From Technologies to Solutions

SugarCRM Developer's Manual

Customize and extend SugarCRM

Learn the application and database architecture of this open-source CRM and develop and integrate your own modules and custom workflows

Dr. Mark Alexander Bain

PACKT PUBLISHING

SugarCRM Developer's Manual

Customize and extend SugarCRM

Learn the application and database architecture of this open-source CRM and develop and integrate your own modules and custom workflows

Dr. Mark Alexander Bain

PACKT PUBLISHING

BIRMINGHAM - MUMBAI

SugarCRM Developer's Manual

Customize and extend SugarCRM

First published: June 2007

Production Reference: 1140607

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-847192-06-6

www.packtpub.com

Cover Image by Andrew Jalali (www.acjalali.com)

Credits

Author

Dr. Mark Alexander Bain

Reviewers

Andrew J. R. Whitehead

Michael J. R. Whitehead

Ajay Gupta

Susie Williams

Aspen Olmsted

Emilio Taylor

Ryuhei Uchida

Acquisition Editor

David Barnes

Development Editor

Mithil Kulkarni

Technical Editor

Shayantani Chaudhuri

Editorial Manager

Dipali Chittar

Project Manager

Patricia Weir

Project Coordinator

Sagara Naik

Indexer

Bhushan Pangaonkar

Proofreader

Chris Smith

Production Coordinator

Shantanu Zagade

Cover Designer

Shantanu Zagade

About the Author

Dr. Mark Alexander Bain first started customizing CRM systems back in the mid '90s when he was team leader for Vodafone's Cascade project—the team took the 'out-of-the-box' Clarify CRM and turned it into a radio base station planning application, complete with a workflow engine for passing jobs between the different departments involved in the planning, building, and implementation of a radio network.

Since then he's lectured at the University of Central Lancashire, and currently Mark writes articles on all things Linux and Open Source for *Linux Format*, Newsforge.com, and *Linux Journal*. He works from his home on the edge of the Lake District in the UK, where he lives with his wife, two dogs and two cats, and gets the odd visit from his sons—Michael and Simon.

SugarCRM customization, therefore, seems the obvious choice for this, his second book, since it combines Mark's knowledge of working with commercial CRM's and the Open Source philosophy.

For Mum, Donna, and Ellie.

And thanks (as always) to Simon, to Michael, and to all of the Packt Publishing team.

About the Reviewers

Andrew J. R. Whitehead

Andrew Whitehead is the lead developer at The Long Reach Corporation and has been working with the SugarCRM framework, in the form of the company's Info At Hand product, for the past three years. He has been implementing web applications both recreationally and professionally for the past fifteen years. Andrew is passionate about designing general solutions to common problems and enjoys working with new languages and technologies. He lives in Toronto, Canada and studies Linguistics part-time at the University of Toronto.

I would like to thank my family and friends, particularly Aziza, for their patience during the editing process.

Michael J. R. Whitehead

Michael Whitehead is a leading authority on the design and implementation of Customer Relationship Management (CRM) systems. Michael's experience and expertise spans a thirty year career in software architecture, design, and development as well as business management and ownership of multiple technology organizations. Among many other accomplishments Michael is the author of *Implementing SugarCRM* (from Packt Publishing) and a contributing author of the *Sugar Open Source User Guide*.

Michael is currently the founder and President of The Long Reach Corporation (`www.infoathand.com`). At The Long Reach Corp. the focus is Info At Hand™—a Customer Relationship & Business Management (CRBM) system for Small & Mid-Size Businesses, built on a base of SugarCRM Open Source. It blends a best-of-breed CRM with extended business management features for Order Management, Project & Resource tracking, Customer Service, and HR.

Ajay Gupta

Ajay Gupta has over ten years of experience in the CRM industry. He has participated in design and development of several CRM applications that have been deployed globally.

Susie Williams

Susie Williams is the Sr. Manager of Community Development at SugarCRM. Her responsibilities include development of the Sugar Adoption Program as well as management of the Forums and Sugar Forge. Prior to joining SugarCRM, Susie managed the Worldwide Sales Engineering group at WebEx Communications; she has also been involved in the CRM industry through various positions (from Engineering to Implementation Consulting) at Aurum Software/Baan Company. Susie holds a B.A. in Electrical Engineering/Computer Science from U.C. San Diego.

Aspen Olmsted

Aspen Olmsted is founder and president of Alliance Software Corporation, a CRM software solutions provider to the entertainment and non profit verticals. Aspen holds an MBA from the University of South Carolina along with certifications in PHP, MySQL, MCSD, MCSE, and many ERP applications.

Emilio Taylor

For the past one and half years, Emilio has been developing as a Project Manager for SugarCRM Development. Specializing in Microsoft SQL Server 2005 implementations, Emilio has integrated Stored Procedures, Database Triggers, Customized Modules, and Java scripting into the development lifecycle of SugarCRM deployment. Plus, in June of 2005, Emilio founded EmillionDreamz.org, a web design company utilizing Joomla! CMS for customized website development and deployment. Emilio has been living Central Florida for the past thirteen years.

Ryuhei Uchida

Ryuhei Uchida is Chief Technology Officer and a partner consultant of CareBrains, an open source consulting company and a reseller partner of SugarCRM Inc. He has broad experience of business strategies, international operations, and innovation management in the IT industry.

Prior to joining CareBrains, he had held positions in business development, product management, consulting, and sales & marketing at Fujitsu, Fujitsu Business Systems of America, J.D. Edwards, and Vitria Technology.

He is one of the enthusiastic evangelists of open-source business applications in Japan, and is leading state-of-the-art open-source communities to bring innovation into the second largest IT industry in the world.

He earned his B.A. in agriculture from Tokyo University in Japan, and also an MBA in Technology Management from Waseda University Business School.

Table of Contents

Preface

This is a developer's manual on SugarCRM. The book focuses on customizing SugarCRM. It provides you with an overview of the architecture of the application and the database. It also shows essential steps for hooking your module into the SugarCRM infrastructure.

What This Book Covers

Chapter 1—This chapter is a smooth introduction to customizing Sugar CRM.

Chapter 2—You will start to customize the SugarCRM application itself, and you will be able to add your own components in the form of module tabs and dashlets.

Chapter 3—You will learn how to modify the look and feel of SugarCRM. This chapter also shows how to add new fields to SugarCRM.

Chapter 4—This chapter looks at the interfaces, and how to use them effectively in your customizations.

Chapter 5—This chapter includes database schematic diagrams, showing the relationships between each table in the database.

Chapter 6—In this chapter we have covered complete database schematics for the SugarCRM application, providing full details on each table.

Chapter 7—You will learn in this chapter how to develop, test, and use SugarCRM in a safe environment using a development server, a test server, and a live server.

Chapter 8—You will learn to incorporate third-party modules into your site and develop your own modules from scratch.

Chapter 9—This chapter deals with developing a custom workflow within SugarCRM.

Chapter 10—You will see various techniques in this chapter for optimizing the performance of SugarCRM implementations, and a few more ways of extending the application.

Who is This Book for

The book is for PHP developers working with SugarCRM, who want to extend its capabilities. Readers should have basic knowledge of SugarCRM as the book does not provide any instructions on installation and usage.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

There are three styles for code. Code words in text are shown as follows: "We can include other contexts through the use of the `include` directive."

A block of code will be set as follows:

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

When we wish to draw your attention to a particular part of a code block, the relevant lines or items will be made bold:

```
insert into fields_meta_data
    (      id,
           name,
           label,
           help,
           custom_module,
           data_type,
           ext1,
           default_value,
           date_modified,
           mass_update
    )
```

Tips and tricks appear like this.

New terms and **important words** are introduced in a bold-type font. Words that you see on the scre]en, in menus or dialog boxes for example, appear in our text like this: "clicking the **Next** button moves you to the next screen".

Reader Feedback

Feedback from our readers is always welcome. Let us know what you think about this book, what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply drop an email to `feedback@packtpub.com`, making sure to mention the book title in the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or email `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer Support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the Example Code for the Book

Visit `http://www.packtpub.com/support`, and select this book from the list of titles to download any example code or extra resources for this book. The files available for download will then be displayed.

The downloadable files contain instructions on how to use them.

Errata

Although we have taken every care to ensure the accuracy of our contents, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in text or code—we would be grateful if you would report this to us. By doing this you can save other readers from frustration, and help to improve subsequent versions of this book. If you find any errata, report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **Submit Errata** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata added to the list of existing errata. The existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with some aspect of the book, and we will do our best to address it.

1
Stamping Your Own Brand on SugarCRM

So, there you are, you've got SugarCRM up and running, you've played with it and maybe you've added an account, created a case, or a project, or maybe even sent an email or two. However, after a while you feel that SugarCRM is very *generic* (well, it has got to be hasn't it?) but generic is probably one thing that your business is not.

You will, of course, be wondering just what customizations can be carried out in order for SugarCRM to truly reflect the way in which your organization operates. To start with, we can group the possible customizations into:

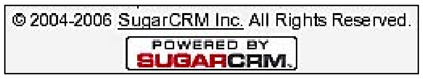
- How SugarCRM looks
- How SugarCRM works

The first two chapters of this book are concerned with tailoring the looks of the tool, and if you're happy with SugarCRM's general look and feel, then just move to Chapter 3. You can always come back here once you've finished customizing the actual operation of SugarCRM.

However, if you do continue with this chapter (or if you have just come back to it) then by the end of Chapter 1 you'll be able to make SugarCRM look the way that you want it to—so that it reflects the image of your company, and uses the terminology of your users.

Before We Start...

I expect that one of the reasons that you've chosen SugarCRM is because it's open source, and any changes that you make are covered by the SugarCRM Public License (of course, only really need to worry, if you're planning to customize and then distribute SugarCRM). With that in mind, you may be wondering if there is anything that you're *not allowed* to change. Only one. At the bottom of every screen you're expected to show the acknowledgement:



© 2004-2006 SugarCRM Inc. All Rights Reserved.

However, if you do want to read the full SugarCRM Public License you'll find it at: `http://www.sugarcrm.com/crm/SPL`

Having identified what we're required to do under the SugarCRM Public License we can move on to choosing an example company. Throughout the book we'll be looking at Penguin P.I.—Private Investigators in the dark world between Windows and Linux.

When Penguin P.I.'s founder—Pygoscelis P. Ellsworthy—started the business it was a simple, two-man setup (or rather one man, one woman since he was working with the famous femme fatale, Korora Blue). However, after a few high profile cases he needed a bigger office, and it wasn't long before he had several offices spread across the world. It was at this point that he realized that a couple of spreadsheets weren't enough to manage all of his staff and his clients. Fortunately that's where SugarCRM comes in.

And that's our starting point. Now, you'll need a server complete with Apache, PHP, MySQL, and SugarCRM. How you do this is up to you, but you could:

- Follow the SugarCRM installation instructions—you'll find these at `http://www.sugarcrm.com/wiki/index.php?title=Installation`.
- Get hold of a decent book on the subject—such as *Implementing SugarCRM* by Michael J. R. Whitehead (ISBN 978-1-904811-68-8), published by Packt Publishing.
- Get someone else to do one of the previous steps for you—Pygoscelis, of course, he didn't actually do any of this *himself*—he got his new IT person, Robby Eudyptes, to do it for him.

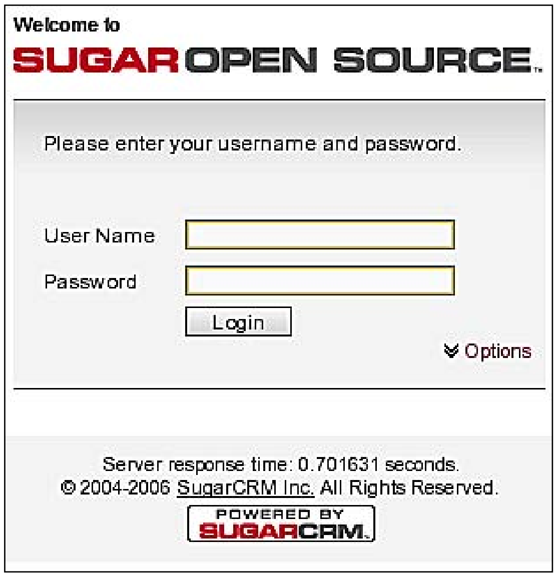
All of this assumes that you've given your servers good names—the most important job that any IT person can do. When Pygoscelis came to name all his servers he thought that he wouldn't follow the more common naming convention of choosing characters from J.R.R.Tolkien's Lord of the Rings. Instead he chose Homer's The Iliad (a fantastic book, and well worth reading if you get the chance—it shows you that human nature was just the same in 700BC as it is today). Therefore, he called his Linux servers Acamas, Aeneas, Cassandra, Hector, and Helenus. He also had one Windows server—Achilles (powerful, but with some major flaws).

