# PageKit 1.13

## User's Guide

### T.J. Mather

### Boris Zentner

Copyright © 2004 T.J. Mather

**Table of Contents**

1. Overview of Features
    - Model/View/Content/Controller approach to design
        - Model
        - View
        - Content
        - Controller
    - Model of OO Perl Classes
        - Base Model Class
        - Derived Model Class
        - Inheritance Hierarchy
    - XSLT and HTML::Template::XPath
        - XSLT
        - HTML::Template::XPath
    - Language Localization
        - Applying to Content: HTML::Template::XPath
        - Applying to Content: XSLT
        - Applying to Model
        - Character set translation
    - Caching
    - Component based architecture
    - Sessions
    - Authentication
    - Form Validation
    - Sticky HTML Forms
    - Multiple Views
    - On-line Editing tools
    - Error Reporting
2. Reference
    - Configuration Options

Global Attributes
Section Attributes
Server Attributes
User Attributes
View Attributes
Page Attributes
Model API
PageKit Extensions
PageKit API
PageKit Template Tags
XML content tags
Model tags
PageKit tags
Request parameters

# Chapter 1. Overview of Features

$Id: features.xml,v 1.21 2003/11/25 10:20:24 borisz Exp $

**Table of Contents**

Model/View/Content/Controller approach to design
Model
View
Content
Controller
Model of OO Perl Classes
Base Model Class
Derived Model Class
Inheritance Hierarchy
XSLT and HTML::Template::XPath
XSLT
HTML::Template::XPath
Language Localization
Applying to Content: HTML::Template::XPath
Applying to Content: XSLT
Applying to Model
Character set translation
Caching
Component based architecture
Sessions
Authentication
Form Validation
Sticky HTML Forms
Multiple Views
On-line Editing tools
Error Reporting

# Model/View/Content/Controller approach to design

PageKit follows a Model/View/Content/Controller design pattern, which is an adaption of the Model/View/Controller pattern used in many other web frameworks, including Java's Webmacro and Struts.

- The Model is the user provided classes, which encapsulate the business logic behind the web site
- View is set of PageKit Templates, or XSLT files that generate PageKit Templates
- Content is set of XML Files
- Controller is PageKit

This approach parallels the division of the job responsibilities of a large web development team. The programmers can focus on the Model, the designers on the View, and the content administrators on the (you guessed it!) Content. PageKit provides the Controller which glues everything together.

This way everybody can focus on what they do best, whether it is programming, design, or content. Since the interfaces are simple and well-defined they can easily work together without interfering with each other.

## Model

The Model is provided by Perl classes which implement the business logic that is custom to the site. These class files should be located in the `Model/` directory. Each URL is translated into a class and method automatically.

It includes support for Data::FormValidator, making the tedious task of input validation easier. To validate a form, you simply specify required fields and constraints. If there is an error, you can return to the input form, and the invalid fields automatically get highlighted in red.

## View

The View is defined by a set of PageKit Templates making up pages and their components located in the `View/` directory. These templates can be in HTML, WML, XML, or any text based format.

Alternatively, you can provide a set of XSLT files that will generate PageKit Templates.

PageKit Template is based on HTML::Template. There are different sets of tags, depending on where the data is being pulled from.

The Model Tags, <MODEL_VAR>, <MODEL_LOOP>, and <MODEL_IF>, are filled with data by the Model.

The Content Tags, <CONTENT_VAR> and <CONTENT_LOOP>, contain XPath queries to the Content XML data.

The PageKit Tags are set internally by the Controller.

It is easy to implement multiple views, such as a printable version of a web page or a co-branded site. To create a new view, simply create a directory containing template files for the view. You only have to create templates for pages and components that you wish to override. That is all views inherit from the default set of templates.

## Content

Content is stored in an XML files. It is accessed either through XPath queries from PageKit Template using HTML::Template::XPath or by applying XSLT stylesheets to transform the XML into a PageKit Template.

Language localization couldn't be easier. Simply use the `xml:lang` attribute in the tags you wish to localize. For example, to have a title available in both English and Spanish use:

```
<title xml:lang="en">Title in English</title>
<title xml:lang="es">Titulo en Español</title>
```

## Controller

The Controller is the glue that holds everything together. To deliver a page it calls the appropriate code in the Model, generates the PageKit Template from the Content if necessary, then fills in the tags in the PageKit Template.

It makes the tedious task of authentication/authorization easy, with support for cookie based logins and session inactivity timeouts. After a login or registration, the user is redirected to the protected page they originally requested, if applicable.

# Model of OO Perl Classes

Model is simply a set of OO Perl Classes whose methods get exposed to the client through URIs.

## Base Model Class

You should implement one base Model class which derives from `Apache::PageKit::Model`. You must specify the class in the model_base_class option.

In this class, you should implement the PageKit Extensions, and any methods that are common across your Derived Model Classes.

## Derived Model Class

The Derived Model Classes derive from your Base Model Class, and should contain methods for the dynamic pages on your site. You must specify the prefix of these classes in the model_dispatch_prefix option of `Config/Config.xml`.

## Inheritance Hierarchy

This figure illustrations how model classes derive from each other and the level of specialization of each class.

```
                +---------------------------------------+
                |         Apache::PageKit::Model        |                Common
                | Model code that provides an interface |       <------- across
                | to PageKit and is common to all sites |                all sites
                +---------------------------------------+
                                     |
                +---------------------------------------+
                |           MyPageKit::Common           |                Specialized
                | Model code that is particular to the  |       <------- For Site, common
                | site, but common across all pages     |                across site
                +---------------------------------------+
                        /           |           \
  +----------------------------+     +----------------------------+
  | MyPageKit::YourClass1      |     | MyPageKit::YourClassN      |    Specialized
  | Model code that is for a   | ... | Model code that is for a   | <- for set of
  | group of pages on the site |     | group of pages on the site |    Pages on site
  +----------------------------+     +----------------------------+
```

# XSLT and HTML::Template::XPath

PageKit Templates use HTML::Template::XPath to include Content from the XML files. In addition, PageKit Templates themselves can be generated from the XML using XSLT stylesheets. Currently the only supported XSLT processer is XML::LibXSLT which uses Gnome libxslt library.

You should use HTML::Template::XPath when you would like to separate some Content from the View, but do not want to go the full route of using XSLT.

On the other hand, if you are starting from scratch, have existing XML and XSLT files, or have complicated transformation needs, then XSLT is probably the way to go.

## XSLT

To use XSLT, you must place the Content XML files in the `Content/` directory, and the View XSLT files in the `View/` directory. You must specify the XSLT template that you would like to use using the `xml-stylesheet` processing instruction by placing the following on the top of your XML file:

```
<?xml-stylesheet type="text/xsl" href="my_xslt_file.xsl"?>
```

Note that the XSLT file specified in the href attribute is relative to the `View/`*`pkit_view/`* or the `View/Default/` directory.

All of the input request parameters are available to the XSLT file by using the `xsl:param` tag in the top level of the file.

The output of XSL Transformation should be a PageKit Template or a HTML/XML/WML file (which is a special case of a PageKit Template file, namely one without any PageKit tags). Note that the Data from the Model gets filled in after the XSL transformation. This is done for performance reasons - the XSL tranformation can be cached, even if the data from the model is updated.

## HTML::Template::XPath

Using HTML::Template::XPath is easy with PageKit, it is build into the PageKit Template by using <CONTENT_VAR> and <CONTENT_LOOP> tags, which contain XPath queries to the Content XML data.

The Content XML file defaults to `Content/`*`page_id`*`.xml` or can be specified using XPath's **document()** function.

# Language Localization

One of the main advantages of separating out the Content from the View is that it is easy to implement multiple languages while sharing the same look-and-feel. You may use the `xml:lang` to label which languages tags are in.

The preferred language of the user is determined as follows.

- The default language preference is set to the `Accept-Language` incoming HTTP header.
- This default value can be overridden by setting the `pkit_lang` request parameter.

## Applying to Content: HTML::Template::XPath

HTML::Template::XPath supports language localization through the use of the `xml:lang` attribute. In PageKit 1.01 and above, the algorithm for selecting the node(s) for the selected languages is as follows:

- First it attempts to use the XPath function `lang` to return the node or the set of nodes whose `xml:lang` attribute(s) are the same as the preferred language. If the node has no `xml:lang`, then the value of the `xml:lang` attribute on the nearest ancestor is used. If the node and the its ancestors have no `xml:lang` attribute, then the `default_lang` language is used.
- If no nodes are found in the preferred language, then it returns the node(s) which are in the `default_lang` language.

The algorithm in PageKit 1.00 is slightly different from the above, but follows the same basic idea.

