Perl template toolkit pdf

Template Toolkit is a set of Perl modules that implement a fast, flexible, powerful, and capable template processing system. It is an input agnostic and can be used equally well to handle any kind of text documents: HTML, XML, CSS, Javascript, Perl code, plain text, and so on. However, it is most commonly used to create static and dynamic web content, so this is what we will focus on here. Although the Toolkit template is written in Perl, you don't need to be a Perl programmer to use it. It was designed to allow non-programmers to easily create and maintain template-based websites without having to tinker with writing Perl code or go crazy with cut-n-paste. However, the Toolkit template is also designed to be extremely flexible and extensible. If you're a Perl programmer, or know someone who is, then you can easily connect the Toolkit template to existing code, data, databases and web applications. In addition, you can easily expand the Toolkit template with its plug-in mechanism and other developer APIs. Whatever context you use it, the main purpose of template tools is to allow you to create a clear separation between the presentation elements of your website and everything else. If you're creating static web pages, you can use it to separate the often repetitive user interface elements on each page (heads, menus, lackeys, etc.) from the main content. If you create a dynamic web page for the front of the web application, then you'll also use it to keep the back-and-forth code completely separate from the front end of HTML templates. Either way, a clear separation of problems is what allows you to focus on one thing at a time, without other things getting in your way. And that's what the Toolkit template is all about. some useful options (see below for the full list) of my $config - INCLUDE_PATH the way - or list the Interpolat referee No qgt; 1, expand $var with a simple text POST_CHOMP No qgt; 1 - clearing the white space PRE_PROCESS qgt; the headline, prefix of each template EVAL_PERL No. - create a template object my $template - Template --$config); identify boilerplate variables to replace my $vars - var1 $value no2, var2, var3, var @list 4 , var4 , var5 ($object,) ; Include the name of the input file, or the file handle, text link, etc. my $input and 'myfile.html'; Process input pattern, replacing variables $template ($input, $vars) die $template--qgt; This documentation describes the Template module, which is Perl's direct interface to the template toolkit. It covers the use of the module and gives a brief summary of configuration and formulaic options Please see Pattern::A guide to a complete guide that goes to much greater depth about the features and use of the Toolkit template. Pattern::Tutorial also as an introductory guide to the use of boilerplate tools. METHODS The new constructor method (implemented by template:::Base basic class) instantly eliminates a new template object. A link to the hash array of configuration items can be referred as a parameter. my $tt is a new template (I INCLUDE_PATH qgt;usr/local/templates, EVAL_PERL qgt; 1,) die $Template:::ERROR, : A link to a new template object is returned or a non-protection from an error. In the latter case, an error message can be received by calling an error () as a class method or by examining the $Template:::ERROR package variable directly. my $tt is a template --'gt; new ('%config) die pattern -'gt;error (), ; my $tt - a template --'gt; new ('%config) die $Template::ERROR, ; For convenience, configuration items can also be listed as items instead of a hash massif reference. They are automatically folded into a hash massif by the designer. my $tt - Pattern -- again (INCLUDE_PATH zgt;'/tmp, POST_CHOMP 1) die $Template::ERROR, ; Process ($template, %vars, $output, %options) Process is called to process a template. The first parameter indicates the input pattern as one: the file name is relative to INCLUDE_PATH if defined; A link to a text line containing the text of a template or a link to a file handle (e.g., IO::Handle or subclass) or GLOB (e.g., zTDIN) from which you can read a template. A link to a hash massif can be referred to as a second setting that defines variable patterns. File name $tt ('welcome.tt2') to die $tt . . . - text link to $text - % INCLUDE headline %Hello world! (% INCLUDE footer); $tt--$text) die $tt--qgt; pen file (GLOB) $tt--gt; processing (DATA) die $tt---qgt;),; __END__ (% LINK headline) This is a pattern defined in the __END__ section, which is available through the DATA file handle. (% INCLUDE footer %) By default, the processed template output is printed on STDOUT. The process method then returns one to show success. The third parameter can be passed on to the process () method to determine another location of the output. This value can be one: a simple line indicating the name of the file that will be open (relatively OUTPUT_PATH if defined) and an exit written; GLOB file open ready to go; A link to a scalar (such as a text line) that attracts output/error; Link to a routine that's called, passing the output as a parameter; or any reference to an object that implements a printing method (e.g., IO::Handle, Apache::Request, etc.) to be called by passing the output generated as a parameter. Examples: $tt welcome.tt2, $vars, 'welcome.html') to die $tt.' Link to the output