With all of that done then you're ready to start customizing.

Customizing SugarCRM URL

At this stage we're not even going beyond SugarCRM's logon screen. If you've used an automatic installer (such as the very effective SpikeSource Windows Installer) then you'll be able open a browser, and type in the equivalent of `http://achilles/sugarcrm`. If you've manually installed SugarCRM on Linux then you'll probably need to type in something like: `http://hector/SugarOS-Full-4.5.0f`.

It doesn't matter if you're using Linux or Windows—in either case you'll be directed to the logon screen:

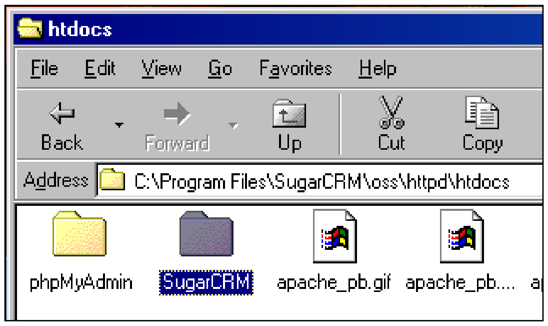


Server response time: 0.701631 seconds.
© 2004-2006 SugarCRM Inc. All Rights Reserved.

And you'll see that the URL reads: `http://achilles/sugarcrm/index.php?action=Login&module=Users` or `http://hector/SugarOS-Full-4.5.0f/index.php?action=Login&modules=Users`. There's nothing actually wrong here, but the URLs don't really *tell* you much (apart from the fact that you're using SugarCRM), so it seems a good idea to change the URL to something more meaningful—something related to the actual project that you're working on. Fortunately this is a very easy change to make.

Changing the SugarCRM URL in Windows

If you're using Windows then the first thing that you'll have to do is to find the SugarCRM directory. Obviously if you've installed everything manually then you'll know where the directory is—however, if you've used the Windows installer then it may not be quite so obvious. You may (or may not) know that the SugarCRM directory has to be in the document root for your web server, and if you've used the installer then this will probably be something like `C:\Program Files\SugarCRM\oss\httpd\htdocs`:



All you have to do is rename the directory (for instance our friend Pygoscelis might want it renamed `penguin_pi` for the Penguin P.I. organization), and then type the new URL into your browser (and for the Penguin P.I. organization this would be `http://achilles/penguin_pi).`

Changing the SugarCRM URL in Linux

In this instance Linux isn't too different from Windows—all you need to do is find the SugarCRM directory (again it will be in your web server's document root), and then rename it appropriately:

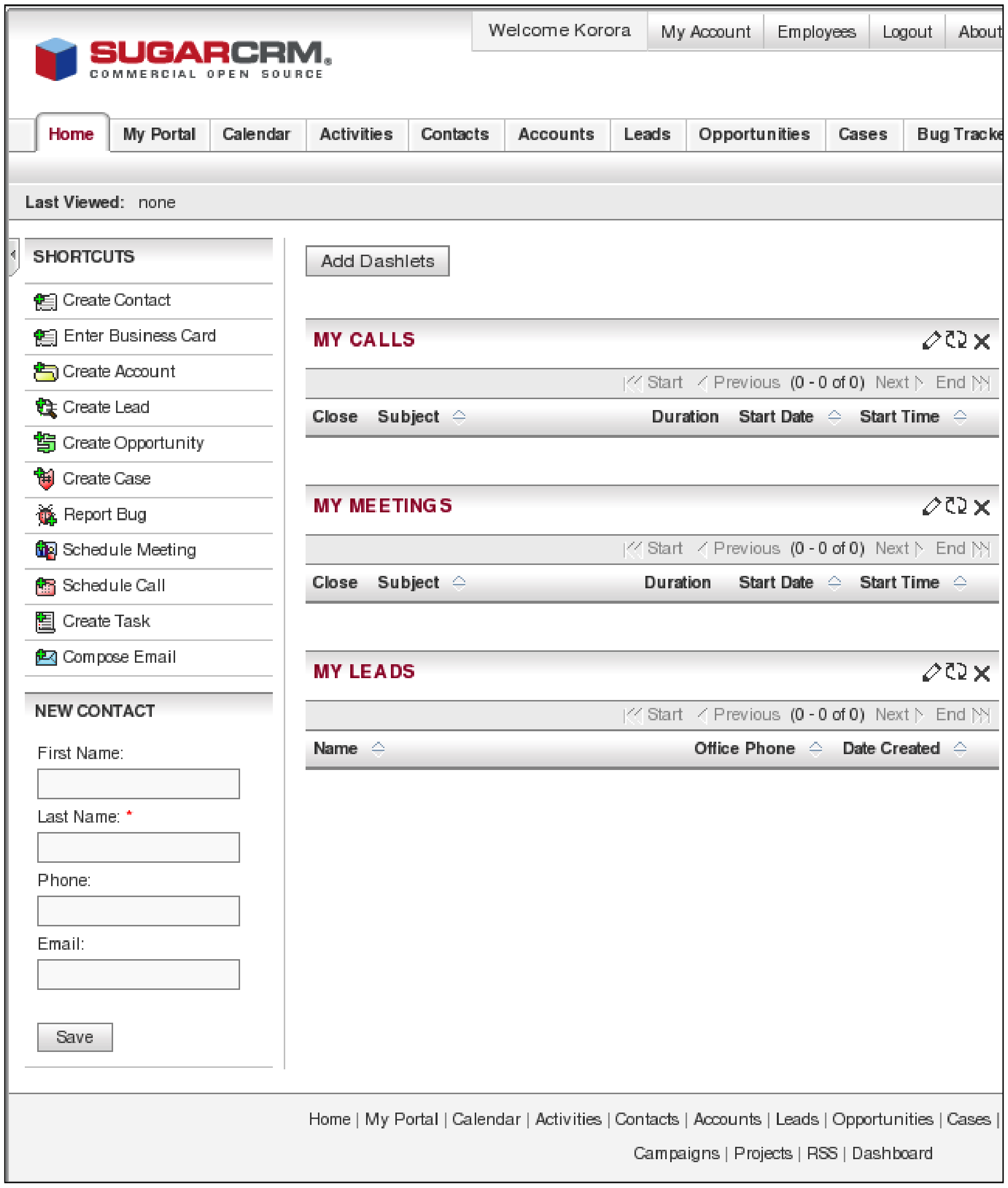
```
mv SugarOS-Full-4.5.0f penguin_pi
```

With that done the new URL will work:
(e.g. `http://hector/penguin_pi/index.php?action=index&module=Home`)

Having sorted out SugarCRM's URL we can turn our attention to the rest of the screen.

Customizing SugarCRM Tabs

Your users are now able to access your SugarCRM implementation via a URL that means something to them. So, their first view of the system will be something like:


SUGARCRM. COMMERCIAL OPEN SOURCE
Welcome Korora | My Account | Employees | Logout | About
Home | My Portal | Calendar | Activities | Contacts | Accounts | Leads | Opportunities | Cases | Bug Tracke
Last Viewed: none
SHORTCUTS
Create Contact
Enter Business Card
Create Account
Create Lead
Create Opportunity
Create Case
Report Bug
Schedule Meeting
Schedule Call
Create Task
Compose Email
NEW CONTACT
First Name:
Last Name: *
Phone:
Email:
Save
Add Dashlets
MY CALLS
Start Previous (0 - 0 of 0) Next End
Close Subject Duration Start Date Start Time
MY MEETINGS
Start Previous (0 - 0 of 0) Next End
Close Subject Duration Start Date Start Time
MY LEADS
Start Previous (0 - 0 of 0) Next End
Name Office Phone Date Created
Home | My Portal | Calendar | Activities | Contacts | Accounts | Leads | Opportunities | Cases |
Campaigns | Projects | RSS | Dashboard

If we imagine Korora Blue looking at this for the first time, Korora would immediately realize that she can navigate around the system by using the SugarCRM Tabs. And she will recognize a lot of them (for example **Emails** and **Calendar**), but a lot will be new to her (for example **My Portal** and **Opportunities**).

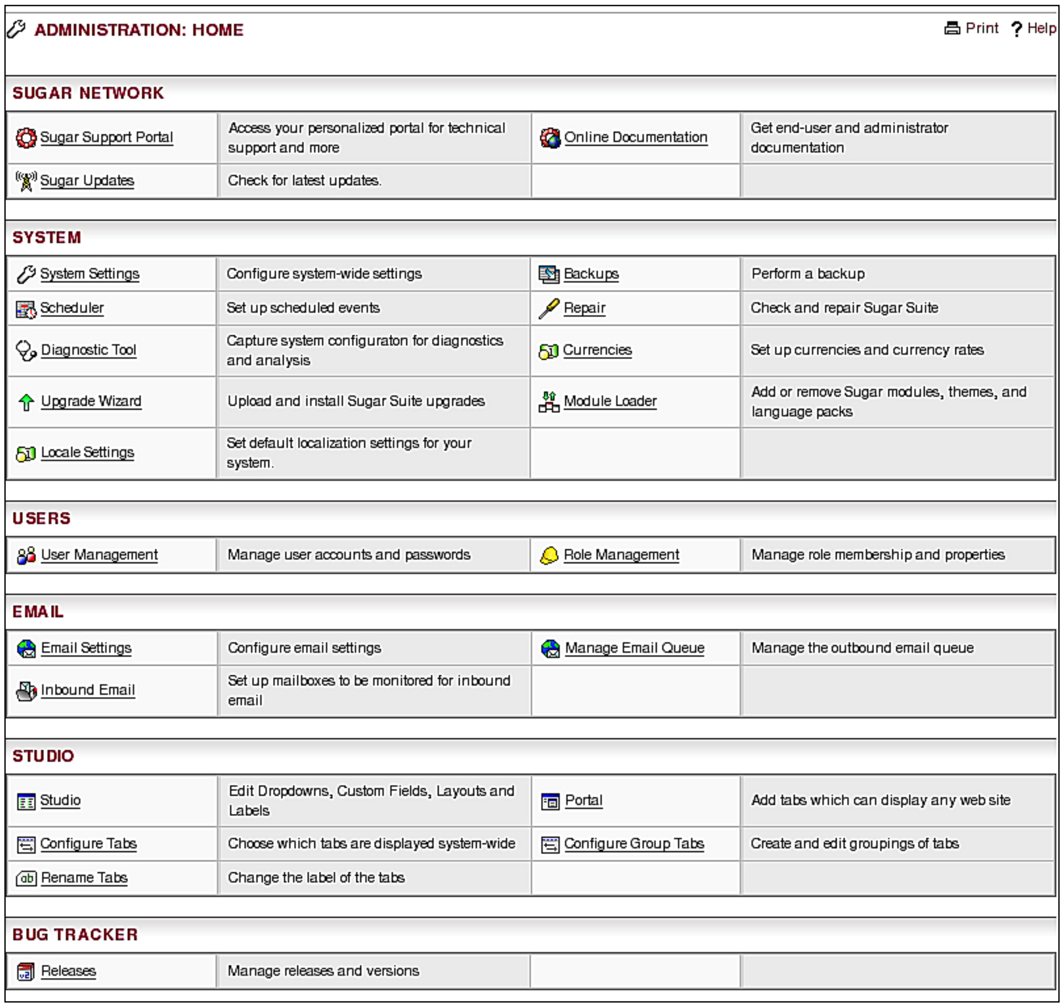
So you've got two choices:

- Re-train your staff so that they map their work to SugarCRM terminology.
- Re-name the SugarCRM tabs to the language of your organization.

