# CiviCRM Developer Training 2015

Parvez Saleh
Deepak Srivastava

# Aims and Objectives

## Day 2

- **Testing** - unit tests and webtests
- **CiviReport Customisations & New Reports**
- **CiviCRM APIs & Hands on DAOs**
- **Debugging** - tips and tricks
- **Advanced features**
  - Settings in the Conf file
  - Tips on Customisations
  - Smarty in message templates
  - checksum authentication
  - cron and scheduled jobs
  - moving CiviCRM gotcha's
- **Form Elements, Quick Form Helpers, Smarty Helpers, CiviCRM Helpers**
- **Contributing to CiviCRM**

# Tests: Unit tests and webtests

me@civisys:/home/me$ cd /var/www/extensions/com.example.myextension
me@civisys:/var/www/extensions/com.example.myextension$ **civix generate:test -h**
Usage: generate:test className

Arguments: className  The full class name (eg "CRM_Myextension_MyTest")

me@civisys:/var/www/extensions/com.example.myextension$ **civix generate:test CRM_Myextension_MyTest**
Write /var/www/extensions/com.example.myextension/tests/phpunit/CRM/Myextension/MyTest.php

# Customising a Report

Create new report extension using Civix (Copy the report you want to customise)

**Example:**

```
civix generate:report-ext --copy CRM_Report_Form_Contribute_Detail uk.co.vedaconsulting.contribution-detailreport CiviContribute
```

**Hint**: If you getting API error, run the below command

```
civix config:set civicrm_api3_conf_path <path to>/sites/default
```

Edit the php file inside the extension to add/remove display columns & filter criteria

**Create report instance:**

1. Administer >> CiviReport >> Create new report from template
2. Click on the extension name
3. Set the display columns & filter criteria
4. Click 'Preview Report'
5. Create report instance under 'Create Report' section

# Add a new API

For example, we need an api to save data in civicrm_training_setting table, and use this api in your extension.

Civix to the rescue!

1. Generate a new API `civix generate:api -h` to get the syntax.

e.g. `civix generate:api Training create`

Action names should be lowercase. The javascript helpers CRM.api() and CRM.api3() force actions to be lowercase. This issues does not present itself in the API Explorer or when the api action is called via PHP, REST, or SMARTY

2. Install/Uninstall the extension
3. Check your new api appears in api explorer
4. Complete the api to save data in external civicrm_training_setting table

# Overrides in Conf file

There are also some settings which can not be set from the administration user interface - but which can be overridden using this same technique. These include the Community Messages URL, and whether or not to allow users to download extensions (refer to Disable automatic installation of extensions for more details). This can be useful if your local development environment is different to your server and therefore you can keep local settings files that are configured correctly. Full details can be found here

## Example: Override Temporary Files directory

To override the Temporary Files directory for a test site, add these lines to *civicrm.settings.php*:

```
// Add this line only once above any settings overrides

global $civicrm_setting;

// Override the Temporary Files directory

$civicrm_setting['Directory Preferences']['customFileUploadDir'] =
'/var/www/testing/sites/default/files/civicrm/upload';

// Turn off user extension install

$civicrm_setting['Extension Preferences']['ext_repo_url'] = false;
```

# Tag your customisations

Once you make a lot of changes it can become difficult for you to find the relevant file and to know if its been customised without checking the entire directory structure. You can help yourself by tagging you own files, and then you'll see

```
<input name="payment_processor" type="hidden" value="18" />
<input name="priceSetId" type="hidden" value="105" />
<input name="frequency_interval" type="hidden" value="1" />
<input id="selectProduct" name="selectProduct" type="hidden" value="" />
<input name="_qf_default" type="hidden" value="Main:upload" />
<input name="MAX_FILE_SIZE" type="hidden" value="10485760" />
</div>


<!-- Customised Civi File - .tpl file invoked: /sites/all/modules/civi_events/templates/CRM/Contribute/Form/Contribution/Main.tpl. -->


<script type="text/javascript">
   cj('.one-off').addClass('initial');
 cj('.monthly-dd').addClass('initial');

 cj(".select25").click(function() {
     cj('.one-off').removeClass('highlight');
     cj('.select25').addClass('highlight');
```

