Qweb

QWeb is the primary [templating](http://en.wikipedia.org/wiki/Template_processor) engine used by Odoo[2](https://www.odoo.com/documentation/10.0/reference/qweb.html#othertemplates). It is an XML templating engine[1](https://www.odoo.com/documentation/10.0/reference/qweb.html#genshif) and used mostly to generate [HTML](http://en.wikipedia.org/wiki/HTML) fragments and pages.

Template directives are specified as XML attributes prefixed with t-, for instance t-if for [conditionals](https://www.odoo.com/documentation/10.0/reference/qweb.html#reference-qweb-conditionals), with elements and other attributes being rendered directly.

To avoid element rendering, a placeholder element <t> is also available, which executes its directive but doesn't generate any output in and of itself:

<t t-if="condition">

<p>Test</p>

</t>

will result in:

<p>Test</p>

if condition is true, but:

<div t-if="condition">

<p>Test</p>

</div>

will result in:

<div>

<p>Test</p>

</div>

**data output**

QWeb has a primary output directive which automatically HTML-escape its content limiting [XSS](http://en.wikipedia.org/wiki/Cross-site_scripting) risks when displaying user-provided content: esc.

esc takes an expression, evaluates it and prints the content:

<p><t t-esc="value"/></p>

rendered with the value value set to 42 yields:

<p>42</p>

There is one other output directive raw which behaves the same as respectively esc but *does not HTML-escape its output*. It can be useful to display separately constructed markup (e.g. from functions) or already sanitized user-provided markup.

**conditionals**

QWeb has a conditional directive if, which evaluates an expression given as attribute value:

<div>

<t t-if="condition">

<p>ok</p>

</t>

</div>

The element is rendered if the condition is true:

<div>

<p>ok</p>

</div>

but if the condition is false it is removed from the result:

<div>

</div>

The conditional rendering applies to the bearer of the directive, which does not have to be <t>:

<div>

<p t-if="condition">ok</p>

</div>

will give the same results as the previous example.

Extra conditional branching directives t-elif and t-else are also available:

<div>

<p t-if="user.birthday == today()">Happy bithday!</p>

<p t-elif="user.login == 'root'">Welcome master!</p>

<p t-else="">Welcome!</p>

</div>

**loops**

QWeb has an iteration directive foreach which take an expression returning the collection to iterate on, and a second parameter t-as providing the name to use for the "current item" of the iteration:

<t t-foreach="[1, 2, 3]" t-as="i">

<p><t t-esc="i"/></p>

</t>

will be rendered as:

<p>1</p>

<p>2</p>

<p>3</p>

Like conditions, foreach applies to the element bearing the directive's attribute, and

<p t-foreach="[1, 2, 3]" t-as="i">

<t t-esc="i"/>

</p>

is equivalent to the previous example.

foreach can iterate on an array (the current item will be the current value), a mapping (the current item will be the current key) or an integer (equivalent to iterating on an array between 0 inclusive and the provided integer exclusive).

In addition to the name passed via t-as, foreach provides a few other variables for various data points:

Warning

$as will be replaced by the name passed to t-as

*$as*\_all

the object being iterated over

*$as*\_value

the current iteration value, identical to $as for lists and integers, but for mappings it provides the value (where $as provides the key)

*$as*\_index

the current iteration index (the first item of the iteration has index 0)

*$as*\_size

the size of the collection if it is available

*$as*\_first

whether the current item is the first of the iteration (equivalent to *$as*\_index == 0)

*$as*\_last

whether the current item is the last of the iteration (equivalent to *$as*\_index + 1 == *$as*\_size), requires the iteratee's size be available

*$as*\_parity

either "even" or "odd", the parity of the current iteration round

*$as*\_even

a boolean flag indicating that the current iteration round is on an even index

*$as*\_odd

a boolean flag indicating that the current iteration round is on an odd index

These extra variables provided and all new variables created into the foreach are only available in the scope of the``foreach``. If the variable exists outside the context of the foreach, the value is copied at the end of the foreach into the global context.