Guess what we're going to start with.

Re-name the SugarCRM Tabs

To start with you need to log on to the administrator account (by default this will be by using the user name 'admin'), and then go to the **ADMINISTRATION: HOME** screen:



ADMINISTRATION: HOME

Print ? Help

SUGAR NETWORK

Sugar Support Portal	Access your personalized portal for technical support and more	Online Documentation	Get end-user and administrator documentation
Sugar Updates	Check for latest updates.		

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 configuraton 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.		

USERS

User Management	Manage user accounts and passwords	Role Management	Manage role membership and properties

EMAIL

Email Settings	Configure email settings	Manage Email Queue	Manage the outbound email queue
Inbound Email	Set up mailboxes to be monitored for inbound email		

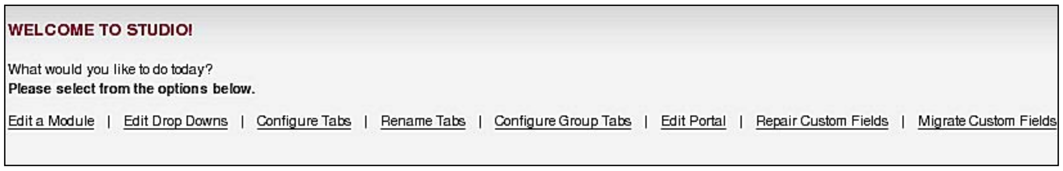
STUDIO

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		

BUG TRACKER

Releases	Manage releases and versions		

Here you can either click **Rename Tabs**, or you can go to the **Studio** screen first:

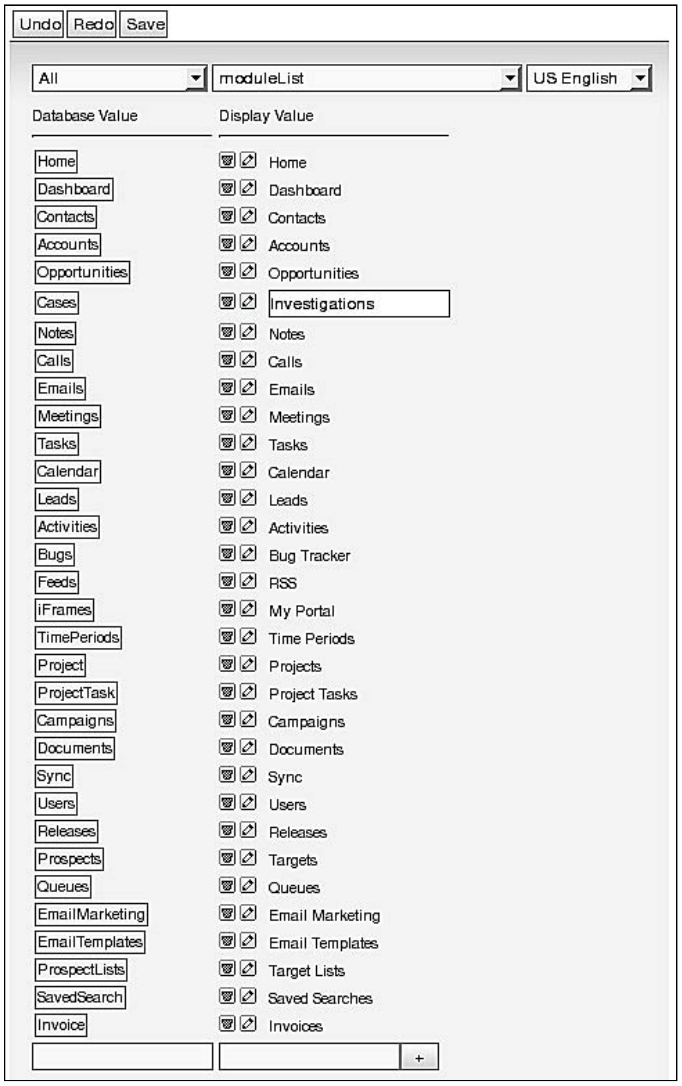


WELCOME TO STUDIO!

What would you like to do today?
Please select from the options below.

Edit a Module | Edit Drop Downs | Configure Tabs | Rename Tabs | Configure Group Tabs | Edit Portal | Repair Custom Fields | Migrate Custom Fields

Now you can rename any of the SugarCRM tabs to something more appropriate to your organization:



Undo | Redo | Save

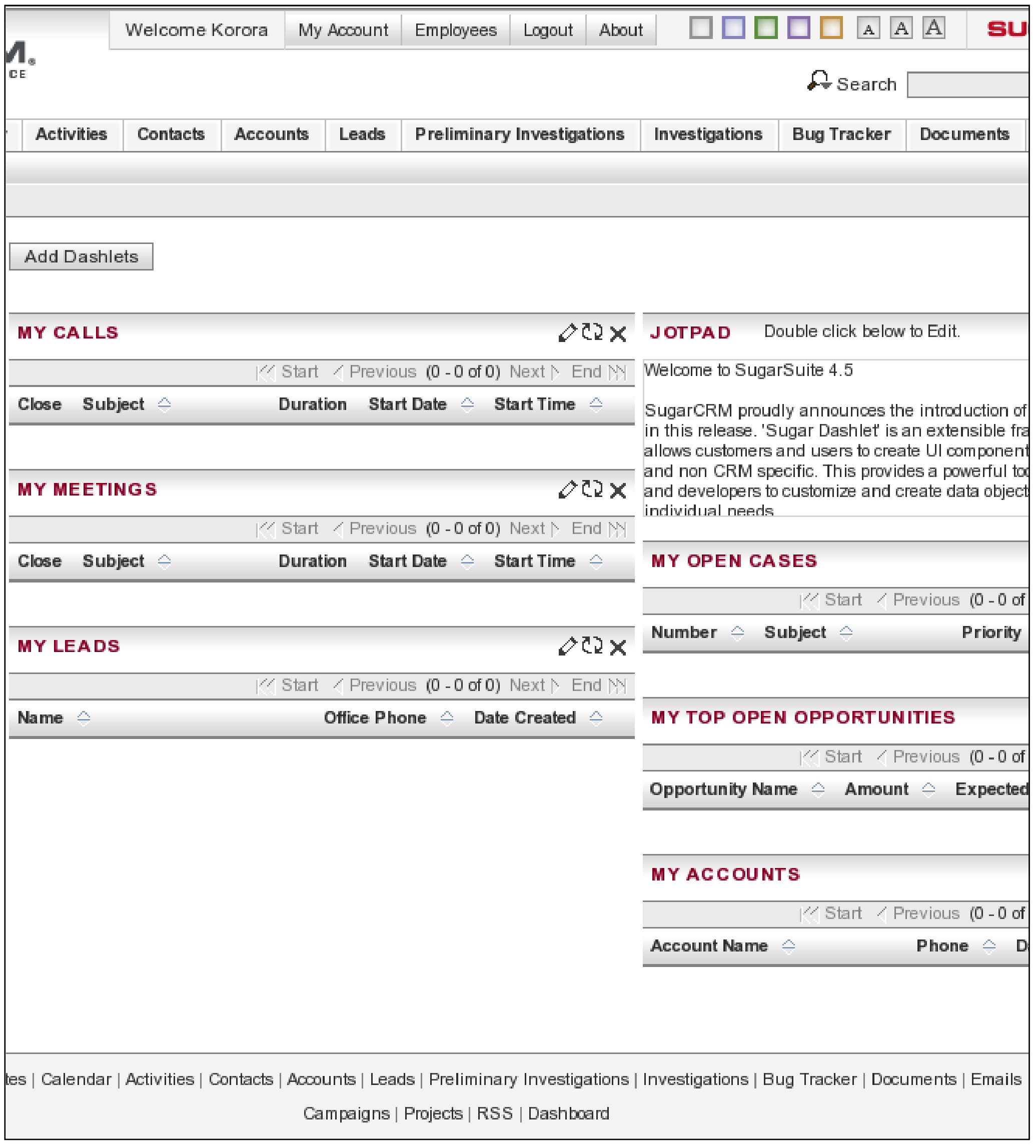
All | moduleList | US English

Database Value	Display Value
Home	Home
Dashboard	Dashboard
Contacts	Contacts
Accounts	Accounts
Opportunities	Opportunities
Cases	Investigations
Notes	Notes
Calls	Calls
Emails	Emails
Meetings	Meetings
Tasks	Tasks
Calendar	Calendar
Leads	Leads
Activities	Activities
Bugs	Bug Tracker
Feeds	RSS
iFrames	My Portal
TimePeriods	Time Periods
Project	Projects
ProjectTask	Project Tasks
Campaigns	Campaigns
Documents	Documents
Sync	Sync
Users	Users
Releases	Releases
Prospects	Targets
Queues	Queues
EmailMarketing	Email Marketing
EmailTemplates	Email Templates
ProspectLists	Target Lists
SavedSearch	Saved Searches
Invoice	Invoices

Once you've saved your changes your users will see something that relates to them. For example, here we've renamed:

- My Portal => **Web Sites**
- Opportunities => **Preliminary Investigations**
- Cases => **Investigations**

And now Korora would have tabs that she'd immediately understand:



MY CALLS

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

Close	Subject	Duration	Start Date	Start Time

MY MEETINGS

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

Close	Subject	Duration	Start Date	Start Time

MY LEADS

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

Name	Office Phone	Date Created

JOTPAD Double click below to Edit.

Welcome to SugarSuite 4.5

SugarCRM proudly announces the introduction of in this release. 'Sugar Dashlet' is an extensible fra allows customers and users to create UI component and non CRM specific. This provides a powerful to and developers to customize and create data object individual needs

MY OPEN CASES

Start | Previous | (0 - 0 of

Number	Subject	Priority

MY TOP OPEN OPPORTUNITIES

Start | Previous | (0 - 0 of

Opportunity Name	Amount	Expected

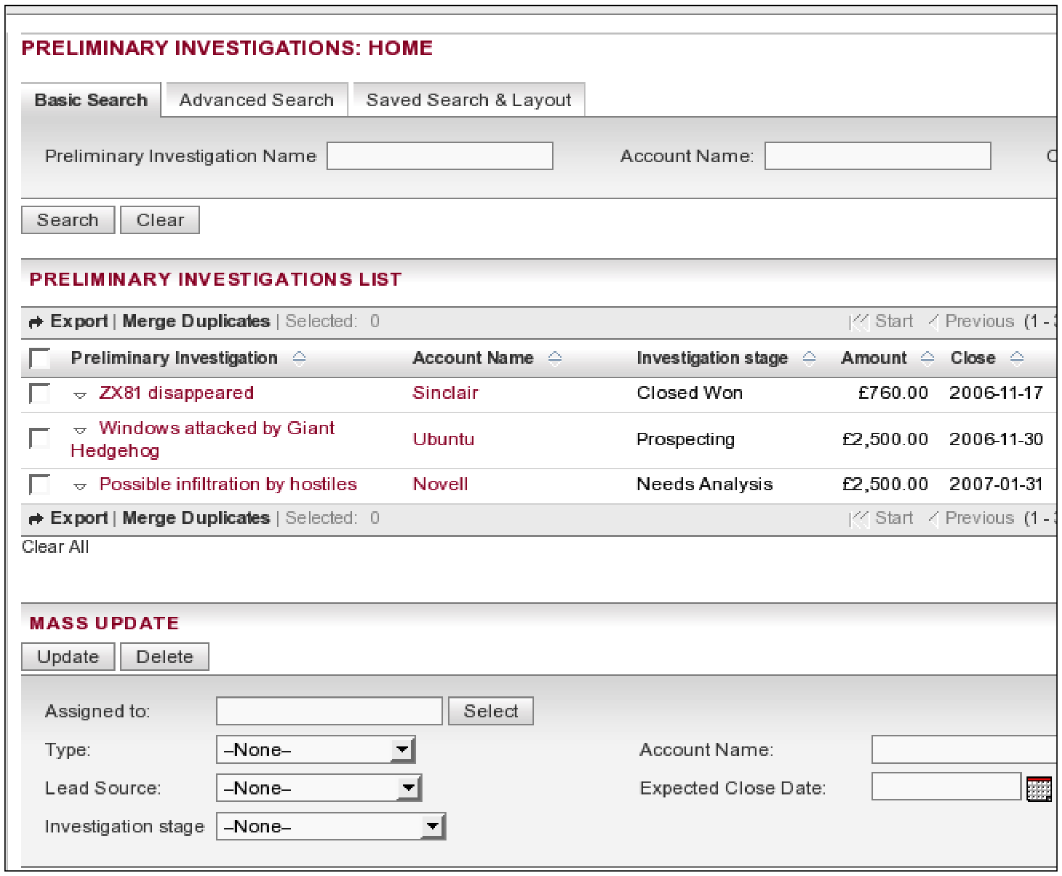
MY ACCOUNTS

Start | Previous | (0 - 0 of

Account Name	Phone	D

tes | Calendar | Activities | Contacts | Accounts | Leads | Preliminary Investigations | Investigations | Bug Tracker | Documents | Emails
Campaigns | Projects | RSS | Dashboard

