

2

Customizing the SugarCRM Application Content

e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

In Chapter 1, we looked at changing the look and feel of a basic SugarCRM implementation. In particular we examined Pygoscelis P. Ellsworthy's organization – Penguin P. I. – and saw how to introduce the day-to-day terminology that his staff uses. You will also remember that we started to change the general look of the screen by introducing our own custom theme.

In this chapter we're going to start with adding our own functionality into SugarCRM. Nothing too elaborate, and we won't touch any of the core functionalities (yet). We'll just see how easy it is to add your own tab screens and **Dashlets** – your own GUI components.

A Note About Terminology

In Chapter 1, we've been referring to **Tab** screens, but you must have already realized that the information for these are stored in a directory named `modules`.

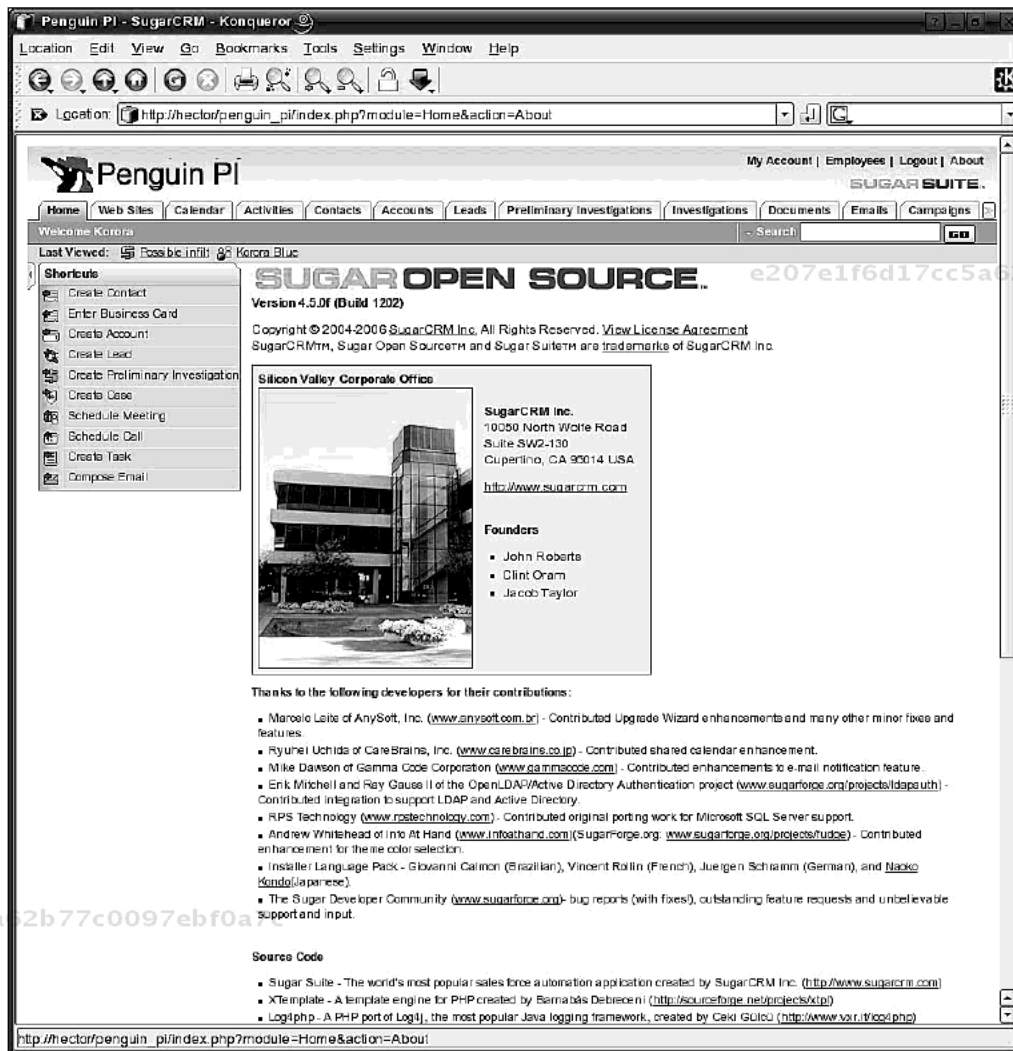
That's because SugarCRM consists of a number of components (i.e. *modules*). If a module has a tab screen then (in SugarCRM talk) this is a *module* tab. OK, got that? Right, let's look at one of the modules – the **Home** module. We're actually going to change the impact of clicking on the **About** link.

e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

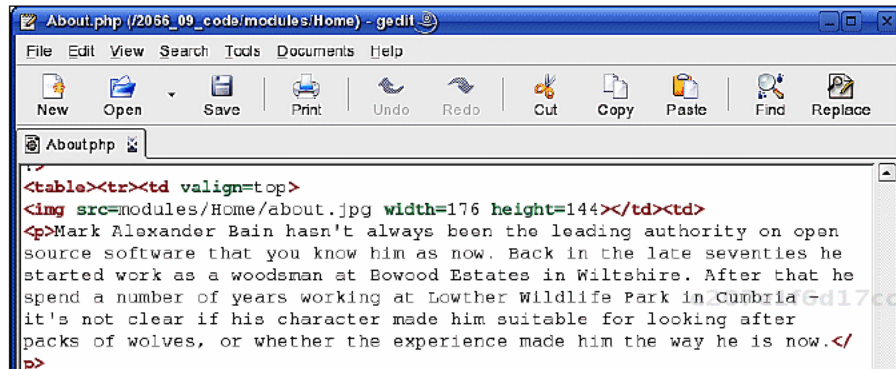
Changing the About Screen

If you click on the **About** link you'll see something like:



All very interesting, but it doesn't really relate to your project or the organization in which SugarCRM is going to be used. However, it does tell us a lot about how SugarCRM is structured. If you look at the URL you can see that we're using the **Home** module and the action is **About**. This means that if you do want to change the contents of the **About** page then you need to look in the `modules/Home` directory – where you will find the `About.php` file. After taking a backup of the file (just for peace of mind) you can edit it so that it contains the information that you

want to display. How you edit the file is up to you; for instance, I use the Linux text editor GEdit:



However, once you've saved About.php you'll be able to view it via your browser:



Of course, you may decide that you want to make the **About** screen more useful, something that will be helpful to your users—i.e. a *help* screen.