<t t-set="existing\_variable" t-value="False"/>

<!-- existing\_variable now False -->

<p t-foreach="[1, 2, 3]" t-as="i">

<t t-set="existing\_variable" t-value="True"/>

<t t-set="new\_variable" t-value="True"/>

<!-- existing\_variable and new\_variable now True -->

</p>

<!-- existing\_variable always True -->

<!-- new\_variable undefined -->

**attributes**

QWeb can compute attributes on-the-fly and set the result of the computation on the output node. This is done via the t-att (attribute) directive which exists in 3 different forms:

t-att-*$name*

an attribute called $name is created, the attribute value is evaluated and the result is set as the attribute's value:

<div t-att-a="42"/>

will be rendered as:

<div a="42"></div>

t-attf-*$name*

same as previous, but the parameter is a [format string](https://www.odoo.com/documentation/10.0/glossary.html#term-format-string) instead of just an expression, often useful to mix literal and non-literal string (e.g. classes):

<t t-foreach="[1, 2, 3]" t-as="item">

<li t-attf-class="row {{ item\_parity }}"><t t-esc="item"/></li>

</t>

will be rendered as:

<li class="row even">1</li>

<li class="row odd">2</li>

<li class="row even">3</li>

t-att=mapping

if the parameter is a mapping, each (key, value) pair generates a new attribute and its value:

<div t-att="{'a': 1, 'b': 2}"/>

will be rendered as:

<div a="1" b="2"></div>

t-att=pair

if the parameter is a pair (tuple or array of 2 element), the first item of the pair is the name of the attribute and the second item is the value:

<div t-att="['a', 'b']"/>

will be rendered as:

<div a="b"></div>

**setting variables**

QWeb allows creating variables from within the template, to memoize a computation (to use it multiple times), give a piece of data a clearer name, ...

This is done via the set directive, which takes the name of the variable to create. The value to set can be provided in two ways:

* a t-value attribute containing an expression, and the result of its evaluation will be set:
* <t t-set="foo" t-value="2 + 1"/>
* <t t-esc="foo"/>

will print 3

* if there is no t-value attribute, the node's body is rendered and set as the variable's value:
* <t t-set="foo">
* <li>ok</li>
* </t>
* <t t-esc="foo"/>

will generate &lt;li&gt;ok&lt;/li&gt; (the content is escaped as we used the esc directive)

Note

using the result of this operation is a significant use-case for the raw directive.

**calling sub-templates**

QWeb templates can be used for top-level rendering, but they can also be used from within another template (to avoid duplication or give names to parts of templates) using the t-call directive:

<t t-call="other-template"/>

This calls the named template with the execution context of the parent, if other\_template is defined as:

<p><t t-value="var"/></p>

the call above will be rendered as <p/> (no content), but:

<t t-set="var" t-value="1"/>

<t t-call="other-template"/>

will be rendered as <p>1</p>.

However this has the problem of being visible from outside the t-call. Alternatively, content set in the body of the call directive will be evaluated *before* calling the sub-template, and can alter a local context:

<t t-call="other-template">

<t t-set="var" t-value="1"/>

</t>

<!-- "var" does not exist here -->

The body of the call directive can be arbitrarily complex (not just set directives), and its rendered form will be available within the called template as a magical 0 variable:

<div>

This template was called with content:

<t t-raw="0"/>

</div>

being called thus:

<t t-call="other-template">

<em>content</em>

</t>

will result in:

<div>

This template was called with content:

<em>content</em>

</div>

**Python**

**Exclusive directives**

**asset bundles**

**"smart records" fields formatting**

The t-field directive can only be used when performing field access (a.b) on a "smart" record (result of the browse method). It is able to automatically format based on field type, and is integrated in the website's rich text edition.

t-options can be used to customize fields, the most common option is widget, other options are field- or widget-dependent.

