Tracker Query

This was an experiment, but was later removed and reports are much better done via PluginList

Also see Tracker Reports in Tiki8 (tiki-tracker_reports.php) and Report (in Tiki9)

A tracker query, or the tracker query lib (first in lib(trackers/trackerquerylib.php then in lib/core/Tracker/Query.php [Tiki v.8]; lib(trackers/trackerlib.php [v.9]) is designed to facilitate and focus on the "getting" of tracker data, as of Tiki 9, migration is more for an api of getting or setting data in trackers.

Query

As of tiki 9, the following chainable methods are available:

- `start($date)` - accepts unix start date
- `end($date)` - accepts unix end date
- `itemIt($itemId)` - defaults is 0, if set overrides to a single tracker item
- `filter(array())` - reusable, allows you to narrow or 'filter' results from a tracker, sort of like if field value = value (if used more than once acts like logical AND)
  - example

  Tracker_Query::tracker(19)
  ->filter(array('field'=>47, 'value'=>'admin'))
  ->filter(array('field'=>48, 'value'=>'success'))
  ->query();
  - example

  Tracker_Query::tracker('My Tracker')
  ->byName()
  ->filter(array('field'=>'User', 'value'=>'admin'))
  ->filter(array('field'=>'Logins', 'value'=>'success'))
  ->query();

- `fields(array())` - an array of fields to return, default is all fields
  - example

  Tracker_Query::tracker(19)
  ->fields(array(47,48,49))
  ->query();

- `status($status)` - string, default 'opc'
  - acceptable - any combination of the letters 'o', 'p', or/and 'c'
- `limit($limit)` - int, limits mysql results from trackers
- `offset($offset)` - int, offset of mysql results from trackers
- `byName($byName)` - bool, switches from id to name of all names, tracker & fields, useful for building apps and hard-coding identifiers
- `lastModif($lastModif)` - bool, sets the default query order to last modified
- `created($created)` - bool, sets the default query order to when created
- `excludeDetails($excludeDetails)` - bool, removes a few details from the result set
• **desc**($desc) - bool, switched the result set sort order to descending
• **render**($render) - bool, if set to false, stops fields from being rendered, can greatly speed up interaction with trackers, but the result is without rendered

• Note chainable methods are to be used with either manually or with static method ‘tracker’ and with the output methods:
  • **getOne()** - returns the first tracker item from the result set
  • **getItemId()** - runs getOne, and returns the itemId of that item
  • **query()** - outputs all found tracker items
  • **queryInputs()** - outputs all found trackers as inputs
  • **queryInput()** - when itemId is set, outputs that item, otherwise it will be a blank item that can be used in creating a new tracker item

For querying, tracker query uses a new method of concatenation of items directly from the mysql tiki database using mysql’s group_concat to return data. This lib then outputs trackers in a very clean manner. Example:

```php
Array
{
   [367] => Array //item repeats, key = itemId
       (       
           [19] => 369 //item field values, key = fieldId
           [20] => 366
           [status5] => c
           [trackerId] => 5
           [itemId] => 367
           [162] => Array //items list associated to fieldId 162
               (               
                   [0] => 176 // itemId
                   [1] => Event Name // static name of an itemId
               )               
       )
}
```

Such array can be return by a call like this

```php
//v9 (to be)
$result = Tracker_Query::tracker("MyTracker")
    ->byName()
    ->query();

//or v8
$trklib = Tikilib::lib('trk');
$result = TrackerQueryLib::tracker("MyTracker")
    ->byName()
    ->query();
```

The base array is a group of tracker items. The children arrays of those items have keys that are the field
from where they are reported from. 162 is an example of a items list, they are grouped in an array. In short what tracker query lib does is turn trackers from a huge list, to a grid. This grid is how we picture trackers and how they are designed. After we obtain data from function tracker_query, we can then add headers, sort the data, do everything that you would could normally do with data, and very easy and fast... for trackers ;). We can also join trackers to this tracker by using "join_trackers". This function looks for the item id of the item that you are joining to within a specified field that it was declared to relate from.