Changing the About Screen into a Help Screen

The first thing that you may want to do is to change the link text from **About** to something more appropriate—such as **Penguin P.I. Help**. To do that we need to return to the `custom/include/language/en_us.lang.php` file that we worked with in Chapter 1. Just add a line:

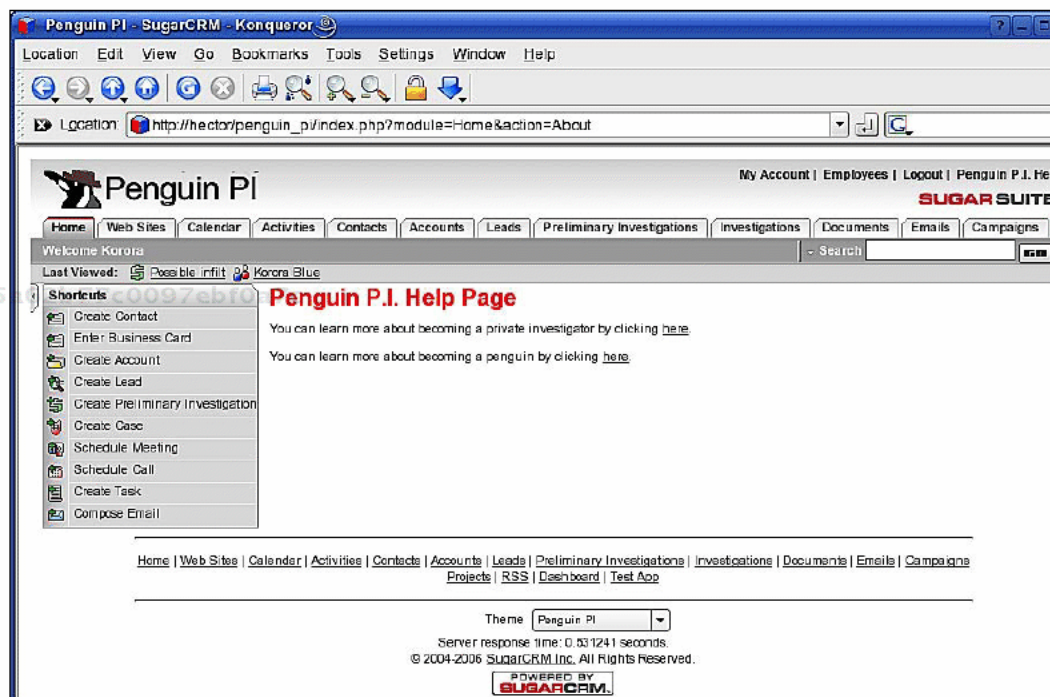
```
$app_strings['LNK_ABOUT'] = 'Penguin P.I. Help';
```

e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

And then refresh your browser:



Now you just need to modify `modules/Home/About.php` again so that it contains some helpful information and refresh the page on the browser:



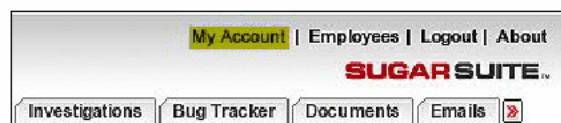
Of course, at this point you're probably thinking that this is all very interesting, but what you actually want to do is to start creating your own tabs. Obviously that's what we need to look at next.

Controlling the Visible Tabs

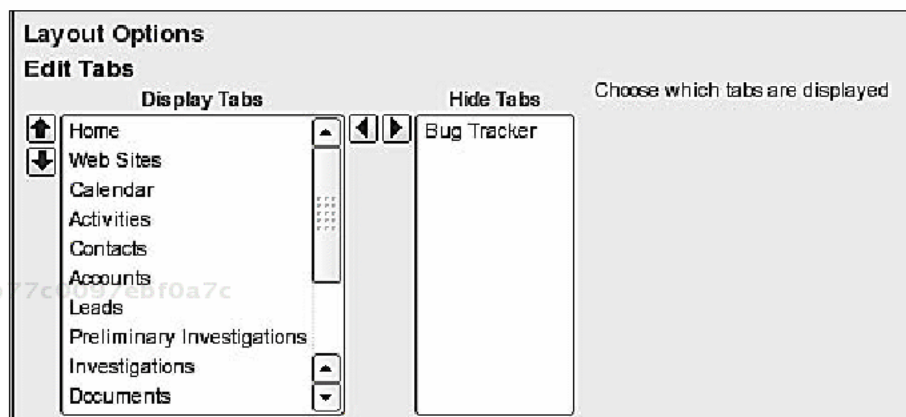
Before we create a new tab it's probably worth having a look at how we can control the visibility of SugarCRM tabs to our users.