## Applying to Content: XSLT

As of release 1.00 there is no support for Language Localization using XSLT. However I plan to offer support in one of the two following ways:

- Different source XML files, `foo.en`, `foo.de`, `foo.es`, and so on.
- PageKit will set the `<xsl:param name="pkit_lang">` tag in the XSLT stylesheet.

Comments, suggestions, and patches welcome!

## Applying to Model

To use the language settings from the Model, simply use the `pkit_lang` method.

This can be useful for selecting content from the database based on language.

## Character set translation

By default, PageKit attempts to output using default_output_charset. PageKit will attempt to translate the PageKit Templates and output passed to output_convert from default_input_charset to default_output_charset.

Note: This also applies also to the message catalog if any. So write your message catalog files in default_input_charset.

PageKit attempts to translate the output to a character set that is specified in the `Accept-Charset` header. If this is not possible, PageKit delivers the page in the default_output_charset.

# Caching

Currently PageKit supports on-disk caching of compiled PageKit Templates. Future versions should also include in-memory caching and caching of Model output.

XSL transformations are also cached, taking into account any input request parameters that are used in `xsl:param` tags contained in the top-level the XSLT file.

# Component based architecture

Components are similar to Server Side Includes (SSIs), in that the include PageKit Templates inside other PageKit Templates. However, the can also have code associated with them that fills in the Model Tags, including <MODEL_VAR>, <MODEL_LOOP>, and <MODEL_IF>.

The PKIT_COMPONENT tag specifies where the component should be included. The Component can either be a PageKit Template file, or a PageKit Template generated from the `Content/`*`component_id`*`.xml` XML file.

The following example illustration how absolute and relative paths work:

```
# absolute path - includes View/pkit_view/foo/bar.tmpl
# or output generated from Content/foo/bar.xml
<PKIT_COMPONENT NAME="/foo/bar">
# relative path, includes the bar.tmpl or bar.xml in the directory
# of the enclosing PageKit Template/XML File.
<PKIT_COMPONENT NAME="bar">
```

# Sessions

PageKit uses a subclass of Apache::SessionX to provide sessions. It sets a cookie named `pkit_session_id` with an expiration of `session_expires`, if applicable.

To access the session, use

```
# gets hash tied to Session
my $session = $model->session;
# gets session value
my $count = $session->{'count'};
$count++;
# sets session value
$session->{'count'} = $count;
```

PageKit takes care of opening and closing sessions. Note that sessions are only created when something is written to session hash.

## Warning!

Because of the way Apache::SessionX works, the session only gets saved if top level element in the tied hash gets changed.

```
# session will not be saved, if 'foo' was already in $session.
my $session = $model->session;
$session->{'foo'}->{'bar'} = 1;

# session will be saved
my $session = $model->session;
$session->{'foo'}->{'bar'} = 1;
# since 'baz' is top level element and is assigned, session will be saved
$session->{'baz'} = 1;
```

To store information during a request, use the `pnotes` method.

As of PageKit 1.05, there is support for associating sessions with authentication. The associated session ID may be specified by the second argument returned my pkit_auth_session_key. When a user logs in, their current session may be merged with the session stored with their user ID. To override the default behavior, use the pkit_merge_sessions hook.

As of PageKit 1.08, there is support for page based sessions.

# Authentication

When a user logins in, the `pkit_login` request parameter must be set to a true value. In addition, if you want the user to be redirected to another page, set the `pkit_done` parameter. To do this, place the following hidden fields in your login form page:

```
<!-- Login Page -->
#  will get set by pagekit to the page the user is requesting
<input type="hidden" name="pkit_done">
<input type="hidden" name="pkit_login" value="1">
```

If `pkit_login` is set to a true value, then PageKit calls `pkit_auth_credential` method. If this method returns a session key, then PageKit redirects to the page specified by `pkit_done`, setting the cookie `pkit_id` to the session_key.

While the user is logged in, PageKit checks the session_key by using the `pkit_auth_session_key` method. If the `pkit_logout` request parameter is set, then the user is logged out.

PageKit access to pages based on the `require_login` attribute. If `require_login` is set to *recent*, then PageKit requires that session is currently active in the last `recent_login_timeout` seconds.

Note, that the pages `default_page`, `verify_page` and `login_page` can not be protected in any way.

# Form Validation

PageKit uses Data::FormValidator to provide easy form validation. Highlights fields in red that user filled incorrectly by using the PKIT_ERRORFONT tag. In addition, error message(s) are displayed using the PKIT_MESSAGES tag. To use, pass an input profile to the `pkit_validate_input` method.

In addition, you may implement your own custom error handling by using `pkit_set_errorfont` to set the PKIT_ERRORFONT tags.

# Sticky HTML Forms

PageKit uses HTML::FillInForm to fill in HTML Forms with the request parameters. You can turn this feature off by setting `fill_in_form` to *no*.

One useful application is if you have set up error handling and if an user submits an HTML form without filling out a required field, PageKit will re-display the HTML form with all the form elements containing the submitted info.

In additon to filling in request parameters, you may fill in the HTML fields from the model by using the `fillinform` method.

# Multiple Views

Any page can have multiple views, by using the `pkit_view` request parameter. One example is Printable pages. Another is having the same web site branded differently for different companies. Another is having different Media outputs such as HTML, XML and WML, by using the content_type configuration options.

To create a new view, create a `View/`*`pkit_view`* directory and place the PageKit Templates and XSLT files for the pages and components that you wish to apply the view to. Note that if PageKit doesn't find a template or XSLT file in the `View/`*`pkit_view`* directory it looks in the `View/Default` directory. That is, the files `View/`*`pkit_view`* "override" the files in `View/Default` directory.

To association a media output such as XML, WML, or PDF with a view, use the View content_type attribute. Note that in order for PDF output to work, you must install the Apache XML FOP processor, available from http://xml.apache.org/fop/, and configure fop_command to point to the FOP processor.

You may set the `pkit_view` request parameter in the request URI or by using `$model->input(pkit_view => `*`pkit_view`*`);` in your model code.

# On-line Editing tools

PageKit supports a set of simple on-line editing tools. To enable, set `can_edit` in your server config. You will also need to call in your Model code:

```
$model->output(pkit_admin => 1);
```

# Error Reporting

PageKit uses Apache::ErrorReport to report errors. It reports warnings and fatal errors to screen or e-mail. Includes detailed information including error message, call stack, uri, host, remote host, remote user, referrer, and Apache handler.

To use, place the following in the your `httpd.conf` file:

```
PerlModule Apache::ErrorReport
PerlSetVar ErrorReportHandler display
```

If `ErrorReportHandler` is set to **display**, errors will be displayed on the screen for easy debugging. This should be used in a development environment only.

If `ErrorReportHandler` is set to **email**, errors will be e-mailed to the site adminstrator as specified in the Apache `ServerAdmin` configuration directive. This should be used on a production site.

# Chapter 2. Reference

$Id: reference.xml,v 1.65 2003/10/07 13:29:54 borisz Exp $

**Table of Contents**


Configuration Options
- Global Attributes
- Section Attributes
- Server Attributes
- User Attributes
- View Attributes
- Page Attributes

Model API
- PageKit Extensions
- PageKit API

PageKit Template Tags
- XML content tags
- Model tags
- PageKit tags

Request parameters

# Configuration Options

## Global Attributes

These settings are global in the sense that they apply over all servers, views, and pages. They are attributes of the <GLOBAL> tag in `Config/Config.xml`.

cache_dir

> Specifies the directory where the PageKit Template cache files are stored. Defaults to `View/pkit_cache`.

cookies_not_set_page

> This is the page that gets displayed if the user attempts to log in, but their cookies are not enabled. Defaults to `login_page`.

default_errorstr

> Default errorstr, that PageKit use for PKIT_ERRORSTR, pkit_set_errorspan and the obsolete pkit_set_errorfont.
>
> Defaults to *#ff0000*.

default_input_charset

> Default charset that PageKit Templates and Model output are encoded in. Defaults to *ISO-8859-1*. PageKit uses this to convert the PageKit templates and output from output_convert to UTF-8.

default_output_charset

> Default charset that PageKit templates compiled to. Defaults to *ISO-8859-1*. This should be the charset that supports your default_input_charset and has good support among the client's browsers.

default_lang

> Default language outputed when no language is specified or request language is not available. Defaults to *en*.

default_page

> Default page user gets when no page is specified. Defaults to *index*.

errorspan_begin_tag

> Specifies the start tag for <PKIT_ERRORSPAN ... > and <PKIT_ERRORFONT ... >. Defaults to *<font color="<PKIT_ERRORCOLOR>">*.

```
errorspan_begin_tag = "&lt;span class=error&gt;"
errorspan_end_tag = "&lt;/span&gt;"
```

See also `errorspan_end_tag`.

errorspan_end_tag

Specifies the end tag for <PKIT_ERRORSPAN ... > and <PKIT_ERRORFONT ... >. Defaults to *</font>*.