sub myout routine - my $output - change; $tt process ('welcome.tt2', $vars, myout) to die $tt--qgt; link to the inference line my $output ''; $tt-> ('welcome.tt2', $vars, «$output)» умереть $tt->(), ; ; Exit: $output; In the Apache/mod_perl processor: sub handler - my $req - shift; ... your code here $req $vars $file $tt $output $req... do $req--log_reason ($tt---SERVER_ERROR; return OK; After an additional third withdrawal argument, an additional link to the hash or list (name, The only option currently supported by binmode, which, when dialed on any true value, will ensure that the files created (but not any existing file handles have passed) will be installed in binary mode. either: hash help on $tt-zgt; ($infile, $vars, $outfile, binmode qgt; 1) Die $tt--,(), or: list of names, pairs of values $tt--qgt; ($infile, $vars, $outfile, binmod zgt; 1) die $tt--qgt;; in addition, the binmode argument can indicate a certain layer of IO, such as :utf8. $tt--gt; ($infile, $vars, $outfile, binmod ('gt; 'utf8') die $tt--()), the OUTPUT configuration element can be used to determine the location of the default output except STPPUT. (% OUTPUT_PATH a directory that must be attached to all the output locations listed as file names. my $tt is a new (OUTPUT OUTPUT_PATH) ... Die Pattern use OUTPUT by default (sub called) $tt-'gt; ('welcome.tt2', $vars) to die $tt--gt;(), (Write a file on '/tmp/welcome.html' $tt process ('welcome.tt2', $vars, 'welcome.html') to die $tt-qgt; Process () method returns 1 to success or undef on error. The error message generated in the latter case can be obtained by calling an error.) See also CONFIGURATION SUMMARY, which describes how bug handling can be further configured. When used as a class method, the value of the $ERROR package is returned. So the following are equivalent. my $tt - Pattern - re-run () die Pattern - my $tt - Pattern --gt; re-die $Template::ERROR, ; When you call as an object method, the value of the internal _ERROR the variable is returned, as set out by the error condition in the previous processing call. $tt ('welcome.tt2') die $tt. Errors are presented in a set of templates by Template::Exception class objects. If the process method returns a false value, the error method may be called to return the object of that class. Type () and information () methods can stick an object out to extract the type of error and string of information, respectively. The as_string method can be called to return the $type form line to $info. This method is also overloaded with a string statement, allowing you to print the object itself to return formatted lines of errors. $tt -'gt; ('somefile') to do - my $error - $tt-qgt's error(); Print the type of error: $error type (Print error information: , $error- Print $error Service Pattern module most of the effort to process templates to the basic template::Service object. This method returns a link to that object. Context::The Service Module uses a basic template::Context of the object for processing patterns during execution. This method returns a link to this object and is equivalent to $template. Pattern ($name) This method is a simple wrapping around a template::Context method with the same name. It returns a written template for the source presented as an argument. The following list gives a brief summary of each template Toolkit configuration option. See the template:::Config for more information. The STYLE pattern and ENCODING analysis options determine the coding of the symbol. START_TAG, END_TAG define tokens indicating the beginning and end of directives (default: '%' and '%)..). TAG_STYLE set START_TAG END_TAG a predetermined style (default: 'pattern' as above). PRE_CHOMP, POST_CHOMP removes the white space before/after the directives (default: 0/0). TRIM Remove lead and rear whitespace from the output pattern (default: 0). Interpolate Interpol to Interpolize variables embedded as $this or $ (default: 0). ANYCASE Allow the directive keywords in the lower case (default: 0 - UPPER only). Pattern files and INCLUDE_PATH one or more directories to find patterns. DeliMITER Delimiter for dividing paths into INCLUDE_PATH (default: ':'). ABSOLUTE Allow absolute file names, such as /foo/bar.html (default: 0). RELATIVE Allow relative file names, for example .. . /foo/bar.html (default: 0). DEFAULT default template to use when the other is not found. BLOCKS Hash array of pre-identified boilerplate blocks. AUTO_RESET turned on by default, causing BLOCK definitions to be reset each time the template is processed. Turn off so that BLOCK definitions are retained. The RECURSION flag allows you to repeat in templates (default: 0). Processing options EVAL_PERL to indicate whether to handle PERL/RAWPERL blocks (default: 0). PRE_PROCESS POST_PROCESS Name of template (s) to be processed before/after the main template. PROCESS Pattern Name (s) to be processed instead of the main template. ERROR Error Name error pattern or link to the types of hash-massive errors for templates. READ MORE output output by default or handler. OUTPUT_PATH catalog in which weekend files can be recorded. DEBUG Include Debugging Messages. Caching and compilation options CACHE_SIZE Maximum number of collected templates for cachet in memory (default: undef - cache all) COMPILE_EXT Filename extension for compiled template files (default: undef - not compiled). COMPILE_DIR