User Control

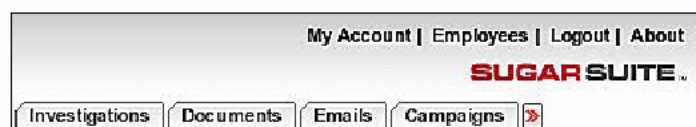
In fact, in an *out-of-the-box* SugarCRM installation, any user can choose which tabs are visible by clicking on **My Account**:



Then, for example, if Korora wishes to remove the **Bug Tracker** she can do this in the **Layout Options** section:



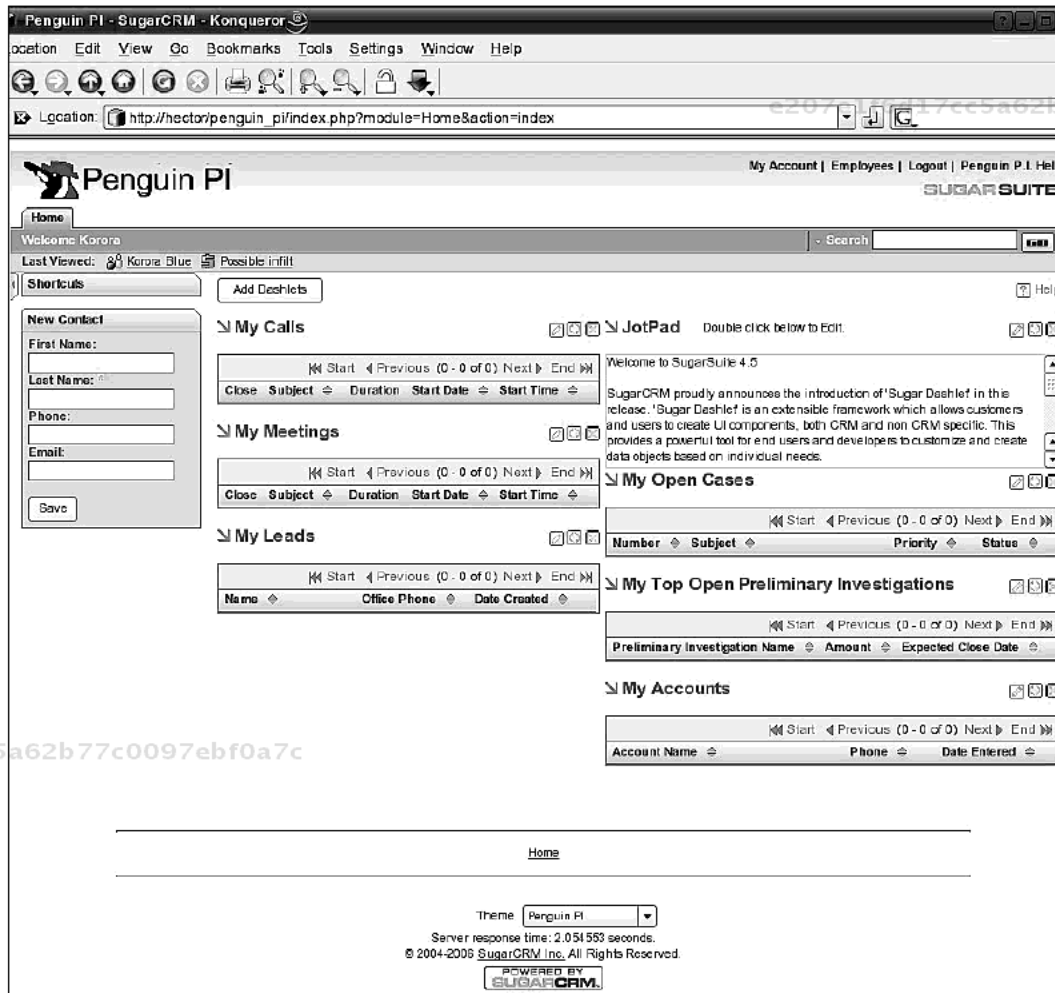
When she clicks the **Save** button then **Bug Tracker** will no longer be visible in the list of tabs:



However, while this is useful it does have its drawbacks:

- User control of tab visibility will make it more difficult for you to create a single set up for your organization.
- Users may choose not to view the tabs that you are going to create.






It's worth noting that there is only one tab that can't be removed by the user – the **Home** tab. However, this can still lead to some extreme situations:



If that's the case then you may wish to limit the users' ability to change the tabs to be shown – the least it will do is prevent a call to the Help Desk from Korora saying "I'm not sure what has happened, but I can't access my emails anymore".

Administrator Control

If you want *only* the administrator to be able to set the visible tabs then you need to log on to your **admin** account, and go to the **Admin** screen, and then click on **Configure Tabs**:

Studio			
 Studio	Edit Dropdowns, Custom Fields, Layouts and Labels	 Portal	Add tabs which can display any web site
 Configure Tabs	Choose which tabs are displayed system-wide	 Configure Group Tabs	Create and edit groupings of tabs
 Rename Tabs	Change the label of the tabs		

Now you can decide:

- Whether or not your users are allowed to configure their own tabs
- Which tabs are available to your users

By default users *are* allowed to configure their own tabs, so uncheck **Allow users to configure tabs**, and then drag and drop tabs until you've got the setup that you require:

SaveCancel

☐ Allow users to configure tabs

Display Tabs

■ Home

■ Web Sites

■ Calendar

■ Activities

■ Contacts

■ Accounts

■ Leads

■ Preliminary Investigations

■ Investigations

■ Documents

■ Emails

■ Campaigns

■ Projects

■ RSS

■ Dashboard

Hide Tabs

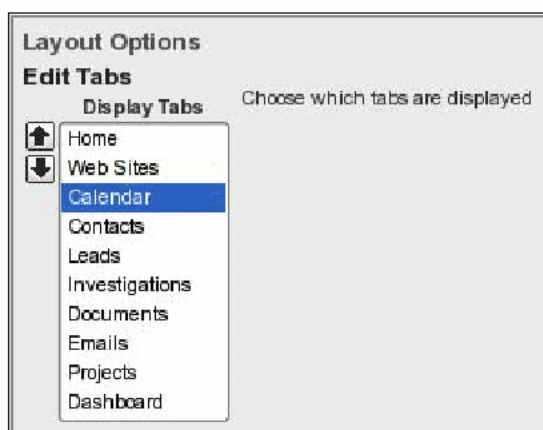
■ Bug Tracker

At this point it's worth considering the browser that you're using. You will find that this screen will work well with:

- Firefox
- Internet Explorer
- Konqueror
- Safari

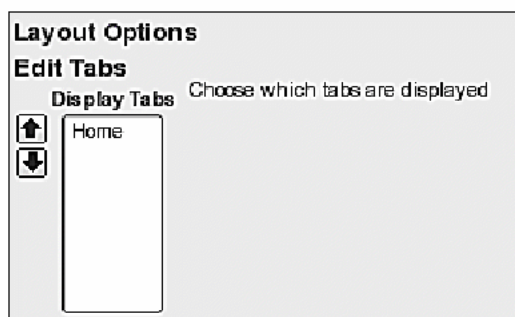
Unfortunately, it won't work with Opera (my personal favorite).

Once you've clicked on the **Save** button your users will not be able to disable any of the tabs, and they will only be able to view the ones that you have selected for them:



A Note about Administering Live Systems

If you decide to restrict tab selection to administrators only on a live system then don't walk away expecting no problems. Let's imagine that the Help Desk has explained to Korora that she needs to click on **My Account** to solve her problem. What she'll see is:



This is, of course, because Korora had previously removed all the other tabs, and now you've removed her ability to add any back in again. Fortunately the solution is quite simple – she just needs to scroll up to the top of her **My Account** screen, to where she'll see **Reset To Default Preferences**:

Users: Korora Blue (bluek) Print Help

[Edit](#) [Change Password](#) [Reset To Default Homepage](#) | [Reset To Default Preferences](#)

Name: Korora Blue	User Name: bluek
Status: Active	

Clicking this link would cause Korora to be logged out, but once she logs back in her tabs would be set up as you had defined them.

And it is worth pointing out that when a user does this *all* of their preferences will be reset, so the first thing that they'll see will be:

Users: Korora Blue (bluek) Print Help

[Edit](#) [Change Password](#) [Reset To Default Homepage](#) | [Reset To Default Preferences](#)

Name: Korora Blue	User Name: bluek
Status: Active	

One worrying effect that all this has is that your user's email signature (if they have one) will suddenly stop appearing when they create new emails. Don't worry – the signature hasn't been deleted, it has just been turned off. Your user can turn it back on by going into **My Account**, clicking **Edit** and then selecting the signature under **Email Options**:

Email Options

Email address: <input type="text" value="korora.blue@linuxtalk.co.uk"/>	Other email address: <input type="text"/>
Reply-to name: <input type="text" value="Korora Blue"/>	Reply-to address: <input type="text"/>
Signature: <input type="text" value="Korora's Default"/> Create Edit	Signature above reply? <input type="checkbox"/>
Email client: <input type="text" value="System Default Mail Client"/>	Show email counts? <input type="checkbox"/>
Compose format: <input type="text" value="Default Email Format"/>	Outbound Character Set: <input type="text" value="ISO-8859-1 (Western European and US)"/>

Now, if you don't want to leave it to individual users to reset their default tabs you can do this in bulk—but not via the SugarCRM application itself. You'll need to log on to your SugarCRM database and use some SQL:

```
update user_preferences
set contents=null
where category='global'
```

But be warned—this will reset the default preferences of *all* users.

Adding a Custom Tab

If you want to create a custom tab for SugarCRM then you'll need to start by creating a new module (remember that a tab is actually a *module* tab). You may be surprised to learn that this is *very* easy. You'll find that there are four steps:

1. Create a directory for your module.
2. Create four default files—Forms.php, index.php, language/en_us.lang.php, and a PHP file with the same name as your module—none of these files need contain anything but they must exist.
3. Update include/module.php to tell SugarCRM that your new tab exists.
4. Update custom/include/language/en_us.lang.php so that it contains the text to be displayed for the tab (just as we did when we renamed tabs in Chapter 1).

So, let us look at those steps in a bit more detail. On Windows or Linux you can create the required directories and files via your file managers or on the command line; for example, on Linux you could move to your SugarCRM directory and then type:

```
mkdir -p modules/TestApp/language
touch modules/TestApp/language/en_us.lang.php
touch modules/TestApp/Forms.php
touch modules/TestApp/index.php
touch modules/TestApp/TestApp.php
```

Then edit include/module.php and add the lines:

```
$moduleList[] = 'TestApp';
$beanList['NewTab'] = 'TestApp';
$beanFiles['NewTab'] = 'modules/TestApp/TestApp.php';
```

Finally edit `custom/include/language/en_us.lang.php` so that it contains the line:

```
'TestApp' => 'Test App',
```

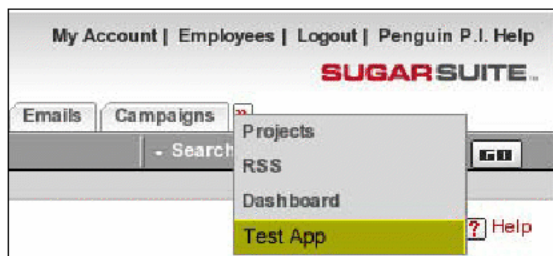
for example:

```
$app_list_strings['moduleList'] = array (  
    'TestApp' => 'Test App',  
    'Home' => 'Home',  
    'Dashboard' => 'Dashboard',  
    'Contacts' => 'Contacts',  
    'Accounts' => 'Accounts',  
    'Opportunities' => 'Preliminary Investigations',  
    'Bug Tracker' => 'Bug Tracker',  
    'Test App' => 'Test App',  
);
```

