blocks can be nested and there is no limit on the number of nesting levels.

%IF and %IFNOT

An identifier following %IF is interpreted as a key which is looked up in `dictionary`. If the key's value represents logical true, the subsequent lines are expanded until an elsif-, else- or endif-directive is found or another if-block is opened. An identifier following %IFNOT is interpreted as a key which is looked up in `dictionary`. If the key's value does not represent logical true, the subsequent lines are expanded until an elsif-, else- or endif-directive is found or another if-block is opened. A key's value represents logical true if its all-lowercase representation is any of "1", "yes" or "true".

%IFEQ and %IFNEQ

An identifier following %IFEQ is interpreted as a key which is looked up in `dictionary` and its value is then compared to the operand that follows the key. If the key's value and the operand match, the subsequent lines are expanded until an else-if-, else- or endif-directive is found or another if block is opened. An identifier following %IFNEQ is interpreted as a key which is looked up in `dictionary` and its value is then compared to the operand that follows the key. If the key's value and the operand do not match, the subsequent lines are expanded until an elsif-, else- or endif-directive is found or another if block is opened.

%IFDEF and %IFNDEF

An identifier following %IFDEF is interpreted as a key which is looked up in `dictionary`. If the key is found in the dictionary, the subsequent lines are expanded until an else-if-, else- or endif-directive is found or another if-block is opened. An identifier following %IFNDEF is interpreted as a key which is looked up in `dictionary`. If the key is not found in the dictionary, the subsequent lines are expanded until an else-if-, else- or endif-directive is found or another if-block is opened.

%ELSIF, %ELSEIFNOT, %ELSIFEQ, %ELSIFNEQ, %ELSIFDEF and %ELSIFNDEF

An %ELSEIF, %ELSIFNOT, %ELSIFEQ, %ELSIFNEQ, %ELSIFDEF or %ELSIFNDEF directive opens an else-if branch in the current if-block. An else-if directive will only be evaluated if no prior if- or else-if branch was expanded. The expression following such an else-if directive is evaluated in the same way an expression following the corresponding if-directive is evaluated.

%ELSE and %ENDIF

An %ELSE directive opens an else branch in the current if-block. The lines following an else branch will be expanded if no prior if- or else-if branch was expanded. Lines are expanded until an %ENDIF directive is found or another if-block is opened. An %ENDIF directive closes the current if-block.

Unconditional Expansion

Any section outside any an if-block is expanded unconditionally, excluding comment lines which are always ignored. Note however, that if-blocks may be nested.

%DEFINE and %UNDEF

A %DEFINE directive followed by a key name causes that key to be temporarily added to the dictionary for the duration of the expansion. If any text follows the key name, that text is stored as the key's value, otherwise the key's value will be an empty string. If the key already exists in the dictionary then its value will be temporarily replaced for the duration of the expansion. An %UNDEF directive followed by a key name causes that key to be temporarily removed from the dictionary for the duration of the expansion.

%LOG, %ECHO and %DEBUG

A %LOG directive followed by a key will cause that key and its value to be logged to the console, which may be used for troubleshooting. The %ECHO and %DEBUG directives are ignored as they have not been implemented yet.

Placeholder Expansion

Any lines to be expanded which contain tagged placeholders are copied to the result returned with the tagged placeholders expanded, that is replaced by their values in `dictionary`. Any lines to be expanded which do not contain any placeholders are copied verbatim to the result returned. Placeholders are recognised by start and end tags passed in `startTag` and `endTag` and they are expanded by using key/value pairs in `dictionary`.

Automatic Placeholder Variables

Placeholder names starting with an underscore "_" character are reserved for automatic placeholder variables. Automatic placeholder variables are automatically entered into the dictionary by the template engine. Currently defined automatic placeholder variables are: _timestamp, _uniqueID and _hostname.

The value of _timestamp is a datetime string with the system's current date and time value formatted to follow the international string representation format YYYY-MM-DD HH:MM:SS ±HHMM at the time the method is invoked.

The value of _uniqueID is a globally unique ID string as generated by method globallyUniqueString of class NSProcessInfo. For each invocation of **stringByExpandingTemplate:withStartTag:andEndTag:usingDictionary:errorsReturned:** a new value for _uniqueID is generated.

