

FREEFORM
s o l u t i o n s



CiviReport 101

-

**A gentle introduction to creating,
modifying and troubleshooting CiviCRM
reports**

About Freeform Solutions

- A **Not-for-profit** social enterprise
- Mission - provide IT consulting for not-for-profits
- We've worked with NFP for over 10 years, CiviCRM for over 6 years
- We use Open Source (Drupal, CiviCRM, Formulize)

The presenter:

- Lola Slade: Developer and Client lead
- 3 years with Freeform
- 2 previous years as a report developer



About this presentation

- This talk is oriented to administrators and developers
 - Who have modified Civireport parameters and saved new reports
 - Who have not modified CiviCRM report files via the PHP templates or are still struggling with it
- Inspiration for this talk
 - Reporting is important!
 - Civireport is hard or at least kind of crusty
- There is a supporting extension with example reports:
<https://github.com/freeform/ca.freeform.civireport101>

Quick Terminology Overview

CiviReport:

- Is a PHP class that can be extended by new reports (**CRM_Report_Form**)
- Provides a default Smarty template for the form controls

A report template:

- Is a PHP class that overrides CRM_Report_Form
- Can include a custom Smarty template or use the default one
- Yes: This is 2 different uses of the word template

A report instance:

- Is a DB record saving report preferences & menu entry

Administration Overview

Registering a report template:

- Is mapping the class name to a default report name, description and webpath
- Use Administer >> CiviReport >> Manage templates
- If you create an extension this is handled for you

Registered Templates

The existing option choices for Report Template group are listed below. You can add, edit or delete them from this screen.

» Register New Report Template

Label	URL	Description	Order	Reserved	Enabled?	Component	
Constituent Report (Summary)	contact/summary	Provides a list of address and telephone information for constituent records in your system.	↓ ↓	No	Yes	Contact	Edit Disable Delete
Constituent Report (Detail)	contact/detail	Provides contact-related information on contributions, memberships, events and activities.	↑ ↓ ↓	No	Yes	Contact	Edit Disable Delete
Activity Report	activity	Provides a list of constituent activity	↑ ↓ ↓	No	Yes	Contact	Edit Disable Delete

Setting up new reports - Manual Method

This is similar to overriding a Smarty template. It consists of:

- **Setting** the Custom PHP and Custom template directories. At Administer >> System Settings >> Directories
- Copying the report class into a subfolder of the Custom PHP folder
- Copying the Smarty template class into a subfolder of the Custom Templates folder
- Register and enable the Report template
- Administer >> CiviReport >> Create New Report from Template

Report Template

New Report Template

Title *
Report title appear in the display screen.


Description *
Report description appear in the display screen.

URL *
Report Url must be like "contribute/summary"

Class *
Report Class must be present before adding the report here, e.g. 'CRM_Report_Form_Contribute_Summary'

Weight *

Component
Specify the Report if it is belongs to any component like "CiviContribute"

Enabled? 

✓ Sa

» Relationship Report <small>Existing Report(s)</small>	Relationship Report
» Database Log Report	Log of contact and activity records created or update
» Activity Report (Summary)	Shows activity statistics by type / date
» DefaultExtSample	DefaultExtSample (ca.freeform.civireport101)
» Example1Empty	Example1Empty (ca.freeform.civireport101)
» Civireport101 Manual Sample	A sample of manual registration

Setting up new reports - Extension

Steps:

- Install and test CiviCRM Buildkit
- Use Buildkit to create a demo site

Or:

- Create a local install demo site manually
- Install **civix**

The following commands:

```
civix generate:module org.myorg.extname  
cd org.myorg.extname  
civix generate:report MyCamelName
```

Troubleshooting reports

- Troubleshoot the SQL first!
- Insert a debugging statement in postProcess to see the SQL: CRM_Core_Error::debug, dpm, print_r etc.
- Oh great, we get this:

```
SELECT {50 columns} FROM {20 tables}  
WHERE {who knows it doesn't fit on the  
screen}
```
- Let's change this to:

```
SELECT {5 columns} FROM {formatted 20  
tables} WHERE {nicely formatted}
```
- Then comment out where clauses and joins one by one
- Possibly add extra WHERE clause to limit results to a couple of test contacts

Troubleshooting example

The bookkeeping report was returning the wrong result for all donations for only one contact.

By following the previous procedure I found that 1) commenting out the group by we were getting duplicate rows for that contact, 2) commenting out the phone table join we got the right result.

The problem turned out to be 2 phones records marked `is_primary`. Probably an import or api bug had caused it.

