Making Extensions

Extensions allow you to modify existing SI features or add new ones with minimal impact on other features and with an ability to preserve the changes when SI updates are loaded. There are many points in the code that look to see if a version of a file exists in an active extension. In each case, if found, the version of the file in the extension will be used in place of the file in the library. A list of what files can be replaced exists in the order of access at the end of this document.

The following examples assume that you know how to write PHP and Smarty template code. You don't have to be an expert. Much of what you need to know already exists in these files and can guide you in adding your changes.

The following examples should help you understand the basics of developing an extension.

Example – Adding invoice total to the Recurrence management list

The first step in creating an extension is to decide on the extension name. For this example, we will use **"cronInvoiceTotal".** Now we can add a directory of this name in the **"extensions"** directory. This means we have path, **"extensions\cronInvoiceTotal".**

In this directory we create the **DESCRIPTION** file. In it we store a short description of the extension that will be displayed on the **Extensions** screen. Add the line, **"Recurrence Invoice Total"**, to this file.

We also create a **README** file that contains the full description of the extension and if necessary, instructions on loading and using the extension.

Next look at the standard **Recurrence** management list screen (select **Money** tab and then the **Recurrence** menu option, you will see the following in the browser address line:

`https://`*`yourSIdomain`***`/index.php?module=cron&view=manage`**

From this line we see the following pertinent pieces of information:

1) **module=cron** --- This identifies two directories. The directory containing the PHP file used to access data for the screen, and the directory containing the ***Smarty*** template file use to render the screen in your browser.

   In this example, the PHP file exists in the **modules/cron** directory and the ***Smarty*** template file exists in the **templates/default/cron** directory. Note that all template directories are in the **templates/default** directory.

2) **view=manage** --- This identifies both the name of the PHP file and template file.

   Combining the information above with the **view** information, we know that the **modules/cron/manage.php** file is used to access information to be displayed, and the **templates/default/cron/manage.tpl** file is used to build the information displayed in your browser.

For this example, we need to modify both files to add the invoice total to the screen and we can add these directories and files to the extension directory. So, we have the following paths:

Making Extensions

extensions\cronInvoiceTotal\modules\cron\manage.php

and

extensions\cronInvoiceTotal\templates\default\cron\manage.tpl

Copy the content of the standard cron manage.php and manage.tpl files to the files in these directories. These are the files you will modify.

In the **manage.php** file we have the line:

```
$crons = Crons::manageTableInfo();
```

This line is accessing the information to display on the screen. We need to add the invoice total amount to each record in the array of records now in the **$crons** variable.

If you look at the logic in the **manageTableInfo()** function of the **Inc\Claz\Crons** class, you will see code that stores information in the **$tableRows** array. Note that each entry in the **$tableRows** array is itself an array representing each line that will be displayed on the Recurrence screen. Also, there is an entry in each entry of the array identified by the key, **"invoiceId."**

Now it gets tricky as the value we need to access the invoice record is stored as the **"id={$row['invoice_id]}"** portion of the hyperlink that is shown as **"invoiceId."** Note that there is a different value stored as the link text. This is the **"index_id."** The reason these are different is that the **"invoice_id"** is the **"id**" field of the invoice record which uniquely identifies a record in the **"si_invoices"** table. But since different, possibly overlapping index numbering schemes can be used, the **"index_id"** that you see displayed on the screen might not uniquely identify the invoice record.

Because the **"invoice_id"** value is embedded in the hyperlink, we need to use code to find and remove it. The best method is to use the **preg_replace()** function. The code to extract the id is:

```
$pattern = "/^.*href.*id=(\d+)'>\d+<\/a>$/";
$id = preg_replace($pattern, $1);
```

Now that we have the ID, we can get the invoice record as follows:

```
$rec = Invoice::getOne($id);
```

Note that you will need to as the line, **"use Inc\Claz\Invoice"**, to the top of the file following the other **"use"** lines.