Now, admit it—you've done that, refreshed your browser and there's no change there? Well, don't worry, you haven't done anything wrong—you're just jumping the gun a little. First you need to log on as an administrator, and go to the **Admin** screen where you'll find your new tab under **Hide Tabs**:



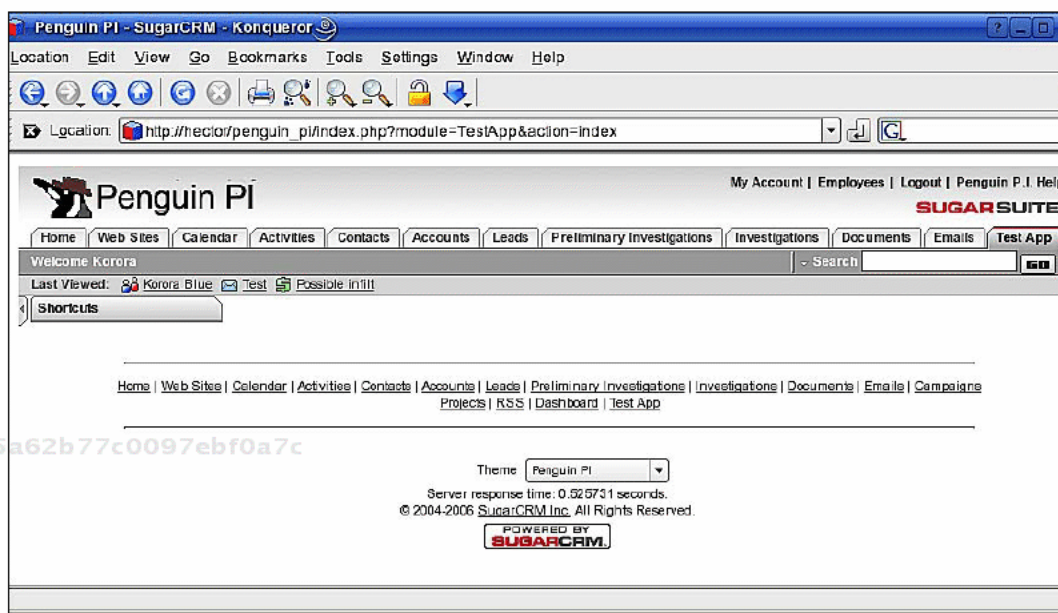
You'll need to drag your new tab into **Display Tabs**, and then it will be made available to your users:



Now it's just a matter of what you want to show in the tab screen...

Custom Tab Contents

At the moment, if you view the **Test App** tab then you'll see:



I'm sure you'll agree that it is nice to see that we can create a new tab, but it's not the most interesting thing in the world, is it?

You will remember that our new module (TestApp) actually consists of four default files:

- modules/TestApp/language/en_us.lang.php
- modules/TestApp/Forms.php
- modules/TestApp/index.php
- modules/TestApp/TestApp.php

You will also remember that these files *do* need to exist, but they *don't* need to contain anything. Obviously the next stage is to edit these files in order to add contents to the tab. In fact we only have to edit one of the files—modules/TestApp/index.php. So, you could start by adding some HTML code to the page:

```
<HTML>
<H1>Test Application</H1>
</HTML>
```

But that's boring—and you don't want to be boring, do you? It would seem more sensible to add something more interactive. But what? Since all of the modules are written in PHP then we can use them as a starting point. For example, you can take a little code from the **Emails** module, and a little code from the **Opportunities** module:

```
<?php
include ('modules/Emails/language/en_us.lang.php');
include ('modules/Emails/ListView.php');

include ('modules/Opportunities/language/en_us.lang.php');
include ('custom/modules/Opportunities/language/en_us.lang.php');
include ('modules/Opportunities/ListView.php');
?>
```

This code will produce:

Emails: My Inbox Print Help

Email Search

Email Status: Subject: Contact:

My Inbox

| Selected: Start Previous (0 - 0 of 0) Next End

Subject	Contacts	Related to	Reply	Status	Date Sent
---------	----------	------------	-------	--------	-----------

| Selected: Start Previous (0 - 0 of 0) Next End

Preliminary Investigation Name: Account Name: Only my items: ☐

Preliminary Investigations List

| Merge Duplicates | Selected: Start Previous (0 - 0 of 0) Next End

Preliminary Investigation	Account Name	Investigation stage	Amount	Close	User
---------------------------	--------------	---------------------	--------	-------	------

| Merge Duplicates | Selected: Start Previous (0 - 0 of 0) Next End

Mass Update

Assigned to:

Type:

Lead Source:

Investigation stage:

Account Name:

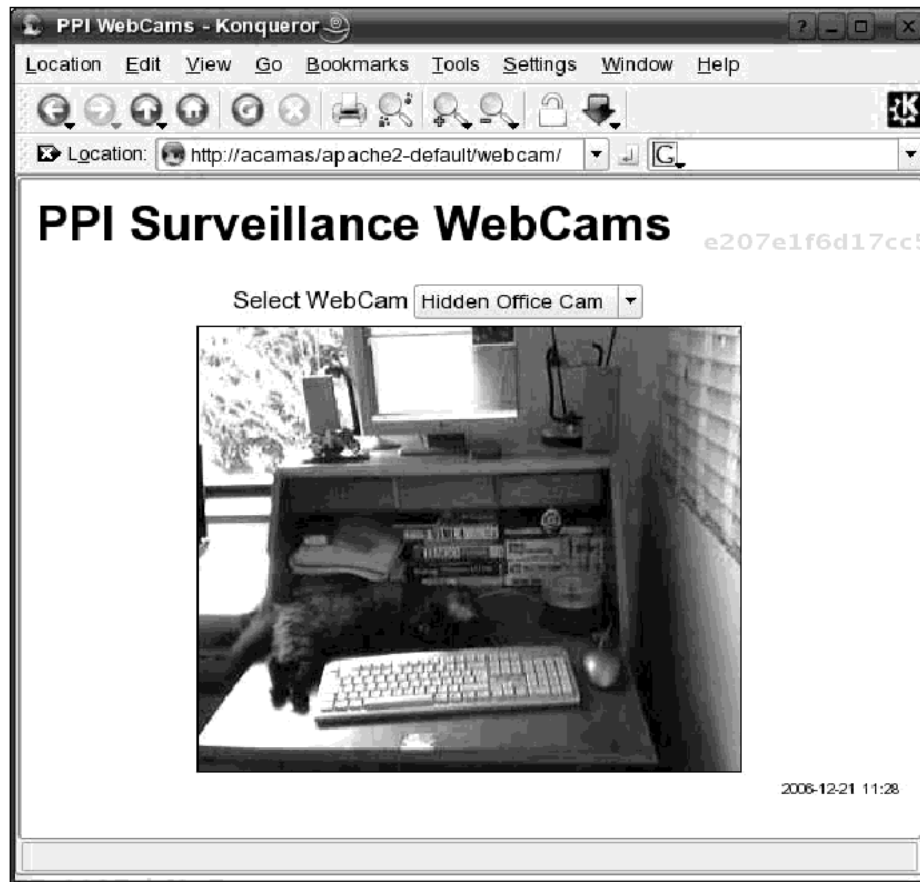
Expected Close Date: (yyyy-mm-dd)

Having just said 'Don't use HTML, use PHP instead', I'm now going to say 'Actually, there is something interesting in HTML that you can make use of.' Why? Well, it's probable that you've got some useful applications that people are already using (and don't particularly want to lose). If these are web-based then you've got a few options:

- Carry on using the existing applications in parallel with SugarCRM— not the best idea since it means that you can't have a single, global point of reference, and can cause a bit of a headache when it comes to maintenance.
- Re-write all of the software into SugarCRM— good plan, but a bit time consuming, plus it may delay the launch of your application.
- Incorporate the existing applications directly into SugarCRM— now that sounds like a good idea.

And that's where the HTML tag `<IFRAME>` comes in.

Let's imagine that Pygoscelis has already employed someone to create a web page that enables staff to use webcams when carrying out some surveillance:

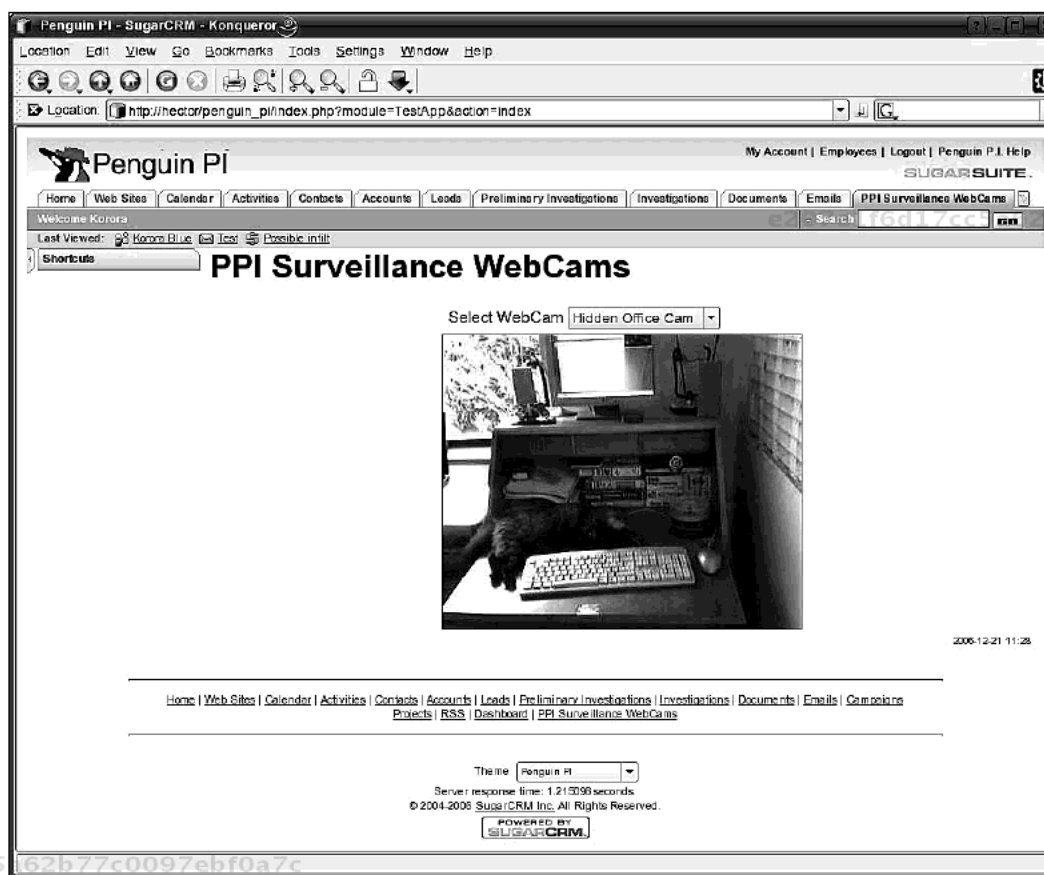


You'll find that you can incorporate any such web page very easily. First we need to edit `modules/TestApp/index.php` so that it contains:

```
<IFRAME SRC="http://acamas/apache2-default/webcam" WIDTH=100%  
HEIGHT=400>  
</IFRAME>
```