See also `errorspan_begin_tag`.

fop_command

Command line that should be used to run Apache XML FOP to generate PDF output. PageKit will append FO file and PDF file arguments at end.

```
# some example fop command lines:
fop_command = "/usr/java/jdk1.2.2_02/bin/java -cp /usr/local/java/lib/fop org.apache.fop.apps.Commandline"
fop_command = "/usr/java/jdk1.2.2_02/bin/java -cp /usr/local/java/lib/fop org.apache.fop.apps.Commandline"
fop_command = "/usr/java/jdk1.2.2_02/bin/java -cp /usr/local/java/lib/fop org.apache.fop.apps.Fop"
fop_command = "/usr/java/bin/java -classpath /opt/fop-0.20.3/lib/xerces-2.0.3.jar:/opt/fop-0.20.3/lib/xalan2jcompat.jar:/opt/fop-0.20.3/lib/xalan-2.6.0.jar:/opt/fop-0.20.3/lib/xalan-2.2.3.jar:/opt/fop-0.20.3/lib/avalon-logkit-1.0b4.jar:/opt/fop-0.20.3/lib/batik-extensions.jar:/opt/fop-0.20.3/lib/batik.jar:/opt/fop-0.20.3/lib/avalon-framework-4.0.jar:/opt/fop-0.20.3/lib/ant.jar:/opt/fop-0.20.3/lib/ant-1.5-optional.jar:/opt/fop-0.20.3/build/fop.jar:org.apache.fop.apps.Fop"
```

Can be overriden by specifying the server fop_command configuration option.

gzip_output

If set to *all*, output is gzipped dynamic and static content for browsers that send a `Accept-Encoding` header containing `gzip`. If set to *static*, output is gzipped for static pages only. Defaults to *none*.

login_page

Page that gets displayed when user attempts to log in. Defaults to *login*.

logout_kills_session

When pkit_logout is called it normally resets just the pkit_id cookie. With logout_kills_session enabled (set to yes), pkit_logout will also reset pkit_session_id and forcing a new pkit_session_id cookie to be set with a new session_id. Defaults to *yes*.

model_base_class

Specifies the base Model class that typically contains code that used across entire the web application, including methods for authentication and connecting to the database.

If you have multiple PageKit applications running on the same mod_perl server, then you'll need to specify a unique `model_base_class` for each application.

Defaults to `MyPageKit::Common`.

model_dispatch_prefix

This prefixeds the Derived Model Classes. Defaults to MyPageKit::MyModel.

Methods in this class take an derived Apache::PageKit::Model object as their only argument.

not_found_page

Error page when page cannot be found. Defaults to `default_page`.

page_session

Sets the default for all non static pages. If set to *yes*, every non static page gets a unique session. Defaults to *no*. This value is overridden with `page_session`.

page_session_class

Name for the Module, that is used to create the page_session objects. Defaults to *Apache::SessionX*.

post_max

Maximum size of file uploads, in bytes. Defaults to 100,000,000 (100 MB).

upload_tmp_dir

Temporary directory for file uploads. Defaults to whatever libapreq finds usefull. This options is only usefull if you use libapreq >= 1.0. The temporary directory usually needs to reside on the same filesystem as the location supplied to the upload object's

`link`

method. See the Apache::Request documentation for further information.

protect_static

If set to *yes* static files can also be protected with the require_login attribute in the SECTION or PAGE tags. Set this option to *no* to be compatible to PageKit < 1.09. Defaults to *yes*.

recent_login_timeout

Seconds that user's session has to be inactive before a user is asked to verify a password on pages with the `require_login` attribute set to *recent*. Defaults to 3600 (1 hour).

relaxed_parser

If set to yes, this option allows template tags to be placed inside HTML comments. It also permits spaces and newlines within the tag itself. This option may be useful to HTML authors who would like to validate their templates' HTML syntax prior to processing, or who use DTD-aware editing tools. Defaults to *no*.

```
relaxed_parser = "yes"

# these tags are all allowed if relaxed_parser is enabled:
<MODEL_VAR NAME="x">
< MODEL_VAR NAME = 'x' >
< MODEL_LOOP NAME = x >
< PKIT_COMPONENT NAME = x />
<!-- MODEL_VAR NAME="x" -->
<!--   CONTENT_VAR   NAME   =    "x"   /-->
```

request_param_in_tmpl

If set to yes, then `<MODEL_VAR>` tags in template automatically get filled in with corresponding request parameters across all pages. Can be overriden by the corresponding page attribute. Defaults to *no*.

session_class

Name for the Module, that is used to create the session objects. Defaults to *Apache::SessionX*.

session_expires

Sets the expire time for the cookie that stores the session id on the user's computer. If it is not set, then the expire time on the cookie will not be set, and the cookie will expire when the user closes their browser.

```
session_expires = "+3h"
```

template_class

Name for the Module, that is used to create the template objects. Defaults to *HTML::Template*.

uri_prefix

Prefix of URI that should be trimmed before dispatching to the Model code.

See also pkit_fixup_uri in the Model API.

use_locale

If set to *yes* pkit translates the original message to the language of the client if possible. If reload is set to *yes*, the translationtables are reloaded on every usage else only on first usage.

Defaults to *no*. See also pkit_gettext and pkit_gettext_message in the Model API.

verify_page

Verify password form. Defaults to `login_page`.

## Section Attributes

These options are global across each server and all Views. They are in the <SECTIONS> tag of `Config/Config.xml`.

All Page attributes are valid. <SECTION> Tags with the longest matching part of the id attribute provide the defaults for pages without the attribute in question. <PAGE> attributes written in the <PAGE> tag have the highest priority. If nothing is found the <SECTION> Tags are scanned for a default. The closest match wins and the search is over. See the line with /xyz in the example bellow, if we request /xyz/abc, the search ends at the line with id='/xyz' and the page does not require a login!

```
<SECTIONS>
<SECTION id='/' require_login='yes'/>
<SECTION id='/xyz' request_param_in_tmpl='yes'/>
<SECTION id='/free' require_login='no'/>
</SECTIONS>
```

id

This is part of the uri path to the page where the defaults take place. '/' Means the whole site. This attribute is needed in every <SECTION> tag.

## Server Attributes

These options are global over all pages, but are local to each server configuration (e.g. production, staging, development). They are located in the <SERVERS> tag of `Config/Config.xml`.

can_edit

If set to *yes*, enables on-line editing tools.

cookie_domain

Domain for that cookies are issued. Note that you must have at least two periods in the cookie domain.

If not set then cookie issued for the current domain name.

cookie_postfix

The content of `cookie_postfix` is appended to `pkit_id` and `pkit_session_id`. With this option it is possible to force different names for your cookies. Usefull if for some reason your cookies overwrite eachother.

fop_command

Command line that should be used to run Apache XML FOP to generate PDF output. PageKit will append FO file and PDF file arguments at end.

```
# some example fop command lines
fop_command = "/usr/java/jdk1.5.0_02/bin/java -cp /usr/local/jasperlibs/fop org.apache.fop.apps.Commandline"
fop_command = "/usr/java/jdk1.5.0_02/bin/java -cp /usr/local/jasperlibs/fop org.apache.fop.apps.Commandline"
fop_command = "/usr/java/jdk1.5.0_02/bin/java -cp /usr/local/jasperlibs/fop org.apache.fop.apps.Fop"
fop_command = "/usr/lib/jvm/bin/java -classpath /opt/fop-0.20.5/lib/avalon-1.1.5.jar:/opt/fop-0.20.5/lib/avalon-framework-4.0.jar:/opt/fop-0.20.5/lib/batik.jar:/opt/fop-0.20.5/lib/bsf.jar:/opt/fop-0.20.5/lib/commons-io-1.3-optional.jar:/opt/fop-0.20.5/build/fop.jar org.apache.fop.apps.Fop"
```

html_clean_level

Sets optimization level for HTML::Clean. If set to 0, disables use of HTML::Clean. Levels range from 1 to 9. Level 1 includes only simple fast optimizations. Level 9 includes all optimizations. Defaults to *0*. A good choice for production is 3. Higher Levels might end up in wrong pages.

reload

If set to *yes*, check for new view, content and config xml files on each request. Should be set to *no* on production servers. Default is *no*.

If you change this from *no* to *yes* it is certainly that the server must be restarted to notice the change.

## User Attributes

These options are like the <GLOBAL> tag. With the difference, that you can store any information here that you like. The are located in the <USER> tag of `Config/Config.xml`.

You can retrieve this information with the pkit_get_config_attr function.