# Smarty in message templates

Users may require message templates to be conditional based on values from civicrm, for example the greeting based on the existence of the first name

To allow smarty in message templates, adjust you civicrm settings

```
define( 'CIVICRM_MAIL_SMARTY', 1 );
```

Then in the template

```
{capture assign=first_name}{contact.first_name}{/capture}
Dear {$first_name|default:Friend},
{if $first_name}
  Hello, {$first_name}, how are you?
{/if}
```

Smarty Syntax - http://www.smarty.net/docsv2/en/language.function.if.tpl

Best Practices - http://www.smarty.net/docsv2/en/language.basic.syntax

# Smarty Message Template API call

Paste the below logic in Contribution Receipt (offline) message template

```
{if $contributionID}
 {crmAPI var='result' entity='Contribution' action='getsingle' sequential=1 id=$contributionID}
  {if $result.contribution_campaign_id}
  {crmAPI var='campaignresult' entity='Campaign' action='getsingle' sequential=1 id=$result.contribution_campaign_id}
  <tr>
            <td {$labelStyle}>
            {ts}Campaign{/ts}
            </td>
            <td {$valueStyle}>
            {$campaignresult.title}
            </td>
  </tr>
  {/if}
{/if}
```

# Checksum Authentication

There is a very powerful feature to allow you to send an email with CiviMail to your constituents with a link to the contribution form, profile, or event registration form with all of their contact information already filled in! This saves them the hassle of filling it out and increasing the chances they would ultimately donate.  Or it can be a simple way to periodically allow people to review their contact info and update it if applicable.

The way it works is you create a "special" link in the CiviMail message that includes the checksum token {contact.checksum}. When people click on the special link, it looks them up in the database and prefills any information on the contribution form or profile with any data that exists in their record. The special link lasts for seven days from the time it was sent out. e.g.

**Checksum for Contribution Pages**: To send people to a contribution page use this path where **N** is the ID of your contribution page:

- Drupal: http://www.myorganization.org/civicrm/contribute/transact?reset=1&id=**N**&{contact.checksum}&cid={contact.contact_id}
- Joomla!: http://www.myorganization.org/index.php?option=com_civicrm&task=civicrm/contribute/transact&reset=1&id=**N**&{contact.checksum}&cid={contact.contact_id}
- WordPress: http://www.myorganization.org/?page=CiviCRM&q=civicrm/contribute/transact&reset=1&id=**N**&{contact.checksum}&cid={contact.contact_id}

How is it determined and how do you nullify it in case of emergency?

```
CRM_Contact_BAO_Contact_Utils::generateChecksum
```

Emergency, disable the page in question or reset everyones hash.

# Cron and Scheduled Jobs

Automated tasks that need to be run e.g.

- Activity Processor/Process Inbound Emails
- Disable expired relationships
- Greetings Updater
- Mail Reports

For full list see:

http://wiki.civicrm.org/confluence/display/CRMDOC/Managing+Scheduled+Jobs

To setup the Cron use

http://wiki.civicrm.org/confluence/display/CRMDOC/Running+Command-line+Scripts+via+URL

# Add a new scheduled job (API)

For example, we need to new scheduled job to populate a custom field in CiviCRM with the total giving for a contact every night.

Civix to the rescue!

1. Generate a new API `civix generate:api -h` to get the syntax.

e.g. `civix generate:api Contact contributiontotal --schedule Daily`

Action names should be lowercase. The javascript helpers CRM.api() and CRM.api3() force actions to be lowercase. This issues does not present itself in the API Explorer or when the api action is called via PHP, REST, or SMARTY

2. Install/Uninstall the extension
3. Check your new scheduled job appears
4. Complete the SQL to populate the custom field with the total giving for all contacts.