Of course, you'll need to change the web page to one that you can actually access. And you might want to change the TestApp title in custom/include/language/en_us.lang.php to something more appropriate:

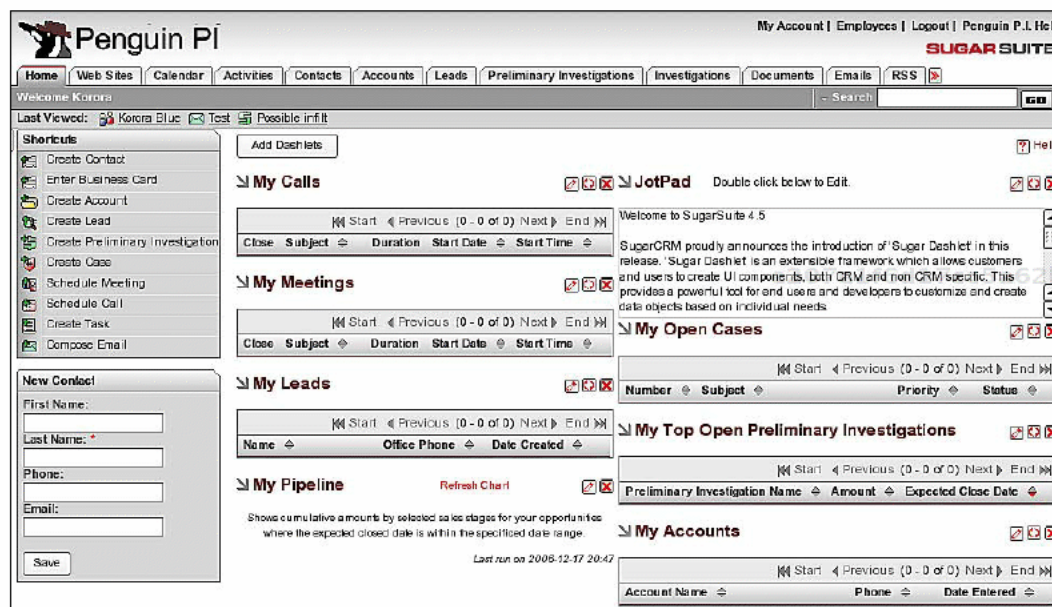
```
'TestApp' => 'PPI Surveillance WebCams',
```



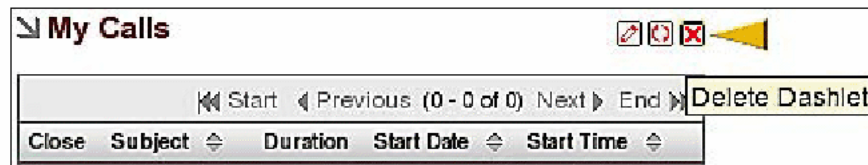
We will return to developing tabs throughout the book, but for now we're going to look at another aspect of customizing the application content – **Dashlets**.

User-Controlled Dashlet Customization

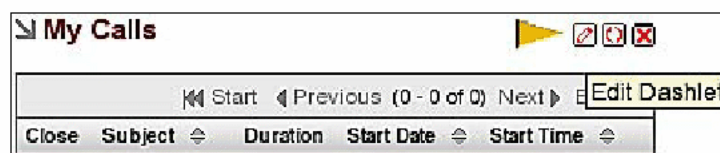
If you look at the **Home** tab then you'll see Dashlets in action:



Users can edit the screen by removing Dashlets:



Users can even customize each Dashlet themselves:



They can also decide which fields are shown in the Dashlet:

My Calls : Options

General

Title: My Calls

Display Rows: 5

Display Columns: Close, Subject, Duration, Start Date, Start Time

Hide Columns: Related to, Direction, Status, Date Created, Last Modified

Filters

My Calls: ☒

Start Date: -None-

Direction: Inbound, Outbound

Status: Planned, Held, Not Held

Assigned to: admin, bainm, bluek

Save

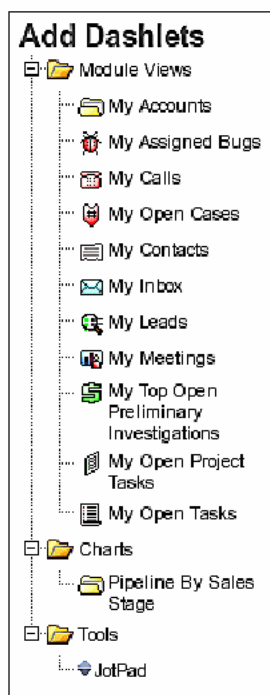
So that we see the tabs, and they can display exactly what they want:

My Calls

Start Previous (0 - 0 of 0) Next End

Subject Duration Start Date Start Time Status Related to

By clicking on **Add Dashlets** we can make the required changes:

e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

Your users can add any Dashlets that are included in the application. Obviously, we want to be able to give the users any extra Dashlets that they require in order to carry out their jobs effectively. So, that's what we'll look at next.

Customizing Dashlets

e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

After having created your own module tabs you've probably got a fair idea of how to create a new Dashlet. You're probably expecting to have to create a directory, and some default files – and you're quite right.