**debugging**

t-debug

invokes a debugger using PDB's set\_trace API. The parameter should be the name of a module, on which a set\_trace method is called:

<t t-debug="pdb"/>

is equivalent to importlib.import\_module("pdb").set\_trace()

**Helpers**

**Request-based**

Most Python-side uses of QWeb are in controllers (and during HTTP requests), in which case templates stored in the database (as [views](https://www.odoo.com/documentation/10.0/reference/views.html#reference-views-qweb)) can be trivially rendered by calling [odoo.http.HttpRequest.render()](https://www.odoo.com/documentation/10.0/reference/http.html#odoo.http.HttpRequest.render):

response = http.request.render('my-template', {

'context\_value': 42

})

This automatically creates a [Response](https://www.odoo.com/documentation/10.0/reference/http.html#odoo.http.Response) object which can be returned from the controller (or further customized to suit).

**View-based**

At a deeper level than the previous helper is the render method on ir.ui.view:

**render(*cr, uid, id[, values][, engine='ir.qweb][, context]*)**

Renders a QWeb view/template by database id or [external id](https://www.odoo.com/documentation/10.0/glossary.html#term-external-id). Templates are automatically loaded from ir.ui.view records.

Sets up a number of default values in the rendering context:

request

the current [WebRequest](https://www.odoo.com/documentation/10.0/reference/http.html#odoo.http.WebRequest) object, if any

debug

whether the current request (if any) is in debug mode

[quote\_plus](http://werkzeug.pocoo.org/docs/urls/#werkzeug.urls.url_quote_plus)

url-encoding utility function

[json](https://docs.python.org/2/library/json.html#module-json)

the corresponding standard library module

[time](https://docs.python.org/2/library/time.html#module-time)

the corresponding standard library module

[datetime](https://docs.python.org/2/library/datetime.html#module-datetime)

the corresponding standard library module

[relativedelta](https://labix.org/python-dateutil#head-ba5ffd4df8111d1b83fc194b97ebecf837add454)

see module

keep\_query

the keep\_query helper function

Parameters

* **values** -- context values to pass to QWeb for rendering
* **engine** ([str](https://docs.python.org/2/library/functions.html#str)) -- name of the Odoo model to use for rendering, can be used to expand or customize QWeb locally (by creating a "new" qweb based on ir.qweb with alterations)

**Javascript**

**Exclusive directives**

**defining templates**

The t-name directive can only be placed at the top-level of a template file (direct children to the document root):

<templates>

<t t-name="template-name">

<!-- template code -->

</t>

</templates>

It takes no other parameter, but can be used with a <t> element or any other. With a <t> element, the <t> should have a single child.

The template name is an arbitrary string, although when multiple templates are related (e.g. called sub-templates) it is customary to use dot-separated names to indicate hierarchical relationships.

**template inheritance**

Template inheritance is used to alter existing templates in-place, e.g. to add information to templates created by an other modules.

Template inheritance is performed via the t-extend directive which takes the name of the template to alter as parameter.

The alteration is then performed with any number of t-jquery sub-directives:

<t t-extend="base.template">

<t t-jquery="ul" t-operation="append">

<li>new element</li>

</t>

</t>

The t-jquery directives takes a [CSS selector](http://api.jquery.com/category/selectors/). This selector is used on the extended template to select *context nodes* to which the specified t-operation is applied:

append

the node's body is appended at the end of the context node (after the context node's last child)

prepend

the node's body is prepended to the context node (inserted before the context node's first child)

before

the node's body is inserted right before the context node

after

the node's body is inserted right after the context node

inner

the node's body replaces the context node's children

replace

the node's body is used to replace the context node itself

No operation

if no t-operation is specified, the template body is interpreted as javascript code and executed with the context node as this

Warning

while much more powerful than other operations, this mode is also much harder to debug and maintain, it is recommended to avoid it