However, the screen still contains a large amount of the default text (which has nothing to do with Korora's job), and this would be even more apparent if Korora were to click on one of the newly named tabs such as **Preliminary Investigations**:



PRELIMINARY INVESTIGATIONS: HOME

Preliminary Investigation Name

Account Name:

PRELIMINARY INVESTIGATIONS LIST

Export | Merge Duplicates | Selected: 0 — Start Previous (1 -

Preliminary Investigation	Account Name	Investigation stage	Amount	Close
ZX81 disappeared	Sinclair	Closed Won	£760.00	2006-11-17
Windows attacked by Giant Hedgehog	Ubuntu	Prospecting	£2,500.00	2006-11-30
Possible infiltration by hostiles	Novell	Needs Analysis	£2,500.00	2007-01-31

Export | Merge Duplicates | Selected: 0 — Start Previous (1 -

Clear All

MASS UPDATE

Assigned to:			
Type:	–None–	Account Name:	
Lead Source:	–None–	Expected Close Date:	
Investigation stage	–None–		

Still not exactly 'custom' is it? And I'm sure that you've had a look around the administration screens to change other details on the screen, but not had any success. That's because any further changes need to be done outside the SugarCRM application, in the SugarCRM `custom` directory.

The SugarCRM Custom Directory

We'll do most of our work in the SugarCRM `custom` directory—in fact we've been using it already. You'll find it in your SugarCRM folder on your web server, and if you take a look there, you will see that there's already a file in the folder—when we saved the changes to the tab names SugarCRM created `custom/include/language/en_us.lang.php`. Examination of the file will reveal the changes made:

```
<?php
$app_list_strings['moduleList'] = array (
  'Home' => 'Home',
  'Dashboard' => 'Dashboard',
  'Contacts' => 'Contacts',
  'Accounts' => 'Accounts',
  'Opportunities' => 'Preliminary Investigations',
  'Cases' => 'Investigations',
  'Notes' => 'Notes',
  'Calls' => 'Surveillance',
  'Emails' => 'Emails',
  'Meetings' => 'Meetings',
  'Tasks' => 'Tasks',
  'Calendar' => 'Calendar',
  'Leads' => 'Leads',
  'Activities' => 'Activities',
  'Bugs' => 'Bug Tracker',
  'Feeds' => 'RSS',
  'iFrames' => 'Web Sites',
  'TimePeriods' => 'Time Periods',
  'Project' => 'Projects',
  'ProjectTask' => 'Project Tasks',
  'Campaigns' => 'Campaigns',
  'Documents' => 'Documents',
  'Sync' => 'Sync',
  'Users' => 'Users',
  'Releases' => 'Releases',
  'Prospects' => 'Targets',
  'Queues' => 'Queues',
  'EmailMarketing' => 'Email Marketing',
  'EmailTemplates' => 'Email Templates',
  'ProspectLists' => 'Target Lists',
  'SavedSearch' => 'Saved Searches',
  'Invoice' => 'Invoices',
);
?>
```

And that should give you a clue as to how language customization works within SugarCRM.

Customizing the Text within SugarCRM Tab Screens

We're not going to look at *every* tab screen—once you know how to modify one then you can make changes as required. Since we've already been looking at the **Opportunities** screen let's carry on with that—but don't forget that the techniques we use will apply on any other screen that you want to customize.

We've seen that we use a file called `en_us.lang.php` in order to make our own changes to the text displayed on the SugarCRM screen, and you may well reason that we'll use this file to make changes to *any* text on the screen. Well, you're nearly right. We *will* use an `en_us.lang.php` file—*but not the same one*. This time we're going to use a file called `custom/modules/Opportunities/language/en_us.lang.php` (remember that `custom` is in your SugarCRM directory—and you'll need to create the sub-directories and the file itself).

The way in which this works is quite simple—SugarCRM will look at its default language files for the text to be displayed. However, if you've got your own definitions in a custom file then it will use those instead. Of course, next you'll need the default definitions so that you know what to put into your custom language file.

You may have already worked this out (but in case you haven't) the default language file for 'Opportunities' is `modules/Opportunities/language/en_us.lang.php`. If you examine the file then you'll find that it contains similar data to the `en_us.lang.php` that we've already seen:

```
$mod_strings = array (
  'LBL_MODULE_NAME' => 'Opportunities',
  'LBL_MODULE_TITLE' => 'Opportunities: Home',
  'LBL_SEARCH_FORM_TITLE' => 'Opportunity Search',
  'LBL_VIEW_FORM_TITLE' => 'Opportunity View',
  'LBL_LIST_FORM_TITLE' => 'Opportunity List',
  'LBL_OPPORTUNITY_NAME' => 'Opportunity Name:',
```

As you can see, it contains the text to be displayed in the tab screen labels, and you may be wondering why we don't just edit this file instead. There are a few reasons:

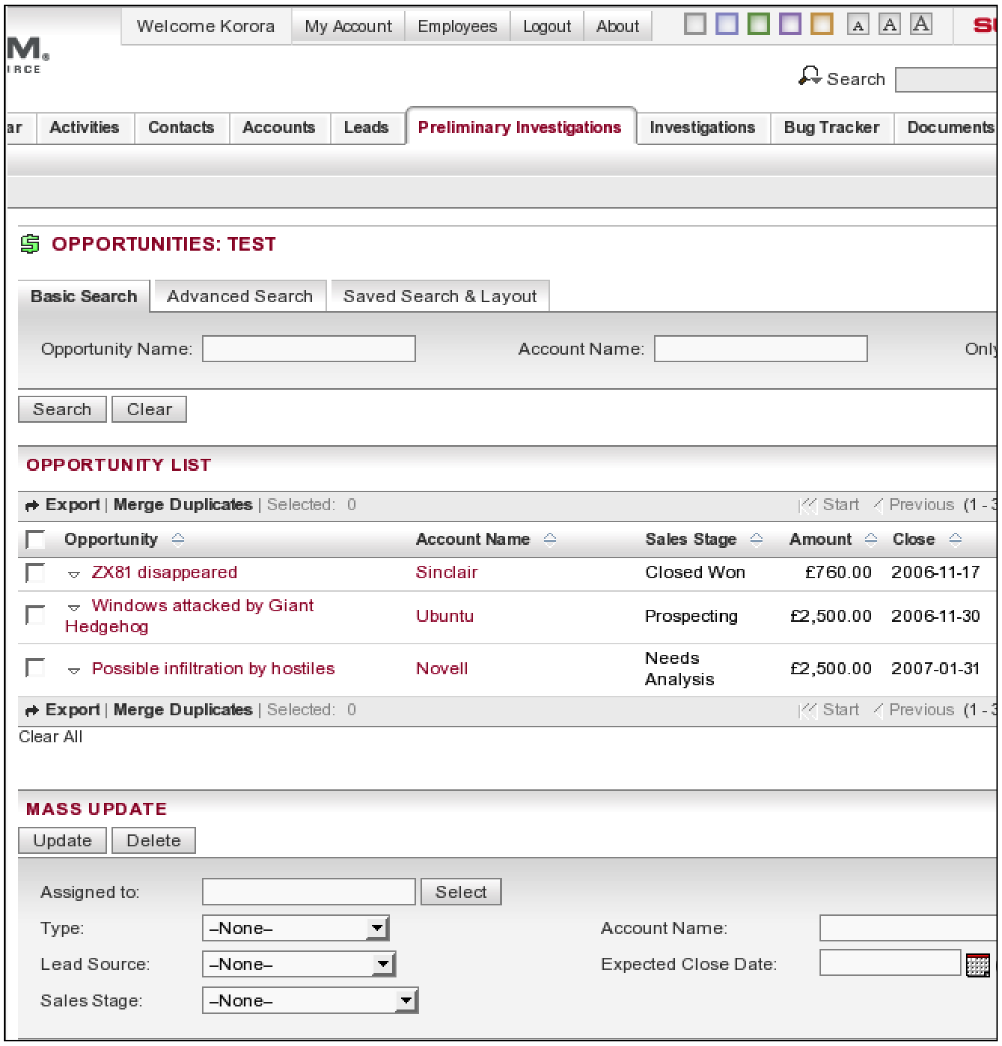
- This gives you a nice fall-back position—if everything goes wrong then you just need to delete the custom files to return to the defaults.
- There are some lines in the default file that *must not* be changed—the application may get *very* upset with you if you do change them.
- You only need to worry about maintaining your own changes and not the whole file.
- If you use the custom files then your changes remain isolated from the core functionality while allowing you to make the customizations that you need.

So, the next stage is to edit the `custom/modules/Opportunities/language/en_us.lang.php` file so that it contains the text that you actually want to be displayed on the **Opportunities** screen:

```
<?php
$opp_single = 'Preliminary Investigation';
$opp_title = $opp_single . 's';
$mod_strings['LBL_MODULE_NAME'] = $opp_title;
$mod_strings['LBL_MODULE_TITLE'] = $opp_title . ': Home';
$mod_strings['LBL_LIST_FORM_TITLE'] = $opp_title . ' List';
$mod_strings['LBL_OPPORTUNITY_NAME'] = $opp_single . ' Name';
$mod_strings['LBL_NEW_FORM_TITLE'] = 'Create ' . $opp_title;
$mod_strings['LNK_NEW_OPPORTUNITY'] = 'Create ' . $opp_title;
$mod_strings['LNK_OPPORTUNITY_LIST'] = $opp_title;
$mod_strings['LBL_TOP_OPPORTUNITIES'] = 'My Top Open ' . $opp_title;
$mod_strings['LBL_DEFAULT_SUBPANEL_TITLE'] = $opp_title;
?>
```

Notice that we haven't just hard coded all of the label details—this time we're using some variables (`$opp_single` and `$opp_title`). This means, of course, that if you decide to change the name of the module then you only need to change one line of code to update all the labels.

Once you've saved the file you can see the effects immediately by going to the **Opportunities** tab:



OPPORTUNITIES: TEST

Basic Search | Advanced Search | Saved Search & Layout

Opportunity Name: Account Name:

Search Clear

OPPORTUNITY LIST

Export | Merge Duplicates | Selected: 0

Opportunity	Account Name	Sales Stage	Amount	Close
ZX81 disappeared	Sinclair	Closed Won	£760.00	2006-11-17
Windows attacked by Giant Hedgehog	Ubuntu	Prospecting	£2,500.00	2006-11-30
Possible infiltration by hostiles	Novell	Needs Analysis	£2,500.00	2007-01-31

Export | Merge Duplicates | Selected: 0

Clear All

MASS UPDATE

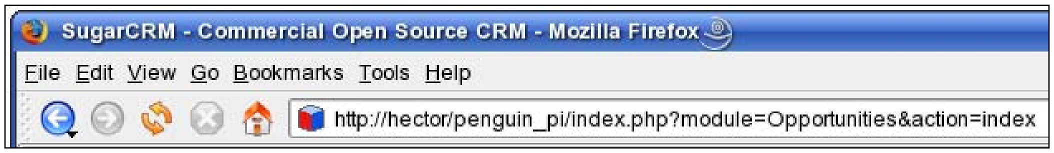
Update Delete

Assigned to:			
Type:	–None–	Account Name:	
Lead Source:	–None–	Expected Close Date:	
Sales Stage:	–None–		

Obviously you need to repeat this process for each of the tab screens, but before long you won't have a generic CRM system—you will have a CRM system that uses the same terminology as your organization.