of the catalog in which the compiled template files should be written (default: undef - not PLUGINS plugins and filters Link to hash array of mapping plugins for Perl packages. PLUGIN_BASE one or more basic classes that you can find plug-ins under. LOAD_PERL flag to signify regular Perl modules should downloaded if the named plugin can't be found (default: 0). FILTERS Hash mapping the names of filters to filter routines or factories. Set up and expand LOAD_TEMPLATES list of template providers. LOAD_PLUGINS list of plug-in vendors. LOAD_FILTERS list of filter vendors. TOLERANT Providers Set to Tolerate Errors Such as Failures (default: 0). SERVICE Link to a custom service facility (default: Pattern::Service). CONTEXT Link to a custom context object (default: Pattern::Context). STASH's WORK on a custom stash object (default: Pattern::Stash). PARSER Link to the user parser object (default: Template::P arser). GRAMMAR Link to custom grammar object (default: Pattern::Grammar). The following list includes a brief summary of each Template Toolkit directive. See template::Guide::D For more information. GET Rate and print variable or value. The keyword GET is optional % variable% (% hash.key %) % list.n% (% code (args) % (% obj.meth (args) % % value: $var % CALL According to GET, but without printing the result (e.g. call code) % OVER SET value % ALSO optional % variable other_variable variable literally text variable $100 interpolated text: $var list val, val, val, val, val, val, val, val, ... va-hashish - var - val, var --val, ... (% default variable and value %) INSERT Insert a file without processing content. (% INSERT legalese.txt %) PROCESS Process another template file or block and paste the output generated. Any BLOCKs template or variables defined or updated in the PROCESSed template will then be defined in the call pattern. (% process pattern%) (% PROCESS VAR and val, ... %) INCLUDE is similar to PROCESS, but using a local copy of current variables. Any BLOCKs or variables defined in the INCLUDEd template remain local to them. (% INCLUDE pattern%) (% INCLUDE pattern var and val, ... %) WRAPPER Content between WRAPPER DIRECTIVES and the relevant END directives is first evaluated, with the result stored in variable content. The named pattern is then processed according to INCLUDE. (% of WRAPPER layout%) Some mark-up of the % blah% pattern... (% END %) A simple layout template might look like something like this: Your headline here... (% content%) Your footman is here... BLOCK Identify the named template block for use BY INCLUDE, PROCESS and WRAPPER. (% BLOCK hello%) Hello World % END % % JOIN HELLO % FOREACH Repeat attached forEACH ... THE END block for each value in the list. % foreach variable in Val, Val, Val % or % variable FOREACH In the %list or variable set at % variable % (% END %) WHILE The block, enclosed between BLOK WHILE and END, is processed, while The condition is correct. (% WHILE condition%) content (% END %) IF / / / / ELSIF / ELSE Closed block is processed if the condition is correct /false. (% IF condition%) content (% ELSIF condition %) Content (% ELSE %) Content (% END %) (% UNLESS condition%) Content (%) ELSIF/ELSE according to IF, higher % content (% END %) SWITCH / CASE Multi-way switch/case statement. (% variable SWITCH%) (% CASE val1 %) content (% CASE - val2, val3 , %) content (% CASE %) or CASE DEFAULT content % (% END%) MACRO Define a named macro. % name of the directive: MACRO % % name MACRO (arg1, arg2) qlt;directive;... (% name%) Name (val1, val2) % FILTER Process attached BY FILTER ... End the block then pipes through the filter. «% FILTER имя %» (или «% FILTER имя (парамс) % » или «% FILTER псевдоним » й имя (парамс) » » » » или содержание »% END %» USE Загрузите модуль плагина (см.<Manual::Plugins), or= any= regular= perl= module= when= the= load_perl= option= is= set.= [%= use= name= %]= #= either= [%= use= name(= params=)= %]= #= or= [%= use= var=name(params=)= %]= #= or= ...= [%= name.method= %]= [%= var.method= %]= perl= rawperl= evaluate= enclosed= blocks= as= perl= code= (requires= the= eval_perl= option= to= be= set).= [%= perl= %]= #= perl= code= goes= here= $stash-=>Шаблон:: набор('foo', 10); печать foo к , $stash->'15) распечатать $context->инклюд ('footer', й var (> $val); (% END %) % RAWPERL % - raw perl code comes here, without magic, but quickly. $output some way out; (% END %) TRY /THROW / CATCH / FINAL Exception Processing. (% TRY %) Content (% THROW Type info %) % CATCH type % catch the contents of % error.type % (% error.info%) % CATCH % or % CATCH DEFAULT % content % FINAL % this block is always processed % END % NEXT Go straight to the next point in the FOREACH or WHILE cycle. (% FINAL%) The last break from the FOREACH or WHILE cycle. (% LAST%) RETURN Stop processing the current pattern and return to the pattern. (% RETURN %) STOP Stop processing all templates and return to the caller. (% STOP%) Identify a new style of tags or symbols (default: % %). (% TAGS HTML %) (% TAGS <!-- --> %) Ignored and removed. (% is a comment to the end of the line foo and 'bar' %) Placing " directly within the directive tag comments on the entire directive % SOURCE CODE REPOSITORY Source code for the Toolkit template is held in git's public git store on Github: Andy Wardley abw@wardley. VERSION Template Toolkit version 3.009, released July 13, 2020.