# Moving CiviCRM

For example cloning live site for development or staging;

Always follow the instructions!

http://wiki.civicrm.org/confluence/display/CRMDOC/Moving+an+Existing+Installation+to+a+New+Server+or+Location

Key steps

- Clearing the caches both at DB and Filesystem
- Setting paths correctly before trying to access
- Rebuild the menu

If a real problem install or changing CMS then

- Install fresh working copy of CiviCRM with same version
- Copy over the DB and custom directories (leave the existing settings files)
- Clear out caches

# Smarty Helpers - Extending Smarty

The Smarty templating language that is used to output the HTML for CiviCRM's templates is fairly powerful in its own right, but sometimes it can't do everything you want.  Smarty can be extended using "plugins" -- you can find some examples and documentation online at Smarty's website or by searching the web.

To install Smarty plugins without editing CiviCRM core (which is difficult to maintain), you will have to implement the hook_civicrm_pageRun hook.  (Activating hooks in Drupal and Joomla is covered here).

Once you've created your module or hook file, you can retrieve the Smarty template object like so:

```
function yourmodule_civicrm_pageRun(&$page) {

    $template = $page->getTemplate(); // retrieve Smarty template object

    array_push($template->plugins_dir, '/absolute/path/to/custom/plugin/directory'); //
add your custom template path to the list of locations Smarty looks for plugins

}
```

# Smarty Helpers - Extending Smarty

Then in that custom plugin directory, you can place whatever Smarty plugins you need.

You can also use this trick to change other Smarty behavior, such as whether it can evaluate PHP placed directly in templates. For instance:

```
function yourmodule_civicrm_pageRun(&$page) {

    $template = $page->getTemplate(); // retrieve Smarty template object

    array_push($template->security_settings['MODIFIER_FUNCS'], 'explode'); // allow the explode() php function to be used as a "modifier" in Smarty templates

}
```

However, be very careful with these settings – if you don't know what you're doing, you can open security holes or create a mess of code that's difficult to maintain.

# DAOs -> BAOs -> APIs

Converting sql work to DAOs. E.g: Training Extension.

Writing BAOs

Writing APIs - http://wiki.civicrm.org/confluence/display/CRMDOC/API+Architecture+Standards

# Debugging:

- Printing PHP variables
- Debug mode
- Printing smarty variables
- Backtrace
- Log files
- Clearing the cache
- Check the queries fired by Dataobject
- Xdebug

# Debug Mode: Printing

```
CRM_Core_Error::debug($name, $variable = null, $log = true, $html = true);
CRM_Core_Error::debug_var('name', $variable = null);
```

# Debug Mode:

Administer > System Settings > Debugging

- Smarty Debug Window - Loads all variables available to the current page template into a pop-up window. To trigger, add *&smartyDebug=1* to any CiviCRM URL query string. Make sure you have pop-up blocking disabled in your browser for the CiviCRM site URL.
- Session Reset - Resets all values in your client session. To trigger, add *&sessionReset=2*
- Directory Cleanup - Empties template cache and/or temporary upload file folders.
  - To empty template cache (civicrm/templates_c folder), add *&directoryCleanup=1*
  - To remove temporary upload files (civicrm/upload folder), add *&directoryCleanup=2*
  - To cleanup both, add *&directoryCleanup=3*
- Display Current Session - Displays the current users session variables in the browser, *&sessionDebug=1*
- CRM_Core_Error::backtrace()

# Debug Mode:

CRM_Core_Error::debug log=true

**Clearing the cache**

Using Drupal, you can clear all caches with the following **drush** command :

- drush cc civicrm
- drush civicrm-cache-clear *(older versions only)*

Alternatively, you can call the following two methods:

- CRM_Core_Config::clearDBCache();
- CRM_Core_Config::cleanup();

**Check the queries fired by Dataobject**

define( 'CIVICRM_DAO_DEBUG', 1 );

**XDebug:** http://wiki.civicrm.org/confluence/display/CRMDOC/Debugging+for+developers

# Submitting Patches: GIT

http://wiki.civicrm.org/confluence/display/CRMDOC/GitHub+for+CiviCRM

Fork and Branch

# Any Questions?