Changing the Browser Title

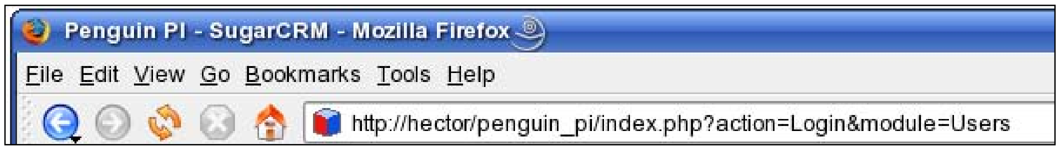
We've now modified the SugarCRM screens so that any users recognize the language, but we can still do more to reflect your company's brand. The first thing to look at is the browser title. At the moment it will look like:


SugarCRM - Commercial Open Source CRM - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://hector/penguin_pi/index.php?module=Opportunities&action=index

While we're not trying to cover up the fact that we're using SugarCRM, we are trying to stamp our own brand onto the application. To do this we'll need to edit the `custom/include/language/en_us.lang.php` file again, and add a line:

```
$app_strings['LBL_BROWSER_TITLE'] = 'Penguin PI - SugarCRM';
```

Now, you need to refresh the browser:


Penguin PI - SugarCRM - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://hector/penguin_pi/index.php?action=Login&module=Users

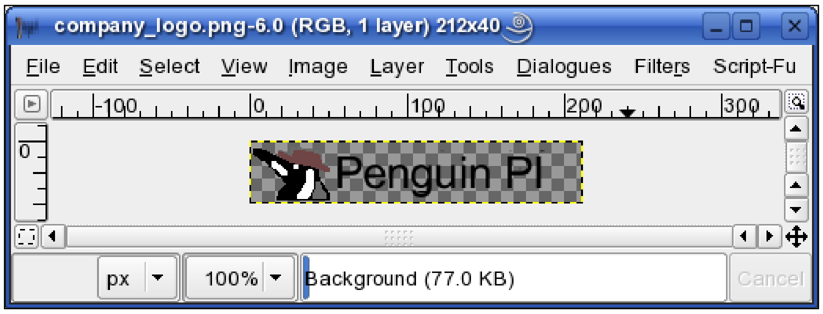
With the title set correctly, we can think about one of the most important parts of a brand—the company logo.

Adding a Company Logo

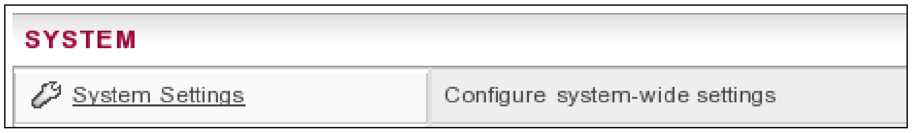
You may already have a logo that you're wanting to use with your SugarCRM installation. However, whether you're going to use an existing one, or you're creating a completely new one, there are a couple of things that you need to keep in mind:

- To be consistent with the SugarCRM layout your company logo must be 220x40 pixels.
- Your company logo should have a transparent background—so that it can work with different SugarCRM themes.

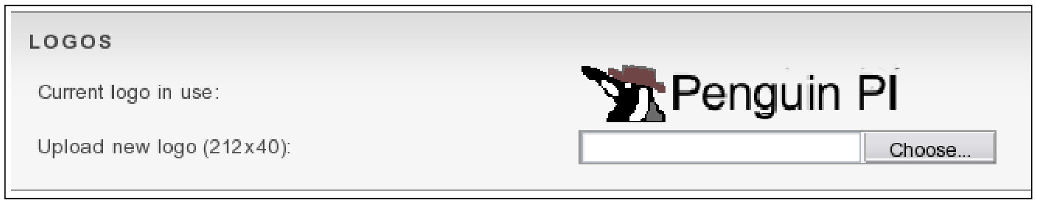
Then it's just a matter of creating the image with an appropriate piece of software; for example, you could use Gimp (Gimp is the GNU Image Manipulation Program, and if it's not already on your computer then you can get it from `http://www.gimp.org`).


company_logo.png-6.0 (RGB, 1 layer) 212x40
File Edit Select View Image Layer Tools Dialogues Filters Script-Fu
Penguin PI
px
100%
Background (77.0 KB)
Cancel

Once you have saved your image you'll want to apply it to your SugarCRM implementation—so, it's back to the main admininistration screen, where you need to click on **System Settings**:



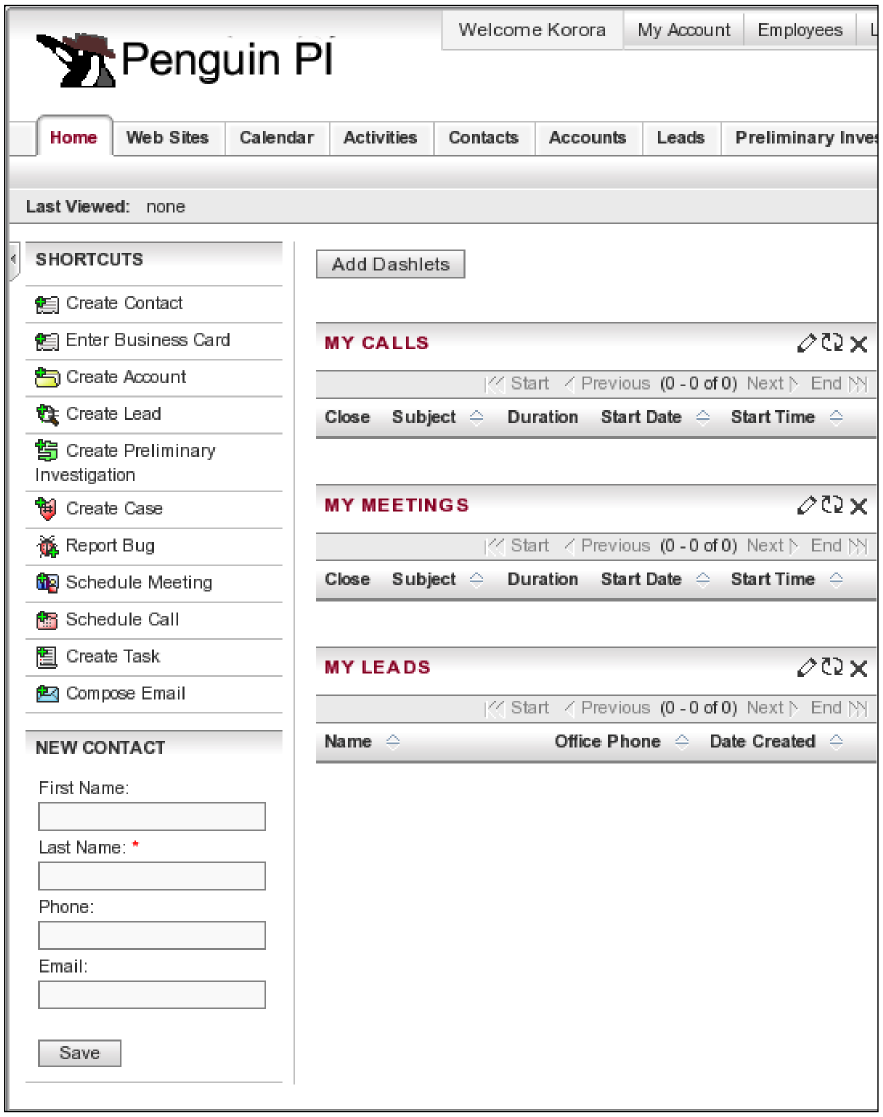
And then you can upload the logo:



With that done you've got a SugarCRM implementation with:

- Tabs with titles that mean something to people in your organization
- Screens that use the correct terminology for your users
- A logo that illustrates the implementation belongs to your company

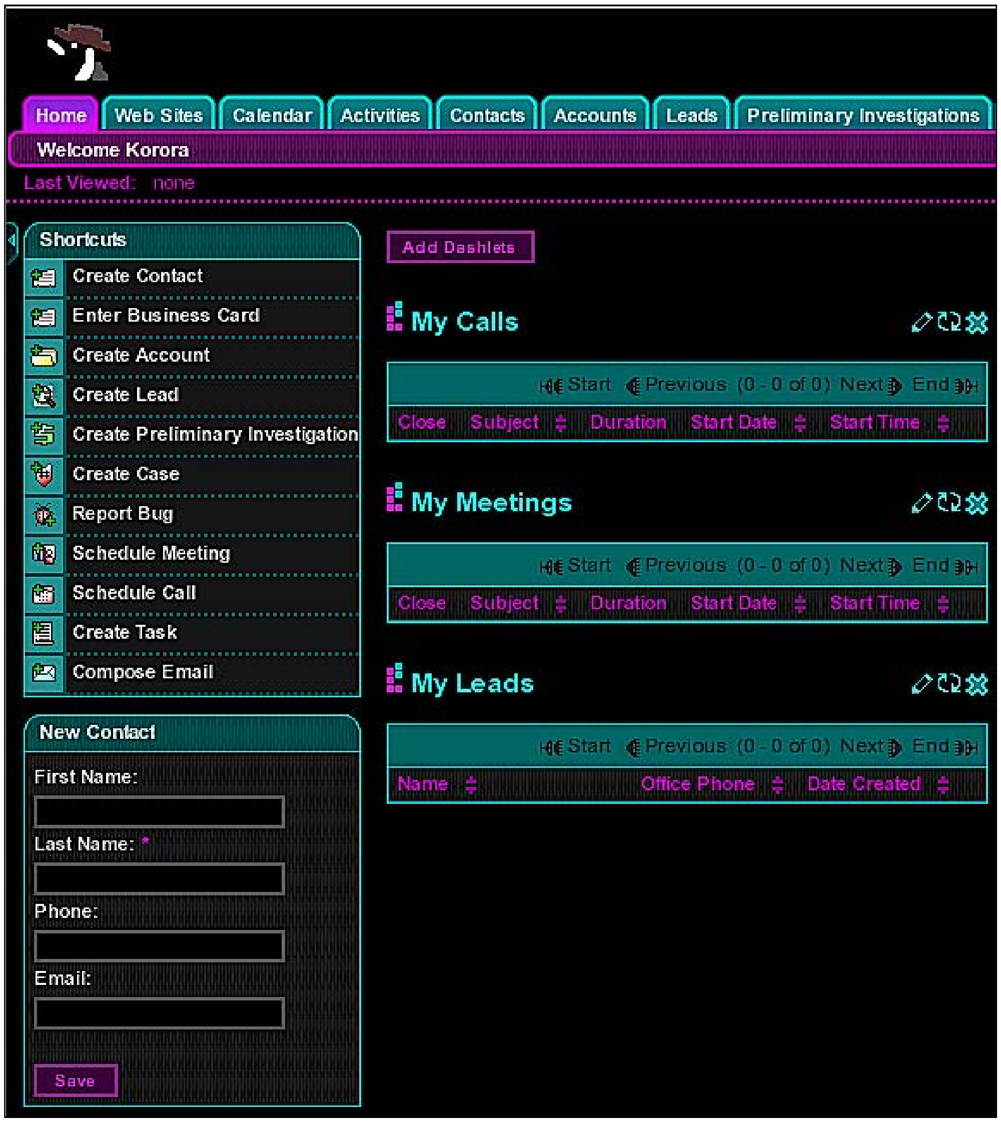
Now your users can log on and see your company's SugarCRM application, and they'll know that it's something designed for them, and (hopefully) they'll be happy to use it:



At this stage you may decide that you don't need to do any further customizations to the look and feel of SugarCRM, and if that's the case then it's time to move on to Chapter 2 where we start looking at customizing the application contents. However, before you do that it's worth spending a little time considering another aspect of the SugarCRM front-end user experience—Themes.