Basic outline of a report

1. Define columns with **__construct**
2. The user selects columns, filters and other options
3. Like any CiviCRM Form the selections are sent to **postProcess** as **\$params**
4. **postProcess** then calls **select, from, where, groupBy** etc. to generate **\$sql**
5. **\$sql** gets executed to generate **\$rows**
6. **\$rows** can be modified using a few functions including **alterDisplay**
7. **statistics** calculates **\$statistics**

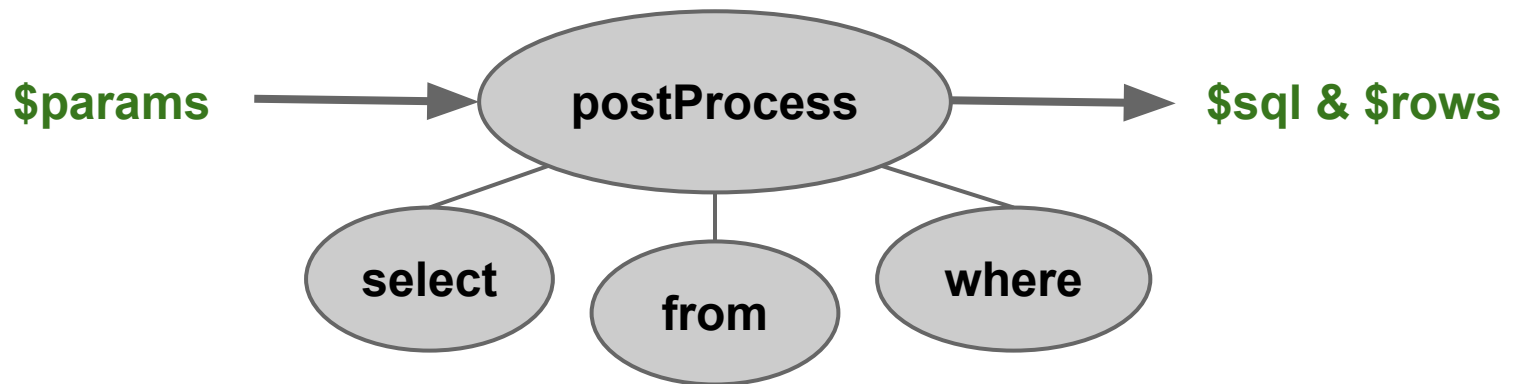
```

1 <?php
2
3 class CRM_Civireport101_Form_Report_Example1Empty extends CRM_Report_Form {
4
5 function __construct() {
6     $this->_columns = array(
7         'table_name' => array(
8             'fields' => array(
9                 'hello' => array(
10                    'title' => ts('Hello'),
11                    'default' => TRUE,
12                ),
13            ),
14        ),
15    );
16    //dpm($this->_columns);
17    parent::__construct();
18 }
19
20 function preProcess() {
21     $this->assign('reportTitle', ts('Empty Report'));
22     parent::preProcess();
23 }
24
25 function select() {
26     $this->_select = 'SELECT "Hello Reporting!" AS hello ';
27     // note2: register columns you want displayed-
28     $this->_columnHeaders = array(
29         'hello' => array( 'title' => 'Hello' ),
30     );
31 }
32
33 function from() { $this->_from = " "; }
34
35 function where() { $this->_where = " "; }
36
37 function groupBy() { $this->_groupBy = " "; }
38
39 function orderBy() { $this->_orderBy = " "; }
40
41 function postProcess() {
42     $this->beginPostProcess();
43     $sql = $this->buildQuery(TRUE);
44     //dpm($sql);
45
46     $rows = array();
47     $this->buildRows($sql, $rows);
48     //dpm($rows);
49
50     $this->formatDisplay($rows);
51     $this->doTemplateAssignment($rows);
52     $this->endPostProcess($rows);
53 }
54
55 function alterDisplay(&$rows) {
56     // TODO: custom code to alter rows
57 }
58 }
59

```

The class creates a SQL statement

CiviCRM does best at rather simple SQL cases that generate a grid of data but then so do most other reporting tools as well



```

2326 | $sql = "{$this->_select} {$this->_from} {$this->_where} {$this->_groupBy} {$this->_having} {$this->_orderBy} {$this->_limit}";
2327 | return $sql;
2328 | }
  
```

Example 1 - Hello Reporting!

- A contrived example to show how much we can tear it down
- Shows some of the services of the parent class
CRM_Report_Form
- Highlights the all important **postProcess** function

__construct: Adding tables and fields

This is where we build the **_columns** array. The first level of keys is usually a table name.

alias - To give the table another name. This is used when we use the same table multiple times in the query

dao, bao - Class name of the DAO or BAO for the table. If both dao and bao are specified the BAO wins. These are used for setting defaults for fields in case you don't fully specify the fields array.

grouping - Section the column selectors are displayed in on the form. Hard to see since the default CSS does not highlight the different groups.

fields, filters, group_bys, order_bys - arrays that we will cover more on the next slide

__construct: Adding tables cont...

name - To specify the table name. This is used when we use the same table multiple times in the query

For name, alias, and dbAlias: If you set dbAlias directly then it takes precedence. Otherwise dbAlias gets computed as alias.name

