---+ Package `TWiki::Templates`

Support for the TWiki template language.

The following tokens are supported by this language:

%TMPL:P%	Instantiates a previously defined template
%TMPL:DEF%	Opens a template definition
%TMPL:END%	Closes a template definition
%TMPL:INCLUDE%	Includes another file of templates

Note; the template cache does not get reset during initialisation, so the haveTemplate test will return true if a template was loaded during a previous run when used with mod_perl or speedycgi. Frustrating for the template author, but they just have to switch off the accelerators during development.

This is to all intents and purposes a singleton object. It could easily be coverted into a true singleton (template manager).

ClassMethod new `($session)`

Constructor. Creates a new template database object.

- $session - session (TWiki) object

ObjectMethod finish `()`

Break circular references.

ObjectMethod haveTemplate `($name) -> $boolean`

Return true if the template exists and is loaded into the cache

ObjectMethod expandTemplate `($params) -> $string`

Expand the template specified in the parameter string using `tmplP`.

Examples:

```
$tmpls->expandTemplate('"blah");
$tmpls->expandTemplate('context="view" then="sigh" else="humph"');



---++ ObjectMethod *tmplP* <tt>($attrs) -> $string</tt>

Return value expanded text of the template, as found from looking
in the register of template definitions. The attrs can contain a template
name in _DEFAULT, and / or =context=, =then= and =else= values.

Recursively expands any contained TMPL:P tags.

Note that it would be trivial to add template parameters to this,
simply by iterating over the other parameters (other than _DEFAULT, context,
then and else) and doing a s/// in the template for that parameter value. This
would add considerably to the power of templates. There is already code
to do this in the MacrosPlugin.
```

```
---++ ObjectMethod *readTemplate* <tt>($name,$skins,$web) -> $text</tt>

Return value: expanded template text

Reads a template, constructing a candidate name for the template thus
   0 looks for file =$name.$skin.tmpl= (for each skin)
      0 in =templates/$web=
      0 in =templates=, look for
   0 looks for file =$name.tmpl=
      0 in =templates/$web=
      0 in =templates=, look for
   0 if a template is not found, tries in this order
      0 parse =$name= into a web name (default to $web) and a topic name and looks for this topic
      0 looks for topic =${skin}Skin${name}Template=
         0 in $web (for each skin)
         0 in =TWiki::cfg{SystemWebName}= (for each skin)
      0 looks for topic =${name}Template=
         0 in $web (for each skin)
         0 in =TWiki::cfg{SystemWebName}= (for each skin)
In the event that the read fails (template not found, access permissions fail)
returns the empty string ''.

=$skin=, =$web= and =$name= are forced to an upper-case first character
when composing user topic names.

If template text is found, extracts include statements and fully expands them.
Also extracts template definitions and adds them to the
list of loaded templates, overwriting any previous definition.


</div><!-- /patternTopic-->

%META{"form"}%
%META{"attachments"}%</div><!-- /patternContent-->
---
%MAKETEXT{"This topic:"}% <nop>%WEB%%META{"parent" prefix="<span class='twikiSeparator'> &gt
%MAKETEXT{"Topic revision:"}% %REVINFO{format="r$rev - $date - <nop>$wikiname"}%
</div><!-- /patternMainContents-->
</div><!-- /patternMain-->
</div><!-- /patternFloatWrap-->
<div class="clear"> </div>
</div><!-- /patternOuter--><div id="patternBottomBar"><div id="patternBottomBarContents"><div id=
</div><!-- /patternPage-->
</div><!-- /patternPageShadow-->
</div><!-- /patternScreen-->
</body></html>
```