Customizing SugarCRM Themes

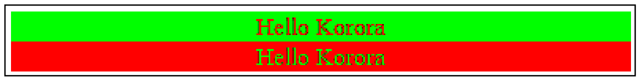
We've spent time getting the general look and feel right, but (as you probably already know) your users can already customize SugarCRM by using themes—sometimes with quite startling results:



If you're striving to create an application with your own brand then some of the themes may not work with your view of how things should look. If that's the case then you can:

- Create your own theme(s)
- Limit the themes that can be accessed by your users

So, that's what we'll look at next. However, before we start have a look at this:



You should see two lines, each saying **Hello Korora**. However, if any of your users are color blind then they may not be able to read the information (personally, I've got a red-green deficiency that means that I can read both lines but they give me a blinding headache).

The point is, of course, if you are going to be creating your own color scheme then be aware of the effect that it will have on your users. The same goes for the size of fonts that you want to use.

Creating a New Theme

You'll find that the easiest way to create a new theme is simply to copy an existing one and then modify it. Start by looking for the `themes` directory—a sub-directory of your main SugarCRM folder. On Linux you can achieve this by typing something like:

```
cd /var/www/htdocs/penguin_pi/themes
cp -R SugarClassic PenguinPI
```

or on Windows:

```
cd "C:\Program Files\SugarCRM\oss\httpd\htdocs\SugarCRM\themes"
copy sugarclassic penguinpi
```

You now need to move to your new directory, edit the `config.php` file, and add the theme name and description:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
$theme_name = "Penguin PI";
$theme_description = "Penguin PI theme";
```

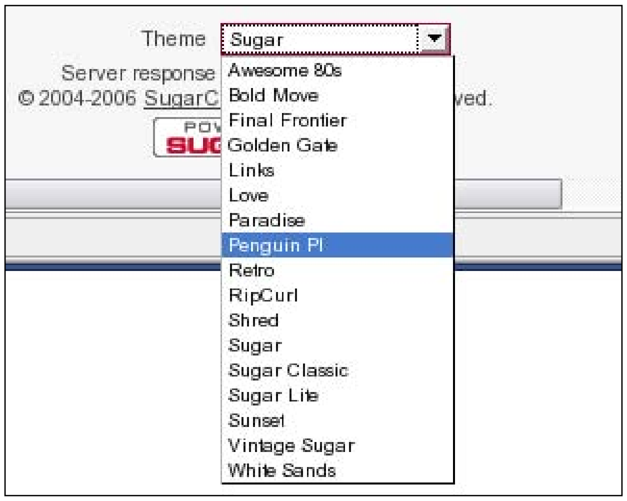
```
$version_compatibility = 3.0;
$max_tabs = 12;
$png_support = false;
?>
```

Also in the directory you'll find all of the files needed for formatting the theme itself. For now we'll just look at a very simple modification—we'll change the theme so that any link is highlighted in yellow when the mouse pointer is placed over it.

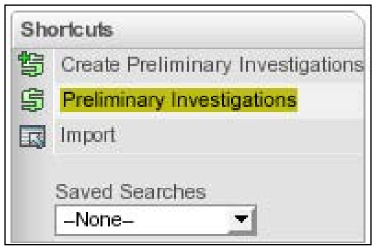
All you need to do is edit the `style.css` file in your new theme folder, and make the following change:

```
    a:hover {
                color: #666666;
                text-decoration: underline;
                background-color:yellow; //highlight the link
          }
```

You'll probably need to restart your browser, but once you have done so, then you'll be able to select your new theme from the drop-down list at the bottom of the SugarCRM screen:



This new theme will appear exactly the same as the **Sugar** theme, except that links will be highlighted in yellow when you place the mouse pointer over them:



Of course, when you've finished creating the theme that you want, then you can start thinking about removing the themes that you don't want.

Removing a Theme

Some themes are incompatible with your company image. So you go to the themes directory and delete the directory containing the offending theme.

Next time the browser is started the theme will be absent—and don't worry, if someone was using the theme that you've deleted then they move to to the next theme on the list by default in their next session.

Summary

Chapter 1 has been a nice, gentle introduction into the world of customizing SugarCRM. In the course of the chapter you learned how to customize the look and feel of the SugarCRM screen—without affecting any of the functionality.

We have changed the SugarCRM URL to something that relates to the project. Installation of the tool will give a name to the directory by default; this can be changed for customization. However, do not forget the directory name is the main part of the SugarCRM URL.

Customizing tab names will make them more specific catering to the need of the company. You can also get creative with the browser title. Also you can change the text on the screen to appropriate terminology to suit the company requirements. You can play around with SugarCRM themes and add the company logo.

After all this, you should be able to produce a SugarCRM implementation that has the appropriate look and feel for the project that you're working on—one that would make Pygoscelis proud.

In Chapter 2 we look at how we can start customizing the application content, so that you are able to give each user exactly what they need—at their fingertips.

2
Customizing the SugarCRM Application Content

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.

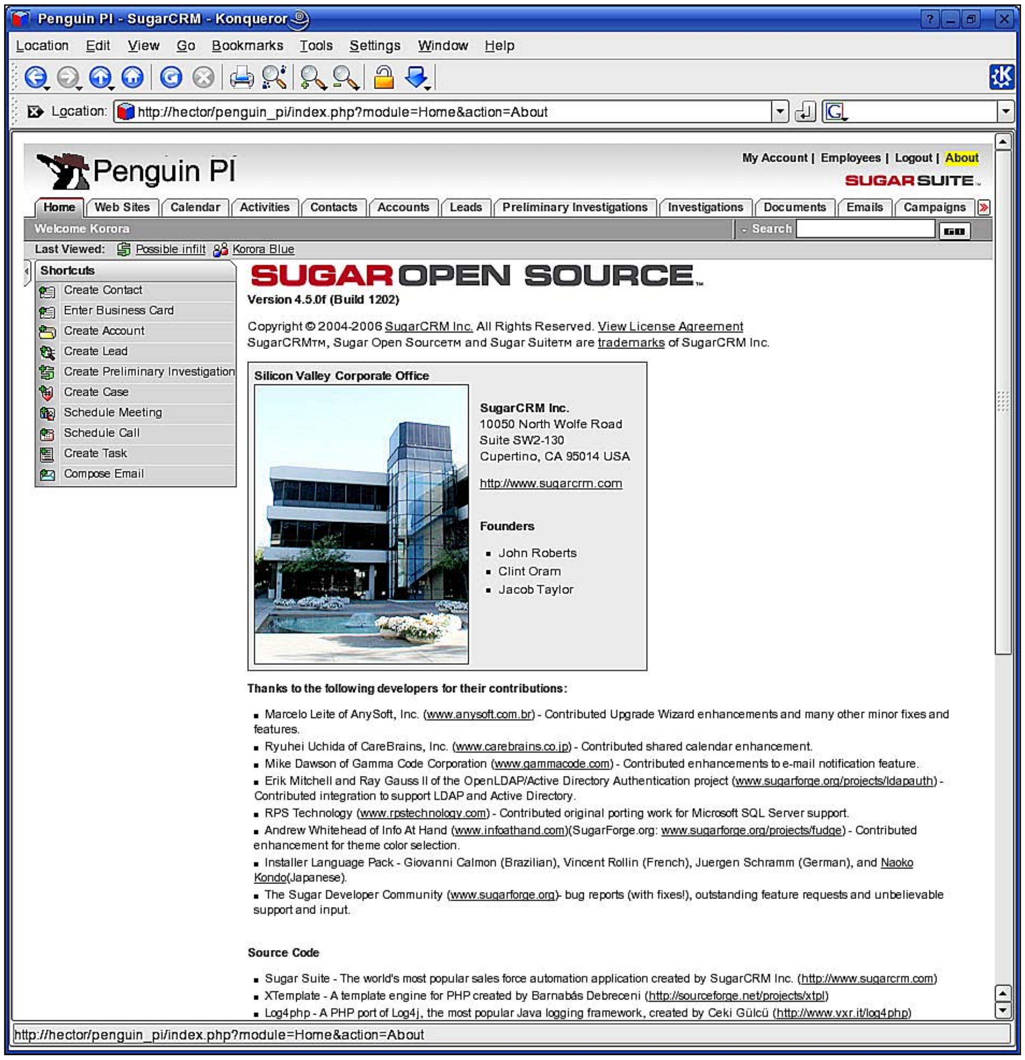
In this chapter we're going to start with adding our own functionality into SugarCRM. Nothing too elaborate, and we won't touch any of the core functionalities (yet). We'll just see how easy it is to add your own tab screens and **Dashlets**—your own GUI components.

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.

Changing the About Screen

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


Penguin PI - SugarCRM - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: http://hector/penguin_pi/index.php?module=Home&action=About
Penguin PI
My Account | Employees | Logout | About
SUGARSUITE
Home
Web Sites
Calendar
Activities
Contacts
Accounts
Leads
Preliminary Investigations
Investigations
Documents
Emails
Campaigns
Welcome Korora
Search
Last Viewed: Possible infilt Korora Blue
Shortcuts
Create Contact
Enter Business Card
Create Account
Create Lead
Create Preliminary Investigation
Create Case
Schedule Meeting
Schedule Call
Create Task
Compose Email
http://hector/penguin_pi/index.php?module=Home&action=About

SUGAR OPEN SOURCE

Version 4.5.0f (Build 1202)

Copyright © 2004-2006 SugarCRM Inc. All Rights Reserved. View License Agreement
SugarCRMTM, Sugar Open SourceTM and Sugar SuiteTM are trademarks of SugarCRM Inc.

Silicon Valley Corporate Office

SugarCRM Inc.
10050 North Wolfe Road
Suite SW2-130
Cupertino, CA 95014 USA

http://www.sugarcrm.com

Founders

- John Roberts
- Clint Oram
- Jacob Taylor

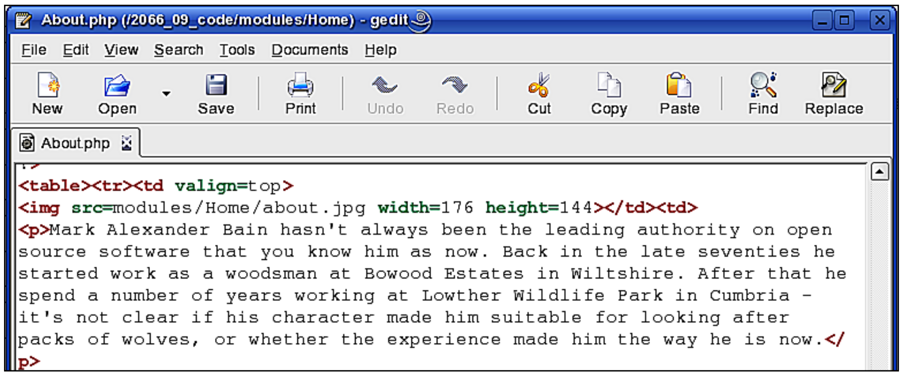
Thanks to the following developers for their contributions:

- Marcelo Leite of AnySoft, Inc. (www.anysoft.com.br) - Contributed Upgrade Wizard enhancements and many other minor fixes and features.
- Ryuhei Uchida of CareBrains, Inc. (www.carebrains.co.jp) - Contributed shared calendar enhancement.
- Mike Dawson of Gamma Code Corporation (www.gammacode.com) - Contributed enhancements to e-mail notification feature.
- Erik Mitchell and Ray Gauss II of the OpenLDAP/Active Directory Authentication project (www.sugarforge.org/projects/ldapauth) - Contributed integration to support LDAP and Active Directory.
- RPS Technology (www.rpstechnology.com) - Contributed original porting work for Microsoft SQL Server support.
- Andrew Whitehead of Info At Hand (www.infoathand.com)(SugarForge.org: www.sugarforge.org/projects/fudge) - Contributed enhancement for theme color selection.
- Installer Language Pack - Giovanni Calmon (Brazilian), Vincent Rollin (French), Juergen Schramm (German), and Naoko Kondo(Japanese).
- The Sugar Developer Community (www.sugarforge.org)- bug reports (with fixes!), outstanding feature requests and unbelievable support and input.

Source Code

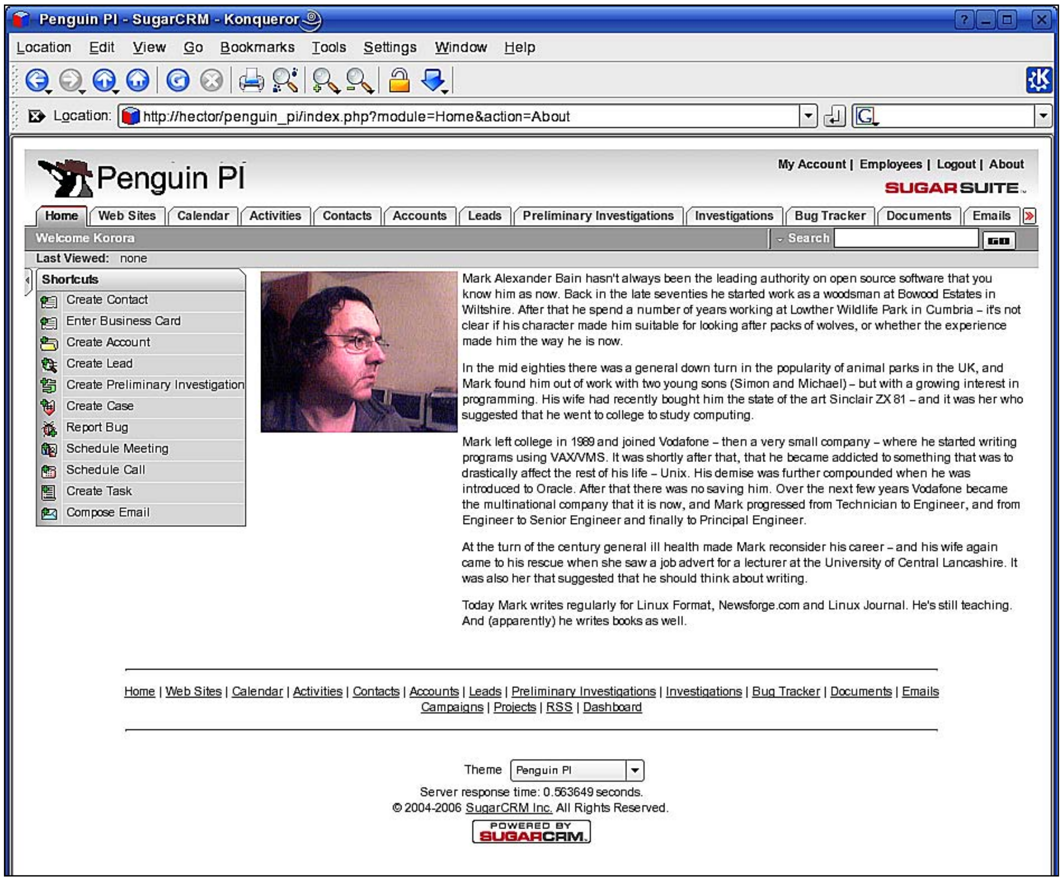
- Sugar Suite - The world's most popular sales force automation application created by SugarCRM Inc. (http://www.sugarcrm.com)
- XTemplate - A template engine for PHP created by Barnabás Debreceni (http://sourceforge.net/projects/xtpl)
- Log4php - A PHP port of Log4j, the most popular Java logging framework, created by Ceki Gülcü (http://www.vxr.it/log4php)

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:


About.php (/2066_09_code/modules/Home) - gedit
File Edit View Search Tools Documents Help
New Open Save Print Undo Redo Cut Copy Paste Find Replace
About.php