The value of _hostname is the system's host name at the time the method is invoked.

On MacOS X 10.4 "Tiger" (and later) locale information is available through automatic placeholder variables _userCountryCode, _userLanguage, _systemCountryCode and _systemLanguage.

Reporting Parsing Errors

For every placeholder that cannot be expanded, a partially expanded line is copied to the result returned with one or more error messages inserted or appended. An NSArray containing descriptions of any errors that may have ocurred during expansion is passed back in `errorLog` to the caller. If expansion completed withough any errors, then `errorLog` is set to nil.

Pre-conditions:

- `templateString` is an NSString containing the template to be expanded.
- `startTag` and `endTag` must not be empty strings and must not be `nil`.
- `dictionary` contains keys representing placeholders to be replaced by their respective values.
- `errorLog` is a pointer to an NSArray passed by reference.

Post-conditions:

- Return value contains a new NSString made by expanding `templateString` with lines outside of %IF blocks expanded unconditionally and lines inside of %IF blocks expanded conditionally.
- If any errors are encountered during template expansion, `errorLog` will be set to an NSArray containing error

descriptions of class TEError (see STSTemplateEngineErrors.h) for each error or warning.
- If there were neither errors nor warnings during template expansion, `errorLog` is set to `nil`.

Template Parsing Error Handling

Template expansion errors are treated gracefully. Various error recovery strategies ensure that expansion can continue even in the event of errors encountered in the template and error descriptions are added to `errorLog`.

If an if-directive is not followed by an expression, the entire if-block opened by that directive will be ignored, that is all lines will be ignored until a matching endif-directive is found. An error description is added to `errorLog` and an error message is written to the console log.

If an else-if directive is not followed by an expression, the else-if branch opened by that directive will be ignored, that is all lines will be ignored until an else-if-, else- or endif-directive is found or another if-block is opened.

If any else-if-, else- or endif-directive appears without a prior if-directive, then the line with that directive will be ignored and expansion continues accordingly. An error description is added to `errorLog` and an error message is written to the console log.

If the end of the template file is reached before an if-block was closed by an endif-directive, the block is deemed to have been closed implicitly. An error description is added to `errorLog` and an error message is written to the console log.

Any placeholders for which the dictionary does not contain keys will remain in their tagged placeholder form and have an error message appended to them in the corresponding expanded line. Expansion continues accordingly. An error description is added to `errorLog` and logged to the console.

If a placeholder without a closing tag is found, the offending placeholder will remain in its incomplete start tag only form, have an error message appended to it and the remainder of the line is not processed. Expansion then continues in the line that follows. An error description is added to `errorLog` and an error message logged to the console.

When template expansion is completed and any errors have occurred, the NSArray returned in `errorLog` contains all error descriptions in the order they ocurred.

Error-conditions:

- If `startTag` is empty or `nil`, an exception TEStartTagEmptyOrNil will be raised.
- If `endTag` is empty or `nil`, an exception TEEndTagEmptyOrNil will be raised.

It is the responsibility of the caller to catch these exceptions.

---

stringByExpandingTemplateAtPath:usingDictionary:encoding:errorsReturned:

```
+ (id)stringByExpandingTemplateAtPath:(NSString *)path
                        usingDictionary:(NSDictionary *)dictionary
                               encoding:(NSStringEncoding)enc
                         errorsReturned:(NSArray **)errorLog;
```

General Description

Invokes method **stringByExpandingTemplateAtPath:withStartTag:andEndTag:usingDictionary:encoding:errorsReturned:** with the STS Template Engine's default tags for placeholders: `startTag` "%«" (percent sign followed by opening chevrons) and `endTag` "»" (closing chevrons). This method is source file encoding-safe.

For further information see **stringByExpandingTemplateAtPath:withStartTag:andEndTag:usingDictionary:encoding:errorsReturned:**

---

stringByExpandingTemplateAtPath:withStartTag:andEndTag:usingDictionary:encoding:errorsReturned:

```
+ (id)stringByExpandingTemplateAtPath:(NSString *)path
                          withStartTag:(NSString *)startTag
                             andEndTag:(NSString *)endTag
                       usingDictionary:(NSDictionary *)dictionary
                              encoding:(NSStringEncoding)enc
                        errorsReturned:(NSArray **)errorLog;
```

General Description

Returns a new NSString made by expanding the template file at `path`. This method reads the template file specified in `path` interpreted in the string encoding specified by `enc` and then it invokes method **stringByExpandingTemplate:withStartTag:andEndTag:usingDictionary:errorsReturned:** to expand the template read from the template file.