Creating Custom Dashlets

In order to create your own Dashlet you'll need:

- A directory in which to store the Dashlet files
- A meta file containing details of how the Dashlet should be displayed
- The Dashlet file itself – this contains the workings of the Dashlet itself

e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

So, the first thing that you need to do is to create a directory in which the dashlet is stored. This directory is going to be in the custom/modules area, and needs to take the format <dashlet name>/Dashlets/<dashlet name>. So, on Linux you can do this by typing:

```
mkdir -p custom/modules/PPIDashlet/Dashlets/PPIDashlet/
```

Or, obviously you could create the structure using a file browser on either Linux or Windows. Next you need to move to the new directory and create the meta file. As you'd expect it has to be named the same as your Dashlet, but has the suffix meta.php. In this case we'll need PPIDashlet.meta.php, and it should contain something like:

```
<?php
$dashletMeta['PPIDashlet'] = array(
    'title' => 'PPI Dashlet',
    'description' => 'A Dashlet for Penguin P.I.',
    'icon' => 'themes/PenguinPI/images/Tasks.gif',
    'category' => 'Tools');
?>
```

e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

Most of the file is self explanatory, the only thing that may be new to you is the category. However, if you look at the **Add Dashlets** dialog then you'll see that you have a choice of categories under which a dashlet can be located.

Next we need to create the dashlet file, PPIDashlet.php. In this case we're just going to get the dashlet to display some text:

```
<?php
//Start by including the base Dashlet class
require_once('include/Dashlets/Dashlet.php');
class PPIDashlet extends Dashlet
{
    function PPIDashlet($id, $def)
    {
        global $current_user, $app_strings;
        parent::Dashlet($id);
        $this->title = 'My PPI';
    }

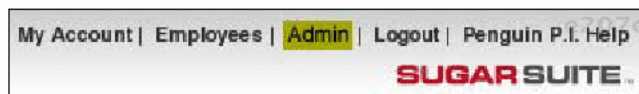
    function display($text = '')
    {
        $text = 'Dashlet for the PPI Organization';
        return parent::display($text);
    }
}
?>
```

e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

OK – not mind boggling functionality, but it's enough to show how to quickly create a Dashlet. As the book progresses we'll make the functionality more complicated. However, for the time being, let's look at how we make our new (simple) Dashlet available to our users.

Making Dashlets Accessible to Users

Although you've created your dashlet, no one will be able to see it yet. As you might expect, we have to do that through the admin account. Once you have logged on as an administrator, then you'll need to go to the **Admin** screen:



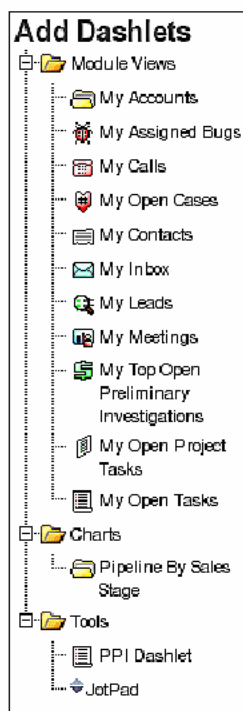
Having clicked on **Admin** you'll need to look for the **System** section, and then (even though this may seem strange) find the link marked **Repair**.

System			
System Settings	Configure system-wide settings	Backups	Perform a backup
Scheduler	Set up scheduled events	Repair	Check and repair Sugar Suite
Diagnostic Tool	Capture system configuration for diagnostics and analysis	Currencies	Set up currencies and currency rates
Upgrade Wizard	Upload and install Sugar Suite upgrades	Module Loader	Add or remove Sugar modules, themes, and language packs
Locale Settings	Set default localization settings for your system.		

You may not think that **Repair** is really a suitable label for this activity. However, the next stage *is* logical – the link that you'll have to look for now is named **Rebuild Dashlets**:

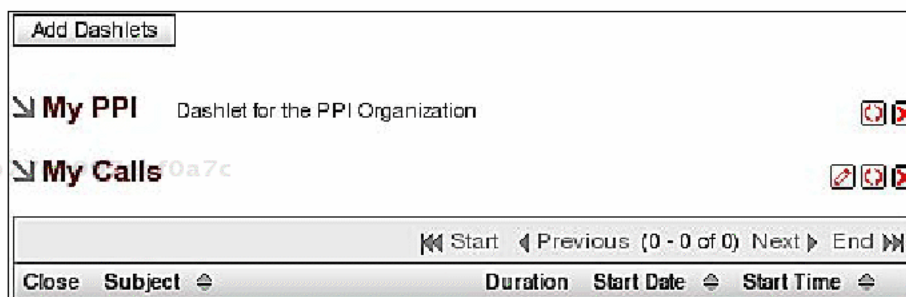
Rebuild Schedulers	Rebuild your out-of-the-box Scheduler Jobs.
Rebuild Dashlets	Rebuild the Dashlets cache file.
Rebuild Javascript Languages	Rebuild javascript versions of language files.

Your new Dashlet will now be available to all your users (in the **Tools** section):



e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

And the end result? Nothing too complicated yet, but it's a good starting point:



e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

With that completed you have the beginnings of your own custom SugarCRM implementation.

e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

Summary

In this chapter we've started to customize the SugarCRM application itself, and you're now able to add our own components in the form of module tabs and dashlets. You've also seen how to add our own About Page and modify the text for the link to the **About** screen. You can see that, by default, users can set which tabs are visible when they access SugarCRM. However, this option can be disabled via the admin account. Thus to sum it all up, this chapter covers various facets to modify SugarCRM to suit our needs

In Chapter 3 we will continue with the customization of module tabs and dashlets, as we start introducing custom fields into SugarCRM.

e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

e207e1f6d17cc5a62b77c0097ebf0a7c
ebruary