```
<table><tr><td valign=top>
<img src=modules/Home/about.jpg width=176 height=144></td><td>
<p>Mark Alexander Bain hasn't always been the leading authority on open
source software that you know him as now. Back in the late seventies he
started work as a woodsman at Bowood Estates in Wiltshire. After that he
spend a number of years working at Lowther Wildlife Park in Cumbria -
it's not clear if his character made him suitable for looking after
packs of wolves, or whether the experience made him the way he is now.</
p>
```

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


Penguin PI - SugarCRM - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: http://hector/penguin_pi/index.php?module=Home&action=About
Penguin PI
My Account | Employees | Logout | About
SUGARSUITE
Home Web Sites Calendar Activities Contacts Accounts Leads Preliminary Investigations Investigations Bug Tracker Documents Emails
Welcome Korora
Search
GO
Last Viewed: none
Shortcuts
Create Contact
Enter Business Card
Create Account
Create Lead
Create Preliminary Investigation
Create Case
Report Bug
Schedule Meeting
Schedule Call
Create Task
Compose Email

Mark Alexander Bain hasn't always been the leading authority on open source software that you know him as now. Back in the late seventies he started work as a woodsman at Bowood Estates in Wiltshire. After that he spend a number of years working at Lowther Wildlife Park in Cumbria – it's not clear if his character made him suitable for looking after packs of wolves, or whether the experience made him the way he is now.

In the mid eighties there was a general down turn in the popularity of animal parks in the UK, and Mark found him out of work with two young sons (Simon and Michael) – but with a growing interest in programming. His wife had recently bought him the state of the art Sinclair ZX 81 – and it was her who suggested that he went to college to study computing.

Mark left college in 1989 and joined Vodafone – then a very small company – where he started writing programs using VAX/VMS. It was shortly after that, that he became addicted to something that was to drastically affect the rest of his life – Unix. His demise was further compounded when he was introduced to Oracle. After that there was no saving him. Over the next few years Vodafone became the multinational company that it is now, and Mark progressed from Technician to Engineer, and from Engineer to Senior Engineer and finally to Principal Engineer.

At the turn of the century general ill health made Mark reconsider his career – and his wife again came to his rescue when she saw a job advert for a lecturer at the University of Central Lancashire. It was also her that suggested that he should think about writing.

Today Mark writes regularly for Linux Format, Newsforge.com and Linux Journal. He's still teaching. And (apparently) he writes books as well.

Home | Web Sites | Calendar | Activities | Contacts | Accounts | Leads | Preliminary Investigations | Investigations | Bug Tracker | Documents | Emails
Campaigns | Projects | RSS | Dashboard
Theme Penguin PI

Server response time: 0.563649 seconds.
© 2004-2006 SugarCRM Inc. All Rights Reserved.

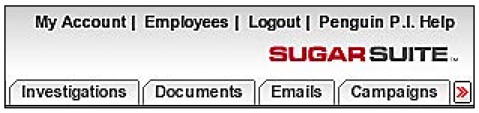
Of course, you may decide that you want to make the **About** screen more useful, something that will be helpful to your users—i.e. a *help* screen.

Changing the About Screen into a Help Screen

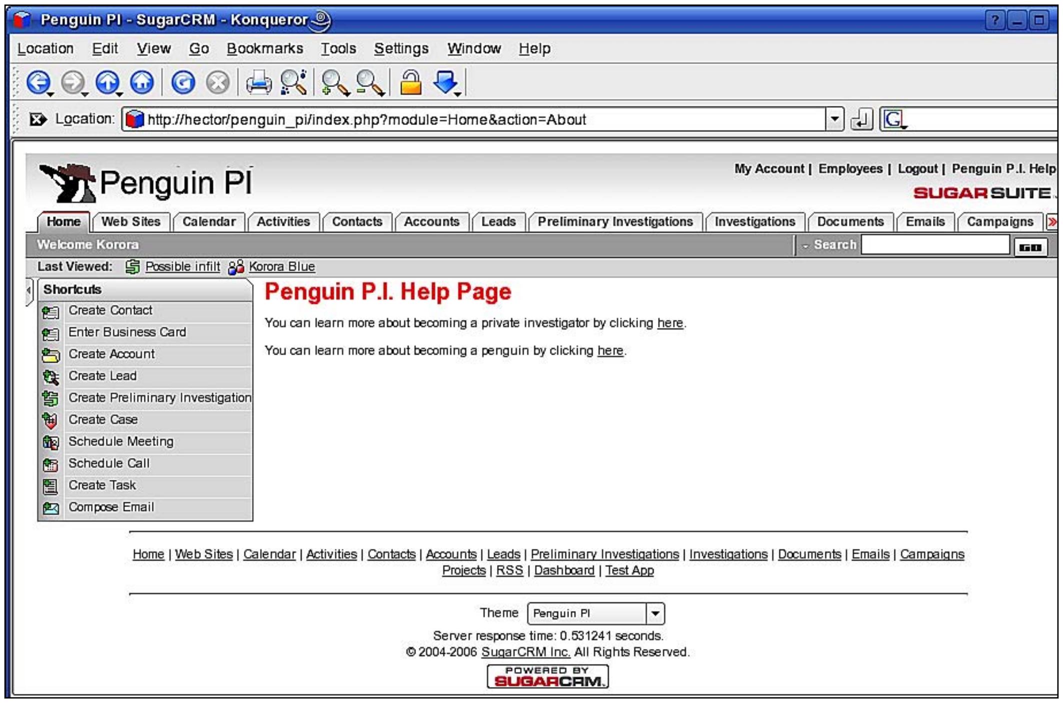
The first thing that you may want to do is to change the link text from **About** to something more appropriate—such as **Penguin P.I. Help**. To do that we need to return to the `custom/include/language/en_us.lang.php` file that we worked with in Chapter 1. Just add a line:

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

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:


Penguin PI - SugarCRM - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: http://hector/penguin_pi/index.php?module=Home&action=About
Penguin PI
My Account | Employees | Logout | Penguin P.I. Help
SUGAR SUITE
Home
Web Sites
Calendar
Activities
Contacts
Accounts
Leads
Preliminary Investigations
Investigations
Documents
Emails
Campaigns
Welcome Korora
Search
GO
Last Viewed: Possible infilt Korora Blue
Shortcuts
Create Contact
Enter Business Card
Create Account
Create Lead
Create Preliminary Investigation
Create Case
Schedule Meeting
Schedule Call
Create Task
Compose Email