For a more detailed description see **stringByExpandingTemplate:withStartTag:andEndTag:usingDictionary:errorsReturned:**

Pre-conditions:

- `path` is a valid POSIX path to the template file to be expanded.
- `startTag` and `endTag` must not be empty strings and must not be `nil`.
- `dictionary` contains keys representing placeholders to be replaced by their respective values.
- `enc` is a valid string encoding as defined by NSStringEncoding.
- `errorLog` is a pointer to an NSArray passed by reference.

Post-conditions:

- Return value contains a new NSString made by expanding the template file at `path` with lines outside of %IF blocks expanded unconditionally and lines inside of %IF blocks expanded conditionally.
- If any errors are encountered during template expansion, `errorLog` will be set to an NSArray containing error descriptions of class TEError (see STSTemplateEngineErrors.h) for each error or warning.
- Errors that prevent the template file to be found or read will cause the method to abort the attempt to expand the template.
- If there were neither errors nor warnings during template expansion, `errorLog` is set to `nil`.

For a description of recovery from and reporting of template parsing errors see method **stringByExpandingTemplate:withStartTag:andEndTag:usingDictionary:errorsReturned:**

Error-conditions:

- If `startTag` is empty or `nil`, an exception TEStartTagEmptyOrNil will be raised.
- If `endTag` is empty or `nil`, an exception TEEndTagEmptyOrNil will be raised.
- If the template file cannot be found or opened, or if there is an encoding error with the contents of the file, one or more error descriptions of class TEError are returned in errorLog.

It is the responsibility of the caller to catch the exceptions.

Comments in the Source Code

The STS Template Engine's sources are very extensively commented, sufficient even for Cocoa novices to understand the code. All comments show intend, that is they are authoritative over the actual source code. Thus, if the comments disagree with the source code, it likely indicates a bug in the source code and with the intend expressed in the comments it should be easy to fix.

Nevertheless, if you find a bug or a discrepancy between comments and code, please do report it to Sunrise Telephone Systems Ltd.

Sample Template

```
%% Configuration template to generate X-Lite SIP profile for Asterisk
%% created 06-JUL-2005 by Sunrise Telephone Systems Ltd.
%% version 1.00
%%
%% The generated configlet must be included in /etc/asterisk/sip.conf
%%
%%
; THIS FILE HAS BEEN AUTOGENERATED %«_timestamp»
; DO NOT EDIT THIS FILE - MODIFICATIONS MAY BE OVERRIDDEN
;
[xlite] ; SIP profile for local X-Lite on %«_hostname»
port=5061
host=127.0.0.1
type=friend
qualify=no
reinvite=no
canreinvite=no
disallow=all
%IF allowUlaw
allow=ulaw
%ENDIF
%IF allowAlaw
allow=alaw
%ENDIF
%IF allowGSM
allow=gsm
%ENDIF
%IF allowILBC
allow=ilbc
%ENDIF
context=autocontext
callerid="%«username»"<%«callerid»>
;
%% END OF TEMPLATE
```

Methodology

All software developed at Sunrise Telephone Systems Ltd is developed using VDM methodology informally. That is to say all software components are defined in terms of pre-conditions, post-conditions and error-conditions before the code is being finalised. Whilst we do not use VDM-SL to define formal semantics, we follow VDM principles and define semantics in plain English instead. Throughout the coding process, all code is verified against pre-, post-, error-conditions and the associated semantics on an ongoing basis.

About Sunrise

Sunrise Telephone Systems Ltd is a PBX and VoIP systems integrator and solutions provider specialising in open source based telephony and VPN solutions. Our head office is in Tokyo, Japan and we have a presence in Australia and the Middle East. Our contact details are: 9F Shibuya Daikyo Bldg., 1-13-5 Shibuya, Shibuya-ku, Tokyo, Japan. Tel.: 03-5420-5502 Ext. 1010. You can also call us on Free World Dialup # 36505