## View Attributes

These options are local to each view, but are global across servers and pages. Currently only the output media can be set, but there are plans to have a `parent_view`, so that there can be multiple levels of derived views.

content_type

> Sets the content type of the output sent to the client. Content types are strings like "text/plain", "text/html" or "application/xml". This corresponds to the "Content-Type" header in the HTTP protocol. The following content types have default views and/or special handing associated with them:

| **content_type** | **Description** | **Default Views** | **Special Handing** |
|---|---|---|---|
| text/html | HTML output. This is for traditional screen browsers such as Netscape and Internet Explorer. | This is the default content_type for all views except for those listed below | PageKit translates sets the charset in the Content-Type headers and translates the output according to the `Accept-Charset` header. |
| application/pdf | PDF Output, for display in Acrobat Reader. Uses Apache XML FOP | pdf | PageKit uses Apache XML FOP to generate the PDF, if fop_command is set. |
| text/vnd.wap.wml | WML output. This is for WAP handhelds such as Palmpilots and cellphones. | wml | *None* |
| application/xml | XML output, for Internet Explorer 5.0 and above. | xml | *None* |

## Page Attributes

These options are local to each page on the site, but are global across each server and all views. The are located in the <PAGES> tag of `Config/Config.xml`.

browser_cache

If set to *no*, sends an Expires = -1 header to disable client-side caching on the browser.

content_type

Sets the content type for output. Overrides the View content_type configuration option.

fill_in_form

When set to *yes*, automatically fills in HTML forms using HTML::FillInForm with values from the request parameters when it detects a `<form>` tag. Default is *yes*.

id

Page ID for this page.

page_session

If set to *yes*, this page gets a unique session. Defaults to *no*. This value overriddes whatever you have requested in `page_session`.

request_param_in_tmpl

If set to yes, then `<MODEL_VAR>` tags in template automatically get filled in with corresponding request parameters. Defaults to the value set by the global request_param_in_tmpl attribute, or if that is not set, then it defaults to *no*.

require_login

If set to *yes*, page requires a login. If set to *recent*, page requires a login and that the user has been active in the last `recent_login_timeout` seconds. Default is *no*.

uri_match

Value should be a regular expression. Servers requests whose URL (after the host name) match the regular expression. For example, `^member\/\d*$` matches `http://yourdomain.tld/member/4444`.

use_template

If set to *yes*, uses HTML::Template files. If set to *no* page code is responsible for sending output. Default is *yes*.

# Model API

## PageKit Extensions

pkit_dbi_connect - Should generate and return DBI database handler, $dbh, which can be accessed by rest of Model using the dbh method.
pkit_on_error - Can be used to catch any fatal error condition.

pkit_session_setup - Returns Session setup arguments.
pkit_auth_credential - Verifies login credentials and returns session key
pkit_auth_session_key - Verifies a session key and returns the user ID and session ID
pkit_common_code - Code common across site.
pkit_post_common_code - Code common across site, executed after rest of code.
pkit_cleanup_code - Cleanup code at the end of request.
pkit_fixup_uri - filter uri for conversion into page IDs
pkit_get_config_attr - Returns the value of the attr you ask for.
pkit_get_default_page - Returns "index" page when no page_id is specified.
pkit_merge_sessions - Merges current session with session associated with login when user first logs in.
pkit_output_filter - Filters output generated by PageKit
pkit_startup - Class method called at server startup.
pkit_default - Default class method

These methods should be defined in your base module as defined by model_base_class in `Config/Config.xml`.

# Name

pkit_dbi_connect — Should generate and return DBI database handler, `$dbh`, which can be accessed by rest of Model using the `dbh` method.

# Synopsis

```
sub pkit_dbi_connect {
 return DBI->connect("DBI:mysql:db","user","passwd");
}
```

# Description

This method will probably be replaced when request-based sessions are implemented.

# Name

pkit_on_error — Can be used to catch any fatal error condition.

# Synopsis

```
## inside your Common.pm, maybe also in your Modelclass to overwrite the one in Common.pm

sub pkit_on_error {
  my ( $model, $err ) = @_;
  my $page_id = $model->pkit_get_page_id;

  if ( $err =~ /DBI/ ) {
    # prevent endless loop if the error happened on our errorpage.
    return $err if ( $page_id =~ /db_error_page$/ );
    $model->pkit_redirect('db_error_page');
    return;
  }

  $model->pkit_message( $err );
```

```
  my $default_page = $model->pkit_get_default_page;
  return $err if ( $page_id =~ /^$default_page$/ );
  $model->pkit_redirect($default_page);
  return;
}
```

# Description

pkit_on_error main purpose is to catch errors for some reason. Maybe you can not get a database connection or your page can not be delivered in a special charset or anywhere inside your modelcode an uncatched die "for unknown reason"; is executed.

If nowhere else in your code this error is handled, pkit_on_error is called with the error message as second parameter and model as the first. This is your chance to deliver a working page. At best with pkit_redirect or pkit_send.

If an error happened inside pkit_on_error it is passed thru Apache and you get the internal server error as before. pkit_on_error returns the error_msg that is logged. So you can change the messsage if you like. If the returned error_msg is ’’ or undef nothing is logged and pkit assumes, that the error is repaired. If you like to log anyway add a warn $error_msg"; inside your pkit_on_error routine.

Also pkit_on_error is no replacement for Apache::ErrorReport. pkit_on_error is more a runtime error catcher for most cases. A::E reports errors more clear with stacktrace and so on. Both can happy live together.

Another advantage is that you need only pkit_on_error to catch the error even if your application has millions of cases where it could fail. There is no need to catch them all.

If pkit_on_error is not defined in your modelclass or Common.pm, it is not used.

pkit_on_error catch only fatal errors (die) NOT warnings.

If your code likes to throw an error let it die.

# Name

pkit_session_setup — Returns Session setup arguments.

# Synopsis

```
# here a MySQL example:
sub pkit_session_setup {
  my $model = shift;
  my $dbh = $model->dbh;
  return {
      session_lock_class => ’MySQL’,
      session_store_class => ’MySQL’,
      session_args => {
                      Handle => $dbh,
                      LockHandle => $dbh,
                      },
      };
}
```

```
# this one is if you prefer PostgreSQL
sub pkit_session_setup {
  my $model = shift;
  my $dbh = $model->dbh;

  my %session_setup = (
      session_store_class => 'Postgres',
      session_lock_class => 'Null',
      session_serialize_class => 'Base64',
      session_args => {
                       Handle => $dbh,
                         IDLength => 32,
                         Commit => 0,
                        }
  );
  return \%session_setup;
}
```

# Description

Method must return a hash reference using Apache::SessionX session setup arguments. This hash reference should contain the following key/value pairs:

session_store_class
: The object store class that should be used for Apache::SessionX session handling.

session_lock_class
: The lock manager class that should be used for Apache::PageKit::Session session handling.

session_args
: Reference to an hash containing options for the `session_lock_class` and `session_store_class`

# Name

pkit_auth_credential — Verifies login credentials and returns session key

# Synopsis

```
sub pkit_auth_credential {
  my ($model) = @_;

  # in this example, login and passwd are the names of the credential fields
  my $login = $model->input('login');
  my $passwd = $model->input('passwd');

  # create a session key
  # your code here.........

  return $ses_key;
}
```

## Description

Verifies the user-supplied credentials and return a session key. The session key is a string that is stored on the user's computer using cookies. Often you'll use the user ID and a MD5 hash of a a secret key, user ID, password.

Note that the string returned should not contain any commas, spaces, or semi-colons.

## Name

pkit_auth_session_key — Verifies a session key and returns the user ID and session ID

## Synopsis

```
sub pkit_auth_session_key {
  my ($model, $ses_key) = @_;

  # check whether $ses_key is valid, if so return user id in $user_id
  # your code here.........

  return $ok ? ($user_id, $session_id) : undef;
}
```

## Description

Verifies the session key (previously generated by auth_credential) and returns the user ID, and session ID.

The returned user ID will be fed to `$r->connection->user`.

The returned session ID will used to retreive the session from the database.

## Name

pkit_common_code — Code common across site.

## Synopsis

```
sub pkit_common_code {
  my $model = shift;

  # code that should be executed for every page on your site here...
}
```

## Description

Code that gets called before the page and component code for every page on the site.

# Name

pkit_post_common_code — Code common across site, executed after rest of code.

# Synopsis

```
  sub pkit_post_common_code {
    my $model = shift;

    # code that should be executed for every page on your site here...
  }
```

# Description

Code that gets called after the page and component code is executed. Note that this is experimental and may change in future releases.

# Name

pkit_cleanup_code — Cleanup code at the end of request.

# Synopsis

```
sub pkit_cleanup_code {
  my $model = shift;
  my $dbh = $model->dbh;
  $dbh->disconnect;
}
```

# Description

One use for this is to cleanup any database handlers:

Although a better solution is to use Apache::DBI.