**debugging**

The javascript QWeb implementation provides a few debugging hooks:

t-log

takes an expression parameter, evaluates the expression during rendering and logs its result with console.log:

<t t-set="foo" t-value="42"/>

<t t-log="foo"/>

will print 42 to the console

t-debug

triggers a debugger breakpoint during template rendering:

<t t-if="a\_test">

<t t-debug="">

</t>

will stop execution if debugging is active (exact condition depend on the browser and its development tools)

t-js

the node's body is javascript code executed during template rendering. Takes a context parameter, which is the name under which the rendering context will be available in the t-js's body:

<t t-set="foo" t-value="42"/>

<t t-js="ctx">

console.log("Foo is", ctx.foo);

</t>

**Helpers**

**core.qweb**

(core is the web.core module) An instance of [QWeb2.Engine()](https://www.odoo.com/documentation/10.0/reference/qweb.html#QWeb2.Engine) with all module-defined template files loaded, and references to standard helper objects \_ (underscore), \_t (translation function) and [JSON](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON).

[core.qweb.render](https://www.odoo.com/documentation/10.0/reference/qweb.html#QWeb2.Engine.render) can be used to easily render basic module templates

**API**

***class* QWeb2.Engine()**

The QWeb "renderer", handles most of QWeb's logic (loading, parsing, compiling and rendering templates).

OpenERP Web instantiates one for the user in the core module, and exports it to core.qweb. It also loads all the template files of the various modules into that QWeb instance.

A [QWeb2.Engine()](https://www.odoo.com/documentation/10.0/reference/qweb.html#QWeb2.Engine) also serves as a "template namespace".

**QWeb2.Engine.render(*template*[, *context*])**

Renders a previously loaded template to a String, using context (if provided) to find the variables accessed during template rendering (e.g. strings to display).

Arguments

* **template** (String) -- the name of the template to render
* **context** (Object) -- the basic namespace to use for template rendering

Returns

String

The engine exposes an other method which may be useful in some cases (e.g. if you need a separate template namespace with, in OpenERP Web, Kanban views get their own [QWeb2.Engine()](https://www.odoo.com/documentation/10.0/reference/qweb.html#QWeb2.Engine) instance so their templates don't collide with more general "module" templates):

**QWeb2.Engine.add\_template(*templates*)**

Loads a template file (a collection of templates) in the QWeb instance. The templates can be specified as:

An XML string

QWeb will attempt to parse it to an XML document then load it.

A URL

QWeb will attempt to download the URL content, then load the resulting XML string.

A Document or Node

QWeb will traverse the first level of the document (the child nodes of the provided root) and load any named template or template override.

A [QWeb2.Engine()](https://www.odoo.com/documentation/10.0/reference/qweb.html#QWeb2.Engine) also exposes various attributes for behavior customization:

**QWeb2.Engine.prefix**

Prefix used to recognize directives during parsing. A string. By default, t.

**QWeb2.Engine.debug**

Boolean flag putting the engine in "debug mode". Normally, QWeb intercepts any error raised during template execution. In debug mode, it leaves all exceptions go through without intercepting them.

**QWeb2.Engine.jQuery**

The jQuery instance used during template inheritance processing. Defaults to window.jQuery.

**QWeb2.Engine.preprocess\_node**

A Function. If present, called before compiling each DOM node to template code. In OpenERP Web, this is used to automatically translate text content and some attributes in templates. Defaults to null.

[[1]](https://www.odoo.com/documentation/10.0/reference/qweb.html#id2) it is similar in that to [Genshi](http://genshi.edgewall.org), although it does not use (and has no support for) [XML namespaces](http://en.wikipedia.org/wiki/XML_namespace)

[[2]](https://www.odoo.com/documentation/10.0/reference/qweb.html#id1) although it uses a few others, either for historical reasons or because they remain better fits for the use case. Odoo 9.0 still depends on [Jinja](http://jinja.pocoo.org) and [Mako](http://www.makotemplates.org).