ChiliProject - Feature # 665: Provide a way to query issues qith Liquid

Status:	Open	**Priority:**	Normal
Author:	Holger Just	**Category:**	Text formatting
Created:	2011-10-17	**Assignee:**	Holger Just
Updated:	2012-07-11	**Due date:**	

Remote issue URL:

Affected version:

Description:

Once we have full liquid support (see #604) it is desirable to have access to about all objects from within the language with issues being the most important object type here.

I started working on an extension which utilizes a simple "treetop":http://treetop.rubyforge.org/ grammar to parse a DSL for filter specification and to subsequently create the issues and make them available to the user.

The syntax currently looks like this:

<pre>{%raw%}
{% query my_query %}
status o
author_id = 1
{% endquery %}

|_. ID |_. Subject |_. Status |
{% for issue in my_query.issues %}
| {{issue.id}} | {{issue.subject}} | {{issue.status.name}} |
{% endfor %}

You have {{my_query.count}} open issues.
{%endraw%}</pre>

This generates the following example output:

_. ID	_. Subject	_. Status
1	This is an issue	New
2	A second issue	New

You have 2 open issues.

What is missing yet is full support for all attributes and more user-friendly input as it currently relies on the filter representation inside the query class which sometimes is not obvious.

Also missing are facilities for sorting and a performant way of slicing (i.e. without first instantiating all previous issues).

History

2011-10-17 05:48 pm - Holger Just

You can find the current code at https://github.com/meineerde/chiliproject/tree/issues%2Funstable%2F665-liquid-issue-queries

It builds on top of the current @issues/unstable/604-liquid@ branch which contains the code for #604.

2012-07-11 02:31 pm - Holger Just

- Description changed from Once we have full liquid support (see #604) it is desirable to have access to about all objects from within the language with issues being the most important object type here.

I started working on an extension which utilizes a simple "treetop":http://treetop.rubyforge.org/ grammar to parse a DSL for filter specification and to subsequently create the issues and make them available to the user.

The syntax currently looks like this:

<pre>
{% query my_query %}
status o
author_id = 1
{% endquery %}

|_. ID |_. Subject |_. Status |
{% for issue in my_query.issues %}
| {{issue.id}} | {{issue.subject}} | {{issue.status.name}} |
{% endfor %}

You have {{my_query.count}} open issues.
</pre>

This generates the following example output:

|_. ID |_. Subject |_. Status |
| 1 | This is an issue | New |
| 2 | A second issue | New |

You have 2 open issues.

What is missing yet is full support for all attributes and more user-friendly input as it currently relies on the filter representation inside the query class which sometimes is not obvious.

Also missing are facilities for sorting and a performant way of slicing (i.e. without first instantiating all previous issues). to Once we have full liquid support (see #604) it is desirable to have access to about all objects from within the language with issues being the most important object type here.

I started working on an extension which utilizes a simple "treetop":http://treetop.rubyforge.org/ grammar to parse a DSL for filter specification and to subsequently create the issues and make them available to the user.

The syntax currently looks like this:

<pre>{%raw%}
{% query my_query %}
status o
author_id = 1
{% endquery %}

```
|_. ID |_. Subject |_. Status |
{% for issue in my_query.issues %}
| {{issue.id}} | {{issue.subject}} | {{issue.status.name}} |
{% endfor %}

You have {{my_query.count}} open issues.
{%endraw%}</pre>
```

This generates the following example output:

```
|_. ID |_. Subject |_. Status |
| 1 | This is an issue | New |
| 2 | A second issue | New |

You have 2 open issues.
```

What is missing yet is full support for all attributes and more user-friendly input as it currently relies on the filter representation inside the query class which sometimes is not obvious.

Also missing are facilities for sorting and a performant way of slicing (i.e. without first instantiating all previous issues).