Please note, that the `session` and `page_session` references are already deleted here. You can not use `$model->session` and `$model->page_session` inside the hook.

# Name

pkit_fixup_uri — filter uri for conversion into page IDs

# Synopsis

```
  sub pkit_fixup_uri {
    my ($model, $uri) = @_;

    $uri =~ s!^/pagekit!!;
    return $uri;
  }
```

## Description

Pre-processes the URI so that it will match the page_id's used by PageKit to dispatch the model code and find the template and content files.

In the example listed above, the request for `http://yourwebsite/pagekit/myclass/mypage` would get dispatched to the mypage method of the myclass class, and the `View/Default/myclass/mypage.tmpl` template and/or the `Content/myclass/mypage.xml` XML file.

Note: that no session and page_session data is available in this function for reading or writing.

See also uri_prefix.

## Name

pkit_get_config_attr — Returns the value of the attr you ask for.

## Synopsis

```
$hash_ref = $model->pkit_get_config_attr('GLOBAL');
$default_page = $model->pkit_get_config_attr( GLOBAL => 'default_page');

$hash_ref = $model->pkit_get_config_attr('SERVER');
$id = $model->pkit_get_config_attr('SERVER', 'id');

$all_pages_hash_ref = $model->pkit_get_config_attr('PAGES');
$hash_ref = $model->pkit_get_config_attr('PAGE', 'restricted');
$require_login = $model->pkit_get_config_attr('PAGE', 'restricted', 'require_login');

$all_views_hash_ref = $model->pkit_get_config_attr('VIEWS');
$hash_ref = $model->pkit_get_config_attr('VIEW', 'pdf');
$media    = $model->pkit_get_config_attr('VIEW', 'pdf', 'media');

$hash_ref = $model->pkit_get_config_attr('USER');
$location = $model->pkit_get_config_attr('USER', 'location');
```

## Description

The first argument is the sectionname in your `Config/Config.xml` this can be one of GLOBAL|SERVER|PAGES|PAGE|VIEWS|VIEW|USER. If no more arguments are given, you get a hashref with all tags as keys and all attrs as values for the section in question back. For the sections PAGE and VIEW you get a href back which values are a href to the subsections. If you call to with two params, the second parameter is the name of the key you ask for. The three parameter form is only allowed for pages and views, where the second argument is the page or view and the third is the name of the key you are interested in. If nothing is found, undef is returned. Otherwise the attr you ask for or a hashref. Note, that the value you get is the value in the Config.xml this is not ever the same as the value pagekit is working with. Ie: if you ask for pkit_get_config_attr( GLOBAL => 'default_page' ); you might get undef but PageKit uses 'index' as this is the default.

# Name

pkit_get_default_page — Returns "index" page when no page_id is specified.

# Synopsis

```
sub pkit_get_default_page {
  my ($model) = @_;

  if($model->pnotes('user_id')){
    # user is logged in, go to account page
    return 'myaccount';
  } else {
    # user not logged in, go to main page
    return 'index';
  }
}
```

# Description

If no page is specified, then this subroutine will return the page_id of the page that should be displayed. You only have to provide this routine if you wish to override the default method, which simply returns the default_page attribute as listed in the `Config/Config.xml` file.

# Name

pkit_merge_sessions — Merges current session with session associated with login when user first logs in.

# Synopsis

```
# This is the default merge method, included in Apache::PageKit::Model
# and called unless you over-ride the method in your Model class
sub pkit_merge_sessions {
  my ($model, $old_session, $new_session) = @_;
  while(my ($k, $v) = each %$old_session){
    next if $k eq '_session_id';
    $new_session->{$k} = $v unless exists $new_session->{$k};
  }
}
```

# Description

As of PageKit 1.05, sessions are associated with logins. If a user logs in, PageKit retreives the session associated with that login. This method can be used to specify how the data from the current session is merged with the data from the retreived session.

# Name

pkit_output_filter — Filters output generated by PageKit

# Synopsis

```
  sub pkit_output_filter {
    my ($model, $output_ref) = @_;
    if($model->apr->parsed_uri->scheme eq 'https'){
      $$output_ref =~ s(http://images.yourdomain.com/)(https://images.yourdomain.com/)g;
    }
  }
```

# Description

Filters the output from the PageKit handler. Should only use when necessary, a better option is to modify the templates directly.

In the example above we filter the image links to that they point to the secure site if we are on a secure page (the only good use of `pkit_output_filter` that I know of)

# Name

pkit_startup — Class method called at server startup.

# Synopsis

```
sub pkit_startup {
  my ($class, $pkit_root, $server, $config) = @_;
  my $pic_cache_dir = $config->get_global_attr('my_picture_cache') || '';
  my $unlink_sub = sub {
    -f && unlink;
  };
  File::Find::find(sub { -f && unlink }, $pic_cache_dir);
}
```

# Description

Called at server startup with PageKit root, server, and configuration object passed as parameters. Note that the configuration API may change.

# Name

pkit_default — Default class method

# Synopsis

```
sub pkit_default {
 # class default code here ...
}
```

# Description

Class method called for every page in the same class which has page code and/or content. It is called after `pkit_common_code` and before page code.

NOTE: currently this is the only pkit_* method which can exist outside Common.pm

# PageKit API

input - Get request parameters
pkit_send - Can be used to send data to the client.
pkit_component_params_hashref - Get access to the component parameters.
pkit_input_hashref - Gets all request paramaters.
fillinform - Fills in HTML Forms
ignore_fillinform_fields - Fills in HTML Forms
pnotes - Pass values from one method/handler to another
output_convert - Outputs data for display, converting charset.
output - Outputs data for display.
pkit_query - Wrapper to HTML::Template::query
apr - Returns Apache::Request object
pkit_status_code - Get or set the status code for you page.
dbh - Returns database handle
session - Gets session object
page_session - Gets page session object
pkit_message - Adds message to PKIT_MESSAGES tag
pkit_gettext_message - Adds translated message to PKIT_MESSAGES tag
pkit_gettext - Translates the text to the clients language if possible else return the messages as it was before.
pkit_internal_execute_redirect - Internal redirection to another page inside the PageKit application. Also execute the code for the new destination page.
pkit_internal_redirect - Internal redirection to another page inside the PageKit application
pkit_redirect - Redirect to another URL.
pkit_set_errorfont - Sets PKIT_ERRORSPAN tags
pkit_set_errorspan - Sets PKIT_ERRORSPAN tags
pkit_validate_input - Validates input from an HTML form.
pkit_get_orig_uri - Gets the original URI requested.
pkit_get_page_id - Gets the page ID.
pkit_get_server_id - Gets server ID.
pkit_get_session_id - Gets the session id
pkit_get_page_session_id - Gets the page session id or undef in case this page has no page_session requested.
pkit_lang - Gets Language preference of user
pkit_root - Gets PageKit root directory
pkit_user - Gets user ID of authenticated user.

Theese methods are available to the user as Apache::PageKit::Model API.

# Name

input — Get request parameters

# Synopsis

```
# get single parameter
my $value = $model->input($key);

# get all parameters
my @keys = $model->input;

# set pkit_view request parameter
$model->input(pkit_view => "printable");
```

# Description

Gets requested parameter from the Apache request object, if called without any parameters, gets all available input parameters:

# Name

pkit_send — Can be used to send data to the client.

# Synopsis

```
#
# for a filename, the media_type will be found by the MIME::Types module.
#
my $status_code = $model->pkit_send($filename);

my $status_code = $model->pkit_send(\$data, 'image/png');

# $data_ref is not zipped by the pkit_send methode, it sends
# it only with this encoding header.
my $status_code = $model->pkit_send($data_ref, 'html/text', 'gzip');

my $status_code = $model->pkit_send($fh);

$model->pkit_status_code($status_code);
```

# Description

Can be used to send data to the client. From a scalar_ref a filename or a reference to a filehandle. If the first argument is a filename, and the second argument media_type is not set, the module `MIME::Types` try to find the media_type for us depending on the suffix. If this fails or the optional second argument media_type is not set, the default `application/octet-stream` is used. If the request was only a head request, only a header is send back to the client. The optional third argument if the content_encoding (ie: gzip|x-gzip|compress|x-compress) it is only used if the media_type is html/text.

# Name

pkit_component_params_hashref — Get access to the component parameters.

# Synopsis

```
    <PKIT_COMPONENT NAME="xxx" headline="title" datasrc="select * from a_table">
    <PKIT_COMPONENT NAME="xxx" headline="another_title" datasrc="select * from another_table">
    <PKIT_COMPONENT NAME="xxx" headline="title" datasrc="select * from a_table">

sub xxx {
  my $model = shift;
  my $fields = $model->pkit_component_params_hashref;
  my $select_statement = $fields->{DATASRC};
  ...
}
```

# Description

This method return a hashref that contain the component parameters (or undef if the calling routine is not a component). All keys are uppercase and the NAME of the component is not present in the hash. If the same component is used twice or more, the component code is called only once.