Now we must make a new array for **$crons** that has entries with the **"total"** in them. Because we want to format the amount properly on the screen, we also include the **"locale"** and **"currency_code"** fields. The following shows the code to accomplish this for the **modules/cron/manage.php** file sans comments:

```
<?php

use Inc\Claz\Cron;
use Inc\Claz\Invoice;
use Inc\Claz\Util;

global $pdoDb, $smarty;
```

Making Extensions

```
Util::directAccessAllowed();

$crons = Cron::manageTableInfo();
$newCrons = [];
foreach ($crons as $cron) {
    $subject = $cron['invoiceId'];
    $pattern = '/^.*href.*id=(\d+)'>\d+<\/a>$/';
    $id = preg_replace($pattern, "$1", $subject);

    $rec = Invoice::getOne($id);
    $cron['total'] = $rec['total'];

    $subject = $rec['locale'];
    $pattern = '/^(.*)_(.*)$/';
    $cron['locale'] = preg_replace($pattern,'$1-$2',$subject);
    $cron['currency_code'] = $rec['currency_code'];

    $newCrons[] = $cron;
}

$data = json_encode(['data' => $newCrons]);
if (file_put_contents("public/data.json", $data) === false) {
    exit("Unable to create public/data.json file");
}

$smarty->assign("numberOfRows", count($newCrons));

$smarty->assign('pageActive', 'cron');
$smarty->assign('activeTab', '#money');
```

Not the logic for altering the **"locale"** field using the **"preg_replace"** function. This was copied from the **Invoice::manageTable()** function that retrieves the data to be shown on the Invoices management screen. The basic result is the underscore in the locale is changed to a dash, ex: **en_US** becomes **en-US**. This is required by the **DataTables render** function.

The net effect of this code is that a field named **"total"** now exists for each recurrence record and can be added to the **Recurrence** screen.

The next step is to add the total field to the **"extensions\cronInvoiceTotal\templates\default\cron\manage.tpl"** file. Fortunately, this is less complex than what we did to the **manage.php** file.

First, we need to decide where the total field to be placed. For this example, we will add it as the last field on the screen.

Making Extensions

To do this, we need to add a heading for the field. Checking the **lang/en_US/lang.php**[1] file for the word **"Total"**. It is in the **$LANG['totalUc']** variable. Adding it to the heading list you will yield the following:

```
<th>{$LANG.customerUc}</th>
<th class="align__text-center">{$LANG.totalUc}</th>
```

Now we add the actual data field that will be displayed. Looking further down in this file, you find the **DataTables** section. Continuing down, you find the **"columns"** list. The **total** field we added to the **$newCrons** table is now added to this list. Iif you just add the field, you will get the total but it won't be formatted as amount with proper monetary sign and decimal places. So we are going to go the invoice management screen and copy the **total** field logic from there. Doing this we have:

```
{ "data": "customerName"},
{
      "data": "total",
      "render": function (data, type, row) {
            let formatter = new Intl.NumberFormat(row['locale'], {
                  'style': 'currency',
                  'currency': row['currency_code']
            });
            return formatter.format(data);
      }
}
```

Now we must position the field. This screen uses specified field sizes. In this example we will reduce the size of the **customerName** field from **30%** to **20%**. This allows us to use **10%** as the size for the **total** field. Also, we want to right justify this field by setting the **className** to **'dt-body-right'**.

Making this change we alter the size of the **customerName** field an place a **comma**, (**','**) following it. Then add the **total** field. Here is the code showing this change:

```
{ "targets": 7, "width": "20%" },
{ "targets": 8, "width": "10%", "className": 'dt-body-right' }
```

Whew!!! That's all. Now we are ready to enable our extension and put it into use. Log in to SI and select the **Settings** tab and the **Customize** menu option. On the screen that comes up, select the **Extensions** menu option.

On the **Extensions** screen, you now see a line for the **cronInvoiceTotal** extension. At the right of the **cronInvoiceTotal** extension line there is an unlit lightbulb and a non-green leaf in the status field. The leaf turns green when you have registered the extension. And the light bulb turns yellow when you have enabled to module.