There is also a separate **_options** array for controls you want on the report form that are not tied to the table.

```
$this->_options =  
    array(  
        'use_advanced_eligibility' => array('title' => ts('Use Advanced Eligibility (Hooks - Memory Intensive)'),  
        'type' => 'checkbox',  
    ),  
);  
parent::__construct();  
}
```

fields: array options

The fields array contains keys for these true/false options:

required - This column cannot be de-selected

default - Checkbox is selected by default

no_repeat - Repeated values in the column are removed
in alterDisplay

no_display - Doesn't appear in the output, but can be
used in the query (or in PHP code that reads \$rows)

filters: array options

The **filters** array contains keys for these options:

title - The title, which should match the field title if also used there

default - Default value

operatorType - Form control to use to enter filter

options - A list of options if the operatorType is T_SELECT or T_MULTISELECT

type - The variable type. Not having this will occasionally cause an error so put it in.

select: Choosing visible columns

Here we have 2 jobs:

- Build the select clause and save in **`$this->_select`**
- Create the **`$this->_columnHeaders`** array which determines which of the fields is visible
- The default code shows required columns + those selected by the user

```
function select() {
    $select = $this->_columnHeaders = array();

    foreach ($this->_columns as $tableName => $table) {
        if (array_key_exists('fields', $table)) {
            foreach ($table['fields'] as $fieldName => $field) {
                if (CRM_Utils_Array::value('required', $field) ||
                    CRM_Utils_Array::value($fieldName, $this->_params['fields'])) {
                    $select[] = "{$field['dbAlias']} as {$tableName}_{$fieldName}";
                    $this->_columnHeaders[ "{$tableName}_{$fieldName}" ]['title'] = $field['title'];
                    $this->_columnHeaders[ "{$tableName}_{$fieldName}" ]['type'] = CRM_Utils_Array::value('type', $field);
                }
            }
        }
    }

    $this->_select = "SELECT " . implode(', ', $select) . " ";
}
```

from: Building up the joins

This builds up the FROM SQL clause and stores it in **\$this->_from**. It is usually straightforward but can have some logic needed when a report needs to operate in 2 fundamentally different ways.

For example, the contributions report needs a very different FROM clause if it should show both soft credits and regular credits.

Example 2 - A Basic report

- A very simple example that pulls in real data
- Highlights the **__construct** and **select** functions

Fields array cont...

name - <text> - To give the field another name. This defaults to the key for this field array. Used when the same field needs to appear twice. EG. id, birth_date as age

alias - <text> - To give the table another name, just for this field. This defaults to alias for the table array, or if not set, the key for the table array. See note below.

dbAlias - <text> - To give the field another name.

statistics - array - To make query compute stats like sum, count, avg on a numeric field. Applies to both "display columns" and "filters".

Using option groups or pseudoconstants

These are lists of values and labels.

They are used for:

- Options for Filters and Group bys
- Replacing values in alterDisplay

The best place to figure out which one to use is to look at other reports that use the same tables or to look at each CiviCRM component folder under civicrm/CRM for a class. EG. CRM_Core_PseudoConstant

If there is no class it might have been removed in recent refactoring. If so use CRM_Core_OptionGroup::values

alterDisplay: Option values, links, and more!

- This function alters the **\$rows** array to change the displayed results.
- Note: Altering rows 1 x 1 may be lower performance than modifying the SQL.
- Often used to add links and display option values.
- It can be used for more advanced row changes.

Example:

Suppose we have a report that shows parents and children. We could use this function to remove child contact details except for name, leaving the parent details depending on the logged in staff role.

Example 3 - A Fancier report

- An improved example that has some formatted columns and multiselect filters
- Highlights the **alterDisplay** function
- Shows an example of **dbAlias** and **_options**

Alternatives to CiviReport

- **Civivisualize** - This is good for viewing data and creating great looking dashboards
 - Does not have export or PDF features
- Forena Reports - Drupal only
- Exports and Report - WP only
- Java reporting or local tools with a connection to the CiviCRM DB
- PHP Reporting tools - Most are quite basic. The best I found was PHPReports

Topics for CiviReport 201

- More Statistics and custom summary tables
- Creating a custom template (like the Event Income report)
- More advanced grouping (like the SYBUNT, LYBUNT reports etc.)
- Misc other stuff: Charts

This has been: CiviReport 101

Questions?

Feedback?

Should we have BOF session for reporting?



Thank you!

- Freeform Solutions <http://freeform.ca> info@freeform.ca
- Lola Slade lola@freeform.ca

Resources:

- The documentation: <http://wiki.civicrm.org/confluence/display/CRMDOC/CiviReport+Reference>
- Allan Shaw's 2012 talk on CiviReport: <https://vimeo.com/groups/civicrm/videos/40102206>
- The samples for this talk: <https://github.com/freeform/ca.freeform.civireport101>
- Eileen McNaughton's extended report extension: <https://civicrm.org/extensions/extended-reports>
- Other report extensions on the extensions directory (about 8 total)
- <https://www.drupal.org/project/forena>
- <https://wordpress.org/plugins/exports-and-reports>
- <http://community.pentaho.com/projects/reporting/>
- <http://jdorn.github.io/php-reports/>