# Name

pkit_input_hashref — Gets all request paramaters.

# Synopsis

```
$params = $model->pkit_input_hashref;
```

# Description

This method fetches all of the parameters from the Apache request object, returning a reference to a hash containing the parameters as keys, and the parameters' values as values. Note a multivalued parameters is returned as a reference to an array.

Note, that the values in this hash are **readonly**. To modify something use $model->`input`( key => $value );.

# Name

fillinform — Fills in HTML Forms

# Synopsis

```
$model->fillinform(email => $email);
```

## Description

Used with HTML::FillInForm to fill in HTML forms. Useful for example when you want to fill an edit form with data from the database.

## Name

ignore_fillinform_fields — Fills in HTML Forms

## Synopsis

```
  $model->ignore_fillinform_fields( qw/email name/ );
```

## Description

All fields named in this call are NOT changed by fillinform.

## Name

pnotes — Pass values from one method/handler to another

## Synopsis

```
sub pkit_auth_credential {
  # ...
  $model->pnotes(user_id => $user_id);
  # ...
}
```

## Description

Wrapper to mod_perl's `pnotes` method, used to pass values from one handler to another.

In the example above the `user_id` is set when the user gets authenticated.

## Name

output_convert — Outputs data for display, converting charset.

## Synopsis

```
      $model->output_convert(output => {foo => $utf8_text},
                             input_charset => 'UTF-8');
```

## Description

This is a wrapper to the output method. It converts the output from the character set specified by the charset argument to default_output_charset. If the character set is not specified, then default_input_charset is used.

## Examples

```
# converts from UTF-8
$model->output_convert(output => { key => $utf8_text },
                       charset => 'UTF-8');
# converts from default_input_charset
$model->output_convert(key => $text)
```

## Name

output — Outputs data for display.

## Synopsis

```
$model->output(NAME => "John Doe");
```

## Description

This is similar to the HTML::Template `param` method. It is used to set the model tags in the PageKit templates..

## Examples

```
# set multiple paramaters
$model->output(firstname => $firstname,
             lastname => $lastname);

# set multiple paramaters with hash ref
$model->output({firstname => $firstname,
             lastname => $lastname});

# sets <MODEL_LOOP NAME="foo">
#       <MODEL_VAR NAME="bar"/>
#       <MODEL_VAR NAME="baz"/>
#      </MODEL_LOOP>
$model->output(foo => [
                       {bar => $bar1, baz => $baz1},
                       {bar => $bar2, baz => $baz2},
                      ]);
```

## Name

pkit_query — Wrapper to HTML::Template::query

## Synopsis

```
my $type = $model->pkit_query(name => 'foo');
```

## Description

Basically a wrapper to the `query` method HTML::Template. Queries the template for the current page_id. Useful when you have multiple views and want to make sure that you need to hit a database.

## Name

apr — Returns Apache::Request object

## Synopsis

```
my $apr = $model->apr;
```

## Description

Returns Apache::Request.

## Name

pkit_status_code — Get or set the status code for you page.

## Synopsis

```
# get status code
my $status_code = $model->pkit_status_code;

# set status code
$model->pkit_status_code(DONE);
$model->pkit_status_code(NOT_FOUND);
$model->pkit_status_code(OK);

# return old status code
return $model->pkit_status_code(OK);

# remove previously set returncode
$model->pkit_status_code(undef);
```

## Description

Set or get the status code for your page. If you set a status code the execution of your model code ends after the peace of code that sets the status code. Even if you set the status code to OK. The status code is passed back to the webserver.

# Name

dbh — Returns database handle

# Synopsis

```
my $dbh = $model->dbh;
```

# Description

Returns a database handle, as specified by the pkit_dbi_connect method.

# Name

session — Gets session object

# Synopsis

```
my $session = $model->session;
```

# Description

Returns a hash tied to Apache::SessionX

# Name

page_session — Gets page session object

# Synopsis

```
my $page_session = $model->page_session;
```

# Description

Returns a hash tied to Apache::SessionX If this was requested in `Config/Config.xml` otherwise undef.

# Name

pkit_message — Adds message to PKIT_MESSAGES tag

# Synopsis

```
    # regular message
    $model->pkit_message("Your listing has been deleted.");
    # error message
    $model->pkit_message("You did not fill out the required fields.",
             is_error => 1);
```

## Description

Adds a message to be displayed to the user. The message can displayed using the <PKIT_MESSAGES> tag.

To add an error message, set `is_error` to a true value.

Note that the message is passed along in the URI if you perform a redirect using the pkit_redirect method.

## Name

pkit_gettext_message — Adds translated message to PKIT_MESSAGES tag

## Synopsis

```
# regular message
$model->pkit_gettext_message("Your listing has been deleted.");
# error message
$model->pkit_gettext_message("You did not fill out the required fields.",
         is_error => 1);
# pkit_gettext_message is a shortcut for
$model->pkit_message($model->pkit_gettext('the message'), ...);
```

## Description

Adds a translated message to be displayed to the user. The message can displayed using the <PKIT_MESSAGES> tag.

To add an error message, set `is_error` to a true value.

Note that the message is passed along in the URI if you perform a redirect using the pkit_redirect method. See also pkit_gettext and pkit_message.

## Name

pkit_gettext — Translates the text to the clients language if possible else return the messages as it was before.

## Synopsis

```
$model->pkit_gettext('You have successfully logged out.');
```

## Description

Translates the text to the clients language, if a .mo file in that language is found. See also pkit_gettext_message.

# Name

pkit_internal_execute_redirect — Internal redirection to another page inside the PageKit application. Also execute the code for the new destination page.

# Synopsis

```
$model->pkit_internal_execute_redirect('myotherpage');
return;
```

# Description

Resets the page_id. This is usually used "redirect" to different template inside your application and execute the modelcode for the new page.

From PageKit v1.12 on pkit_internal_redirect handle full url's and strip them. Parameters at the end of request are removed. This is usefull if you get the url from a pkit_done like parameter. See also pkit_redirect and pkit_internal_redirect

# Name

pkit_internal_redirect — Internal redirection to another page inside the PageKit application

# Synopsis

```
$model->pkit_internal_redirect('myaccount');
return;
```

# Description

Resets the page_id. This is usually used "redirect" to different template.

Note that you should perform a pkit_redirect for **POST** requests. From PageKit v1.12 on pkit_internal_redirect handle full url's and strip them. Parameters at the end of request are removed. This is usefull if you get the url from a pkit_done like parameter.

# Name

pkit_redirect — Redirect to another URL.

# Synopsis

```
$model->pkit_redirect("http://www.pagekit.org/");
```

## Description

It is strongly recommend that you use this method on pages where a query that changes the state of the application is executed. Typically these are POST queries that update the database.

Note that this passes along the messages set my pkit_message if applicable.

## Name

pkit_set_errorfont — Sets PKIT_ERRORSPAN tags

## Synopsis

```
$model->pkit_set_errorfont('state');

# possible since version 1.08
$model->pkit_set_errorfont( 'state', '#00ff00' );
```

## Description

Superfluous since PageKit 1.08 please use pkit_set_errorspan.

Sets the corresponding PKIT_ERRORSPAN tag in the template. Useful for implementing your own custom constraints. Since version 1.08 it is possible to select the errorstr. This is done with the second parameter. The color must be a string thar starts with a # followed by exactly six hexdigits. If you do not specify the errorstr, the string is taken from the `default_errorstr`. See also PKIT_ERRORSTR.

## Name

pkit_set_errorspan — Sets PKIT_ERRORSPAN tags

## Synopsis

```
$model->pkit_set_errorspan('state');

# possible since version 1.08
$model->pkit_set_errorspan( 'state', '#00ff00' );
```

## Description

Sets the corresponding PKIT_ERRORSPAN tag in the template. Useful for implementing your own custom constraints. Since version 1.08 it is possible to select the errorstr. This is done with the second parameter. The color must be a string thar starts with a # followed by exactly six hexdigits. If you do not specify the errorstr, the string is taken from the `default_errorstr`. See also PKIT_ERRORSTR.

# Name

pkit_validate_input — Validates input from an HTML form.

# Synopsis

```
# very simple validation, just check to see if name field was filled out
my $input_profile = {required => [ qw ( name ) ]};
# validate user input
unless($model->pkit_validate_input($input_profile)){
  # user must have not filled out name field,
  # i.e. $apr->param('name') = $model->input('name') is
  # not set, so go back to original form
  # if you used a <PKIT_ERRORFONT NAME="name"> tag, then it will be set to
  # red
  $model->pkit_internal_redirect('orig_form');
  return;
}
```

# Description

Takes an hash reference containing a Data::FormValidator input profile and returns true if the request parameters are valid.

