Preview

Preview is a lightweight template engine class that extends the View (view) class.

Namespace: `\`
File location: `lib/base.php`

Render a Preview Template

The Preview Template class offers a mechanism for extremely lightweight templating that just calls some php expressions which are wrapped into the `{~ ~}` control chars. This is useful for any kind of template style, like XML/HTML, haml (http://haml.info/), or anything else.

Lets have a look at a simple example:

```
<div>
    <h1>My favorite books</h1>
    <ul>
        {~ foreach (@books as @id=>@book): ~}
        <li>{~ if (@book.starred): ~}<i class="icon-star"></i>{~ else: ~}<i class="ic
on-normal">{~ endif ~}
            <a href="{{ @BASE.@book[slug] }}">{{ @book.title }}</a>
        </li>
        {~ endforeach ~}
    </ul>
</div>
```

Methods

build

Assemble markup

```
string build ( string $node )
```

This method builds the php code out of the given markup, that is used later in the pre-rendered template. To know about this method becomes useful when you're creating new template extensions or custom tag handlers.

In example it converts this piece of token:

```
{{ @book.title | esc }}
```

into:

```
<?php echo $this->esc($book['title']); ?>
```

filter

Register token filter

```
mixed filter ( [ string $key = NULL [, string $func = NULL ] ] )
```

You can use this method to add your own template token filter like `{{ @content | myfilter }}`. For instance:

```
\Preview::instance()->filter('badwords','\Helper::instance()->badwords');


class Helper extends \Prefab {
    function badwords($val) {
        $bad_words = array("badword","jerk","damn");
        $replacement_words = array("@#$@#", "j&*%", "da*@");
        return str_ireplace($bad_words, $replacement_words, $val);
    }
}
```

Now you can use `{{ @user_comment | badwords }}` to filter the `user_comment` var for badwords. It's also possible to combine multiple filter: `{{ @user_comment | badwords, raw }}`.

When the function is called without any parameter, it just returns an array of all registered filter names. When the function is called with a `$key` but without a `$func` parameter, it returns the registered function string.

render

Render and return a template given by its filename

```
string render ( string $file [, string $mime = 'text/html' [, array $hive = NULL [, i
nt $ttl = 0 ]]] )
```

The `$file` argument expects a file path that is within any defined directories by F3's UI (quick-reference#ui) system variables. Remember, for your convenience, UI (quick-reference#ui) can be multiple paths.

Actually, the render method first try to load your specified template from the temporary folder TEMP (quick-reference#temp) acting as a file cache for compiled templates. The specified template file is only load if needed: F3 is smart enough to detect if a newer version of the template file has been saved and then it builds a php based view file and only then renders the php view again.

Once rendered, F3 escape the content according to the ESCAPE (quick-reference#escape) system variable, and sets the appropriate `Content-Type:` value of the HTTP header. If you'd like to get the response returned as JSON, XML or as E-Mail content, just change the `$mime` type parameter accordingly to your needs. (While the charset of the content is defined by the ENCODING (quick-reference#encoding) system variable)

The `$hive` argument allows you to use an explicit array of variables for that template. By default the rendered template has access to the whole F3 hive with all it's variables.

The `$ttl` argument specifies, in seconds, the Time To Live in the F3 cache for the php based compiled view file. When a `$ttl` is given, F3 will store the compiled view in the cache. On the next request, if the time specified by `$ttl` has passed, the compiled view will be rebuild from the template and stored again in the cache for another cycle.

Examples:

```
echo $tmpl->render('layout.html'); // assumes layout.html is in the UI folder


$flow = $tmpl->render('widgets/tweeter-feeds.html', 'application/json', NULL, 300 );
// cache for 5 minutes
```

resolve

Render template string

```
string resolve ( string $str [, array $hive = NULL ] )
```

Example:

```
$tmpl = \Preview::instance();
$f3->set('title','<b>F3 rocks</b>');
$content = '<p>{{ @title | esc }}</p>';
echo $tmpl->resolve($content);
// return: <p>&amp;lt;b&amp;gt;F3 rocks&lt;/b&amp;gt;</p>
```

token

Convert token to variable

```
string token ( string $str )
```

Example:

```
echo $template->token ('My {{@color}} car looks nice');

// returns the string  'My $color car looks nice'
```