Some call details

Each call must begin with TrackerQueryLib::tracker("MyTracker") and finish with query()
There are a lot of functions that can be use between these 2 parts to filters the result. The order of these functions are not important.
Here are some examples

```
//v9 (to be)
$result = Tracker_Query::tracker("MyTracker")
  ->byName()
  ->getOne()
  ->query();

//v8
$result = TrackerQueryLib::tracker("MyTracker")
  ->byName()
  ->getOne()
  ->query();

$result = TrackerQueryLib::tracker(myTrackerId)
  ->search(array("toto", "todo"))->fields(array(23, 23)) //filter the fieldId 23 on toto or todo
  ->start(strtotime("1/1/2011")) // items last modified after this date
  ->end(strtotime("1/2/2011")) // items last modified before this date
  ->limit(20) // returns only the first 20 elements
  ->offset(10) // beginning after the 10 first items
  ->status('c') // only closed items
  ->sort(32_asc) // sort on fieldId 32 ... to be checked
  ->query();
```

Query Input(s)

Tracker queryInput and queryInputs were added in Tiki 9 to make it easy to obtain the rendered (smarty + javascript + naming) html input of a tracker. What does it allow you to do? Build an application on top of trackers!

Query Input Usage

Query input acts just like a standard tracker query in that the query can be setup the same. The output though is somewhat different:

```
//Get input for 1 item, which can easily be json_encoded and returned on an ajax request
Tracker_Query::tracker('Wiki Attributes')
  ->byName()
  ->itemId((int)$_REQUEST['itemId'])
```
->queryInput();

//Output:
Array
{
    [Page] => <input type="text" id="page_selector_276" name="ins_276" value="WebServerA" />
    [Attribute] => <input type="text" id="ins_277" name="ins_277" value="" />
    [Value] => <input type="hidden" name="mode_wysiwyg" value="" />
    <input type="hidden" name="mode_normal" value="" />
    <div class='edit-zone'>
        <textarea  id="area_278" name="ins_278" cols="50" rows="15" onkeyup="" class="wikiedit" data-codemirror="" data-syntax="" style="width:99%">Does this have to do with computers?</textarea>
    </div>
    <input type="hidden" name="wysiwyg" value="n" />
    [Type] => <input type="text" id="ins_279" name="ins_279" value="Question" />
}

Or perhaps you want to get several items, notice the plural "queryInputs":

Tracker_Query::tracker('Wiki Attributes')
    ->byName()
    ->queryInputs();

Example of QueryInput

1. On your webserver, in your tiki root directory, create a .php file (example, queryInputTest.php), and in that file enter something like:

```php
<?php
//queryInputTest.php
require_once('tiki-setup.php');
require_once('lib/core/Tracker/Query.php');
require_once('lib/trackers/trackerlib.php');
$user = "myusername";
$trackerName = "GuestList";
$result = Tracker_Query::tracker($trackerName)
    ->byName()
    ->itemId((int)$_REQUEST['itemId'])
    ->queryInput();  //displays a single blank record of editable text boxes
print_r($result);
?>
```

Open a browser to your your tiki site:

http://domain.tld/t/queryInputTest.php?itemId=50

and you should see the raw Array Output of your tracker as shown above.
Questions re QueryInput(s)

Thanks for this, Robert. This could be really huge. I (petjal) realized today, that my complex Tiki Applications should never have been created using User tools like TRACKER* plugins. This API might help us alot.

1. How does one put this into a wiki page? With a Save (or "Save and Next Page") button? (from petjal)
2. In Tiki trunk (possibly in 9.1 when it is released) you can do this:

```php
<?php
$inputs = json_encode(
    Tracker_Query::tracker('Tracker')
    ->byName()
    ->itemId((int)$itemId) //if not set, 0 = new
    ->inputDefaults(array//since we use the method byName above, we can identify fields by their name, else we'd use their id. Also, inputDefaults makes new items have a default value
        'Page' => $page,
        'Type' => 'Question'
    ))
    ->queryInput()
);
TikiLib::lib('header')->add_jq_onready("var inputs = $inputs;
var form = $('<form />'
    .submit(function() {
        $.post('tiki-ajax_services.php?controller=tracker&action=insert_item&' + from.serialize() + '
            &itemId=' + itemId + '&trackerId=' + trackerId, function() {
            document.location = document.location + '';
        });
    });
});
$.each(inputs, function(i) {
    form.append(inputs[i]);
});
";
```