# Name

pkit_get_orig_uri — Gets the original URI requested.

# Synopsis

```
my $orig_uri = $model->pkit_get_orig_uri;
```

# Name

pkit_get_page_id — Gets the page ID.

# Synopsis

```
my $page_id = $model->pkit_get_page_id;
```

# Name

pkit_get_server_id — Gets server ID.

## Synopsis

```
my $server_id = $model->pkit_get_server_id;
```

## Description

Gets the server_id for the server, as specified by the `PKIT_SERVER` directive in the `httpd.conf` file.

## Name

pkit_get_session_id — Gets the session id

## Synopsis

```
# the following two line are equivalent, with one difference, if the session does
# not exist already then the second line creates a new session. The first line
# does not.
my $session_id = $model->pkit_get_session_id;
my $session_id = $model->session->{_session_id};
```

## Description

Gets the session id if you have set up session management using pkit_session_setup.

## Name

pkit_get_page_session_id — Gets the page session id or undef in case this page has no page_session requested.

## Synopsis

```
my $page_session_id = $model->pkit_get_page_session_id;
```

## Description

Gets the page session id if you have set up session management using pkit_session_setup and this page want a page_session see: page_session and page_session.

## Name

pkit_lang — Gets Language preference of user

## Synopsis

```
my $pkit_lang = $model->pkit_lang;
```

## Description

Gets the language preference of the user, as set by `Accept-Language` incoming HTTP header or by the `pkit_lang` request parameter.

Can be used for selecting which content should be retreived from the database.

## Name

pkit_root — Gets PageKit root directory

## Synopsis

```
my $pkit_root = $model->pkit_root;
```

## Description

Gets the PageKit root directory, as defined by `PKIT_ROOT` in your `httpd.conf` file.

## Name

pkit_user — Gets user ID of authenticated user.

## Synopsis

```
my $pkit_user = $model->pkit_user;
```

## Description

Gets the user_id from `$apr->connection->user`, set with the return value of pkit_auth_session_key.

# PageKit Template Tags

PageKit Templates are templates that contain HTML::Template-esque tags. They are either stored as files in the View directory or generated from XSLT transformations. There are three main classes of tags, XML content tags, Model tags, and PageKit tags.

## XML content tags

CONTENT_VAR - Extract a node from an XML file using XPath.
CONTENT_LOOP - Extract a set of nodes from an XML file using XPath.
CONTENT_IF, CONTENT_UNLESS, CONTENT_ELSE - Displays enclosed section if content sets value to true/false.

Content tags use XPath queries to retreive data from XML Content files. The XML::LibXML and HTML::Template::XPath modules are used behind the scenes.

# Name

CONTENT_VAR — Extract a node from an XML file using XPath.

# Synopsis

```
<CONTENT_VAR NAME="document('site.xml')//title"/>
```

# Description

Uses the XPath query specified in the NAME to extract content from the XML file specified by the document function. Note that the document function is relative with respect to the page_id of the enclosing page. If document is omitted, then defaults to the content file that corresponds to the page_id of the enclosing page.

# Name

CONTENT_LOOP — Extract a set of nodes from an XML file using XPath.

# Synopsis

```
<CONTENT_LOOP NAME="document('site.xml')//name">
  <CONTENT_VAR NAME="surname"/>
  <CONTENT_VAR NAME="firstname"/>
</CONTENT_LOOP>
```

# Description

Uses the XPath query specified in the NAME to extract content from the XML file specified by the document function. If document is omitted, then defaults to the content file that corresponds to the URI being view.

# Name

CONTENT_IF, CONTENT_UNLESS, CONTENT_ELSE — Displays enclosed section if content sets value to true/false.

# Synopsis

```
<CONTENT_IF NAME="foo">
 Text to be included if foo is included in Content and is true
<CONTENT_ELSE>
 Text to be included if foo not inclued in Content or is false
</CONTENT_IF>

<CONTENT_UNLESS NAME="bar">
 Text to be included if bar is included in Content and is true
</CONTENT_UNLESS>
```

# Description

If the XPath query in the name attribute returns a true value, then the text enclosed by the CONTENT_IF tag will be included in the template.

Please note: if you ask for loop_context_vars like (__first__, __odd__, __inner__ or __last__) that they must be written in lowercase!.

## Model tags

MODEL_VAR - Filled in with scalar value set from Model's output method
MODEL_LOOP - Filled in with array reference of hash references set from Model's output method.
MODEL_IF, MODEL_UNLESS, MODEL_ELSE - Displays enclosed section if model sets value to true/false.

Model tags refer to data set by the output method of the model classes. They correspond to the HTML::Template. The PageKit pre-processor simply replaces MODEL with TMPL before running the template through HTML::Template.

# Name

MODEL_VAR — Filled in with scalar value set from Model's output method

# Synopsis

```
<MODEL_VAR NAME="foo">
```

# Description

This tag is very simple. It is replaced by the corresponding value set in the Model by the output method. You may also use ESCAPE="HTML" and ESCAPE="URL" to escape HTML, and URLs respectively.

# Name

MODEL_LOOP — Filled in with array reference of hash references set from Model's output method.

# Synopsis

```
<MODEL_LOOP NAME="foo">
  <MODEL_VAR NAME="bar"/>
  <MODEL_VAR NAME="baz"/>
</MODEL_LOOP>
```

# Description

This tag used for displaying a "table" of data. The data is set from the model using the output method, and is a reference to an array of hash references. Using the table analogy, the hash references are the rows, the keys of the hash references are the column headers, the hash values are the data.

# Name

MODEL_IF, MODEL_UNLESS, MODEL_ELSE — Displays enclosed section if model sets value to true/false.

# Synopsis

```
<MODEL_IF NAME="foo">
 Text to be included if foo is true
<MODEL_ELSE>
 Text to be included if foo is false
</MODEL_IF>

<MODEL_UNLESS NAME="bar">
 Text to be included if bar is false
</MODEL_UNLESS>
```

# Description

If the parameter in the name attribute is set to true by the `output` method of the Model, then the text enclosed by the MODEL_IF tag will be included in the template.

## PageKit tags

PKIT_COMMENT - Write comments inside your templates. The comments are stripped out of your source. Before it is delivered.
PKIT_COMPONENT - Include templates and associate model code.
PKIT_ERRORSTR - Named field for your errorstr.
PKIT_MACRO - Placeholder for components.
PKIT_ERRORFONT - Highlights Invalid Fields
PKIT_ERRORSPAN - Highlights Invalid Fields
PKIT_HAVE_MESSAGES - Include the block between <PKIT_HAVE_MESSAGES> and </PKIT_HAVE_MESSAGES> only, if one or more messages are avail.
PKIT_HAVE_NOT_MESSAGES - Include the block between <PKIT_HAVE_NOT_MESSAGES> and </PKIT_HAVE_NOT_MESSAGES> only, if no messages are avail.
PKIT_HOSTNAME - Fills in the hostname in the URL.
PKIT_MESSAGES - Display messages passed to pkit_message method.
PKIT_SELFURL - URL of current page.
PKIT_VIEW - Wraps a section of text to be displayed for a view.
PKIT_ELSE - Can be used to add a else part to PKIT_... tags.
PKIT_IS_ERROR - Used to display messages only if it is a errormessage.
PKIT_NOT_ERROR - Used to display messages only if it is a NOT a errormessage.

PageKit tags refer to data that is set by the PageKit controller. These correspond to functionality or utility functions that are common across different web applications.

# Name

PKIT_COMMENT — Write comments inside your templates. The comments are stripped out of your source. Before it is delivered.

# Synopsis

```
<PKIT_COMMENT>This is a comment!</PKIT_COMMENT>
```

# Description

The <PKIT_COMMENT> tag is used to comment the block between <PKIT_COMMENT> and </PKIT_COMMENT>. This is done in a ballanced manner, so it is possible to have nested blocks of comments.

If you have more opening tags than closing ones, the spare opening tags are seen as text in the comment, as long as they are surrounded by any close tag.

Close tags are closed as soon as possible, so if your source contains more close tags than open tags, spare close tags are in your delivered source. </PKIT_COMMENT> is available as of Apache::PageKit 1.13.

# Name

PKIT_COMPONENT — Include templates and associate model code.

# Synopsis

```
<PKIT_COMPONENT NAME="html_header">

<PKIT_COMPONENT NAME="top_10" loop=top_10_cd headline="CD charts" >
<PKIT_COMPONENT NAME="top_10" loop=top_10_maxi headline="Maxi charts" >

<PKIT_COMPONENT NAME="newsbox" news='<MODEL_VAR NAME="joke">' >
```

# Description

This has two functions. The first works like Server Side Includes, including another PageKit Template that is in the View directory or is generated from the corresponding XML file using XSLT.