On the left there is a green leaf with a white plus sign on the lower right corner. The white plus sign means that you can click this icon to register the module. After you click this icon, you will be on the registration

[1] The lang/en_US file is used as it contains all the values that can be used. Some of these may not have been translated into your local language file so by default, that value comes from the **lang/en_US/lang.php** file.

screen. Note the title say's you are "**About to *register*: cronInvoiceTotal**". Simply select the **Save** button to register the module.

You are returned to the **Extension** management screen and the leaf on the right side of the **cronInvoiceTotal** line is green. This means that the module is registered and can be turned on.

On the left side of the screen there is now a **light switch** icon. Also note that the previously white plus sign on the leaf is now red. The **red** means that you can select the icon to **unregister** the module.

At this point, we want to enable the module. So we select the **light switch** icon and the **light bulb** on the left will turn **yellow** showing the module is enabled.

To see the module in action, select the **Money** tab and select the **Recurrence** menu option. Note that the **Total** column now appears and contains the right-justified, properly formatted total amount for this invoice. You can verify this by select the link to the invoice when is the **Invoice ID** shown for the invoice.

You can see the **Total** for the invoice in the **Invoice** section of the **Financial status** box on the bottom left of the screen.

If you decide that you no longer want to use this module, you simply go to the **Extensions** screen and select the **Light Switch** icon. This causes the **Light Bulb** icon on the right to turn off (no longer yellow). Go to the **Recurrence** screen and the **Total** field is no longer displayed.

Locations in code where extension files looked for / loaded

As mentioned, I stated that there are many locations where an extension can replace the standard code SI uses. The following list details the files that can be replaced in the order that SI accesses the files:

extensions/$extName/include/smarty_plugins	Allows user to replace/add to standard smarty plugins with their own.
extensions/$extName/Inc/Claz/" . $extName . "acl.php";	For ACL (Access Control List) in an extension, simple place any of the following commands in the acl.php file for the extension. Note, see Inc/Claz/SiAcl.php for examples of each of the following: acl->addRole(role) acl->addResource(resource) acl->allow(role, permission, resource) acl->deny(role, permission, resource)
extensions/$extName/include/init.php	This follows immediately after the standard **init.php** file is processed. It doesn't replace it but supplements it.

Making Extensions

extensions/$extName/modules/$module/$view.php	If this is an **API, XML or AJAX** request, this is where the **modules php** file to access is set. Note that the **$module** and **$view** values are extracted from the browser address line. At this point, **modules.php** is executed to render the screen and no further processing occurs.
extensions/$extName/include/js/$extName.jquery.ext.js	Adds user jQuery files to the html screen. Note that the jQuery file must be named: **jquey.ext.js**
extensions/$extName/templates/default/hooks.tpl	Allows extension level **hooks** to be accessed. Note that **hooks** are a topic of their own. So, use this only if you understand their function.
extensions/$extName/templates/default/header.tpl	Allows the extension to use its own **header.tpl** file. This replaces the default file.
extensions/$extName/modules/$module/$view.php	Note that this extension occurs above. However there it is limited to **API, XML or AJAX** requests. This instance is where it is provided for all other (aka standard) requests. This is where your extension **modules** file is loaded unless it is for the **reports/index.php** or **system_defaults/edit.php** file. These special cases do not replace the php file. Rather they use logic to insert code into **modules/system_defaults/edit.php** file is a special case explained in **Example 2** above.
extensions/$extName/templates/default/reports/$view.php	This is for **reports** module and either **export.php** or **email.php** files only.
extensions/$extName/include/js/$extName.post_load.jquery.ext.js.tpl	This is a jQuery script loaded after the module php has been loaded.

Making Extensions

extensions/$extName/templates/default/main.tpl	This replaces the standard **templates/default/main.tpl** file.
extensions/$extName/templates/default/$module/$view.tpl	This is where the extension view file is loaded. Note there is special processing for a module/view settings if **reports/index** or **system_defaults/manage**.
extensions/$extName/templates/default/footer.tpl	This replaces the standard **templates/default/footer.tpl** file.