The second is to associate a method from the Model the page. This is used to fill in any MODEL_VAR and MODEL_LOOP tags in the included PageKit Template.

The 2. 3. and 4. example define also some macros. These macros are also available for XSLT with the <xsl:param name="..."> tag. See `PKIT_MACRO`

## Name

PKIT_ERRORSTR — Named field for your errorstr.

## Synopsis

```
<PKIT_ERRORSTR>
<font color="<PKIT_ERRORSTR>">bla</font>
```

## Description

This tag is replaced by the string that you define with `default_errorstr`. If it is not defined the default is *#ff0000*.

See also `PKIT_MESSAGES` for a example.

## Name

PKIT_MACRO — Placeholder for components.

## Synopsis

```
<MODEL_VAR NAME="<PKIT_MACRO NAME=headline>">
<MODEL_LOOP NAME="<PKIT_MACRO NAME=loop>">
  <MODEL_VAR NAME="title"/>
</MODEL_LOOP>
```

## Description

A `PKIT_MACRO` is replaced with the value of the key in `PKIT_COMPONENT` that match the name of the macro.

This tag is only valid inside a template that is loaded with `PKIT_COMPONENT`

## Name

PKIT_ERRORFONT — Highlights Invalid Fields

## Synopsis

```
<PKIT_ERRORFONT NAME="lastname"> Last Name </PKIT_ERRORFONT> <input name="lastname">
<PKIT_ERRORFONT> new for PageKit 1.10 </PKIT_ERRORFONT>
```

## Description

PKIT_ERRORFONT is the outdated alias for PKIT_ERRORSPAN

This tag highlights fields in red that Model reported as being filled in incorrectly. Or, since version 1.10 of pkit it is possible to suppress the NAME attribute. Then the field is always colored or whatever you want.

# Name

PKIT_ERRORSPAN — Highlights Invalid Fields

# Synopsis

```
<PKIT_ERRORSPAN NAME="lastname"> Last Name </PKIT_ERRORSPAN> <input name="lastname">
<PKIT_ERRORSPAN> new for PageKit 1.10 </PKIT_ERRORSPAN>
```

# Description

This tag highlights fields in red that Model reported as being filled in incorrectly. Or, since version 1.10 of pkit it is possible to suppress the NAME attribute. Then the field is always colored or whatever you want.

# Name

PKIT_HAVE_MESSAGES — Include the block between <PKIT_HAVE_MESSAGES> and </PKIT_HAVE_MESSAGES> only, if one or more messages are avail.

# Synopsis

```
<PKIT_HAVE_MESSAGES>
            Here are some messages for you: <p>
           <PKIT_MESSAGES>
           <PKIT_IS_ERROR><font color="<PKIT_ERRORSTR>"></PKIT_IS_ERROR>
           <PKIT_MESSAGE>
           <PKIT_IS_ERROR></font></PKIT_IS_ERROR>
           <p>
  </PKIT_MESSAGES>
 </PKIT_HAVE_MESSAGES>
```

# Description

Include the block between <PKIT_HAVE_MESSAGES> and </PKIT_HAVE_MESSAGES> only, if one or more messages are avail.

# Name

PKIT_HAVE_NOT_MESSAGES — Include the block between <PKIT_HAVE_NOT_MESSAGES> and </PKIT_HAVE_NOT_MESSAGES> only, if no messages are avail.

## Synopsis

```
<PKIT_HAVE_NOT_MESSAGES>
            No messages for you! <p>
 </PKIT_HAVE_NOT_MESSAGES>
```

## Description

Include the block between <PKIT_HAVE_NOT_MESSAGES> and </PKIT_HAVE_NOT_MESSAGES> only, if no messages are avail.

## Name

PKIT_HOSTNAME — Fills in the hostname in the URL.

## Synopsis

```
<PKIT_HOSTNAME>
```

## DESCRIPTION

Returns the hostname in the URL of the page being served. Particulary useful when you have production and development servers and you need to link to a secure page.

Note that if you are running a proxy server in front of the PageKit server, you probably want to use mod_proxy_add_uri.c. PageKit will extract the hostname from the frontend server using the X-Original-URI header that mod_proxy_add_uri sets.

## Name

PKIT_MESSAGES — Display messages passed to pkit_message method.

## Synopsis

```
<PKIT_MESSAGES>
         <PKIT_IS_ERROR><font color="<PKIT_ERRORSTR>"></PKIT_IS_ERROR>
         <PKIT_MESSAGE>
         <PKIT_IS_ERROR></font></PKIT_IS_ERROR>
         <p>
</PKIT_MESSAGES>
```

```
<PKIT_MESSAGES>
  <PKIT_IS_ERROR><font color="<PKIT_ERRORSTR>"><PKIT_MESSAGE></font></PKIT_IS_ERROR>
</PKIT_MESSAGES>

<PKIT_MESSAGES>
  <PKIT_NOT_ERROR><font color="<PKIT_ERRORSTR>"><PKIT_MESSAGE></font></PKIT_NOT_ERROR>
</PKIT_MESSAGES>
```

## Description

Displays messages passed to pkit_message method in the Model code.

## Name

PKIT_SELFURL — URL of current page.

## Synopsis

```
<PKIT_SELFURL exclude="foo bar">
```

## Description

The URL of the current page, including CGI parameters, but excluding those listed in the exclude attribute. Appends a '&' or '?' at the end to allow additionial parameters.

Note that if you are running a proxy server in front of the PageKit server, you probably want to use mod_proxy_add_uri.c. PageKit will take the URL from the frontend server using the X-Original-URI header that mod_proxy_add_uri sets.

## Name

PKIT_VIEW — Wraps a section of text to be displayed for a view.

## Synopsis

```
<PKIT_VIEW NAME="print">This is text display for the printable view</PKIT_VIEW>
```

## Description

Displays the enclosed text if the pkit_view request parameter is set to NAME attribute.

## Name

PKIT_ELSE — Can be used to add a else part to PKIT_... tags.

## Synopsis

```
<PKIT_VIEW NAME="print">This is text display for the printable view<PKIT_ELSE>And this text if hte view is NOT printable</PKIT_VIEW>
```

## Description

Can be used to add a else part to PKIT_VIEW, PKIT_HAVE_MESSAGES, PKIT_HAVE_NOT_MESSAGES, PKIT_IS_ERROR or PKIT_NOT_ERROR.

# Name

PKIT_IS_ERROR — Used to display messages only if it is a errormessage.

# Synopsis

```
<PKIT_HAVE_MESSAGES>
      Here are some error messages for you: <p>
     <PKIT_MESSAGES>
     <PKIT_IS_ERROR>
       <PKIT_MESSAGE>
     </PKIT_IS_ERROR>
 </PKIT_MESSAGES>
 </p>
   </PKIT_HAVE_MESSAGES>
```

# Description

Used to display messages only if it is a errormessage. Or display errors and warnings in different places.

# Name

PKIT_NOT_ERROR — Used to display messages only if it is a NOT a errormessage.

# Synopsis

```
  <PKIT_HAVE_MESSAGES>
     Here are some warnings messages for you: <p>
    <PKIT_MESSAGES>
    <PKIT_NOT_ERROR>
      <PKIT_MESSAGE>
    </PKIT_NOT_ERROR>
</PKIT_MESSAGES>
</p>
  </PKIT_HAVE_MESSAGES>

  <PKIT_HAVE_MESSAGES>
     Here are some error messages for you: <p>
    <PKIT_MESSAGES>
    <PKIT_IS_ERROR>
      <PKIT_MESSAGE>
    </PKIT_IS_ERROR>
</PKIT_MESSAGES>
</p>
  </PKIT_HAVE_MESSAGES>
```

## Description

Used to display messages only if it is NOT a errormessage. Or display errors and warnings in different places.

## Request parameters

These are parameters that are specified in **GET** requests and **POST** requests where Content-type is one of `application/x-www-form-urlencoded` or `multipart/form-data`.

pkit_done

The page to return to after the user has finished logging in or creating a new account.

pkit_lang

Sets the user's preferred language, using a ISO 639 identifier.

pkit_login_page

This parameter is used to specify the page that user attempted to login from. If the login fails, this page is redisplayed.

pkit_login

If this is set to true, then an attempt to log in is made.

pkit_logout

If this is set to true, logs user out.

pkit_remember

If set to true upon login, will save user's cookie so that they are still logged in next time they restart their browser.

pkit_view

Used to implement multiple views/co-branding. For example, if set to print, will search for templates in the `View/print` directory before using templates in the `View/Default` directory, and sets the `<PKIT_VIEW NAME="print">` tag in the view to true.

---