Home | Web Sites | Calendar | Activities | Contacts | Accounts | Leads | Preliminary Investigations | Investigations | Documents | Emails | Campaigns
Projects | RSS | Dashboard | Test App
Theme Penguin PI
POWERED BY SUGARCRM

Penguin P.I. Help Page

You can learn more about becoming a private investigator by clicking here.

You can learn more about becoming a penguin by clicking here.

Server response time: 0.531241 seconds.
© 2004-2006 SugarCRM Inc. All Rights Reserved.

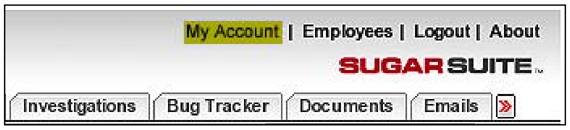
Of course, at this point you're probably thinking that this is all very interesting, but what you actually want to do is to start creating your own tabs. Obviously that's what we need to look at next.

Controlling the Visible Tabs

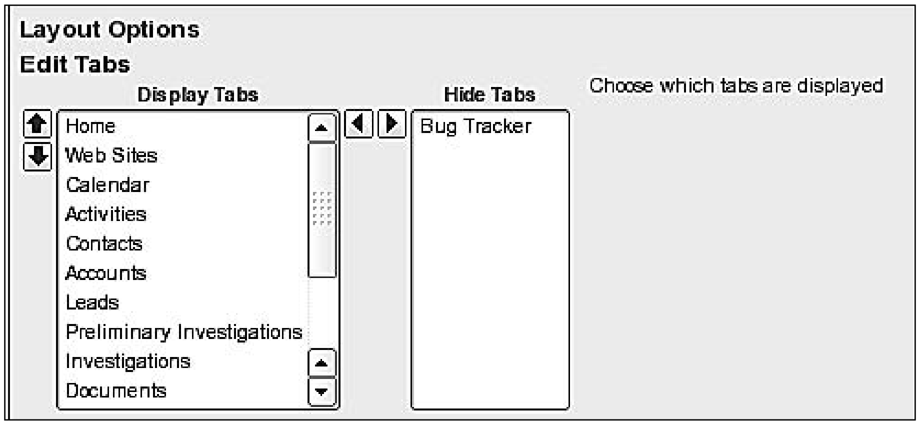
Before we create a new tab it's probably worth having a look at how we can control the visibility of SugarCRM tabs to our users.

User Control

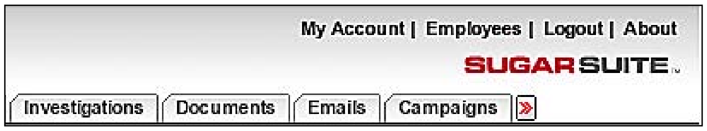
In fact, in an *out-of-the-box* SugarCRM installation, any user can choose which tabs are visible by clicking on **My Account**:



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



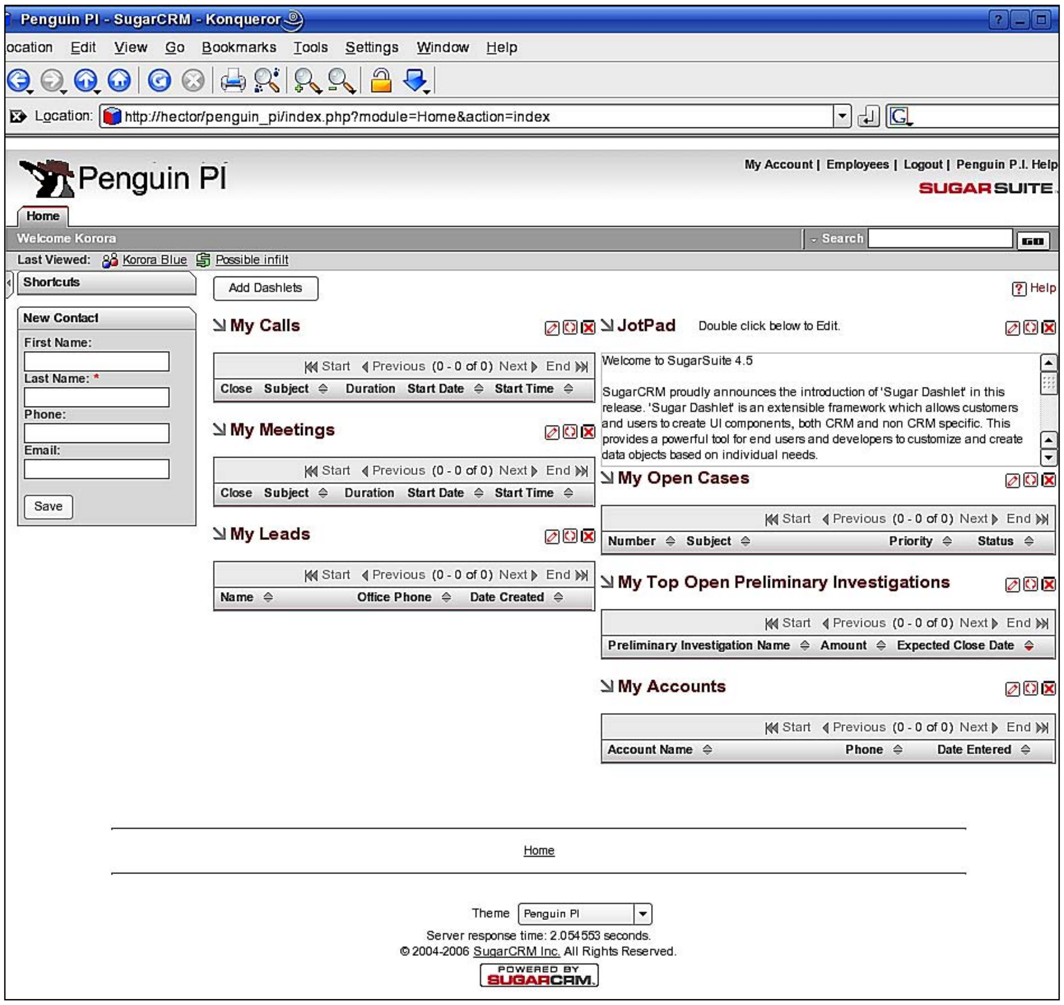
When she clicks the **Save** button then **Bug Tracker** will no longer be visible in the list of tabs:



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:


Penguin PI - SugarCRM - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: http://hector/penguin_pi/index.php?module=Home&action=index
Penguin PI
My Account | Employees | Logout | Penguin P.I. Help
SUGARSUITE
Home
Welcome Korora
Search
Last Viewed: Korora Blue Possible infilt
Shortcuts
Add Dashlets
Help
New Contact
First Name:
Last Name: *
Phone:
Email:
Save
My Calls
Close Subject Duration Start Date Start Time
My Meetings
Close Subject Duration Start Date Start Time
My Leads
Name Office Phone Date Created
JotPad Double click below to Edit.
My Open Cases
Number Subject Priority Status
My Top Open Preliminary Investigations
Preliminary Investigation Name Amount Expected Close Date
My Accounts
Account Name Phone Date Entered
Start Previous (0 - 0 of 0) Next End
Home
Theme Penguin PI
Server response time: 2.054553 seconds.
© 2004-2006 SugarCRM Inc. All Rights Reserved.
POWERED BY SUGARCRM

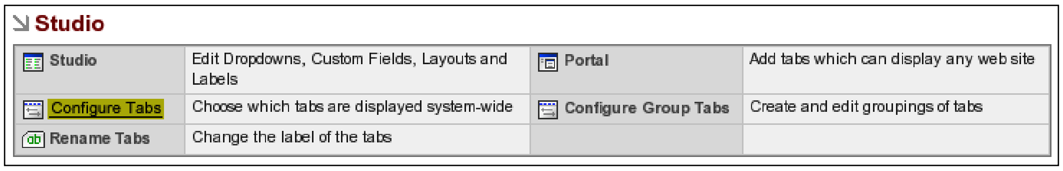
Welcome to SugarSuite 4.5

SugarCRM proudly announces the introduction of 'Sugar Dashlet' in this release. 'Sugar Dashlet' is an extensible framework which allows customers and users to create UI components, both CRM and non CRM specific. This provides a powerful tool for end users and developers to customize and create data objects based on individual needs.

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**:



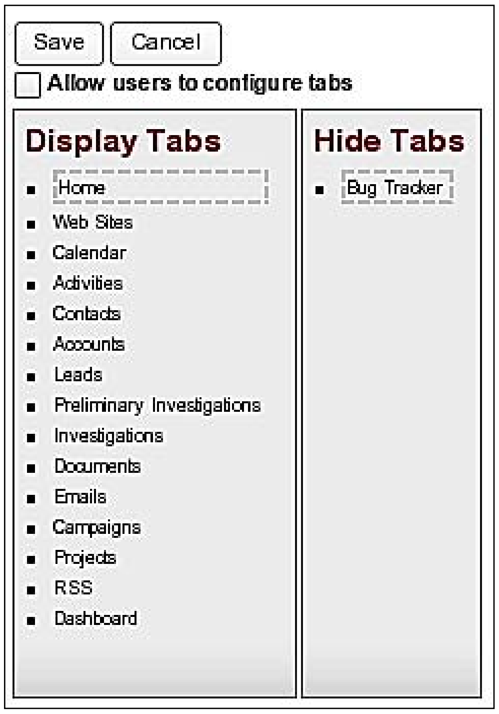
Studio

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:



Save Cancel

☐ Allow users to configure tabs

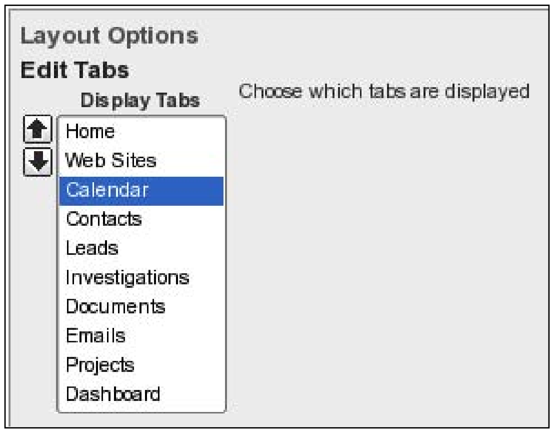
Display Tabs	Hide Tabs
Home	Bug Tracker
Web Sites	
Calendar	
Activities	
Contacts	
Accounts	
Leads	
Preliminary Investigations	
Investigations	
Documents	
Emails	
Campaigns	
Projects	
RSS	
Dashboard	

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

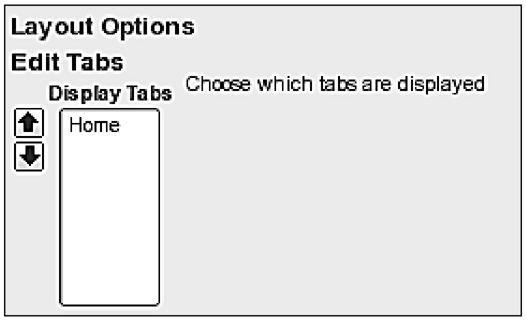
Unfortunately, it won't work with Opera (my personal favorite).

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:

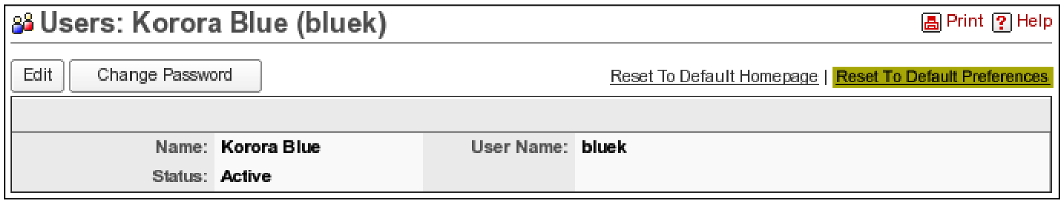


A Note about Administering Live Systems

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:



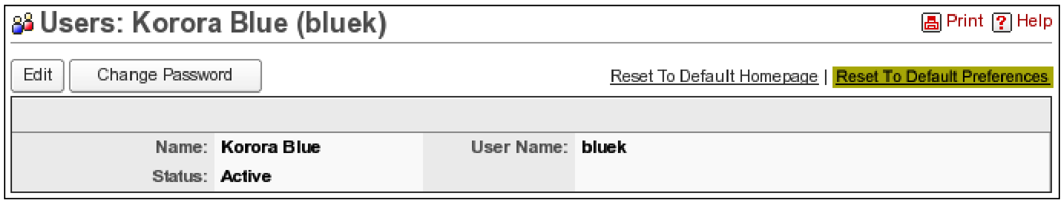
This is, of course, because Korora had previously removed all the other tabs, and now you've removed her ability to add any back in again. Fortunately the solution is quite simple—she just needs to scroll up to the top of her **My Account** screen, to where she'll see **Reset To Default Preferences**:


Users: Korora Blue (bluek)

Name: Korora Blue
Status: Active
User Name: bluek

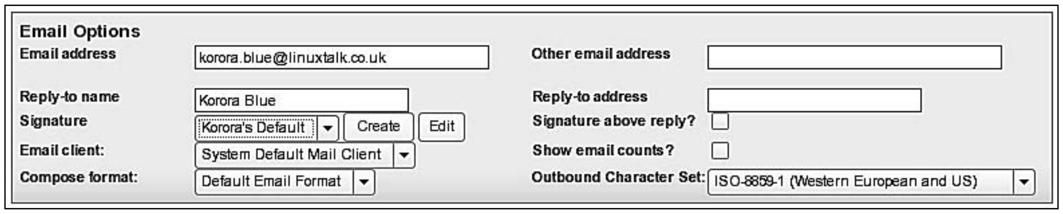
Clicking this link would cause Korora to be logged out, but once she logs back in her tabs would be set up as you had defined them.

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)

Name: Korora Blue
Status: Active
User Name: bluek

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	korora.blue@linuxtalk.co.uk	Other email address	
Reply-to name	Korora Blue	Reply-to address	
Signature	Korora's Default	Signature above reply?	☐
Email client:	System Default Mail Client	Show email counts?	☐
Compose format:	Default Email Format	Outbound Character Set:	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';
```

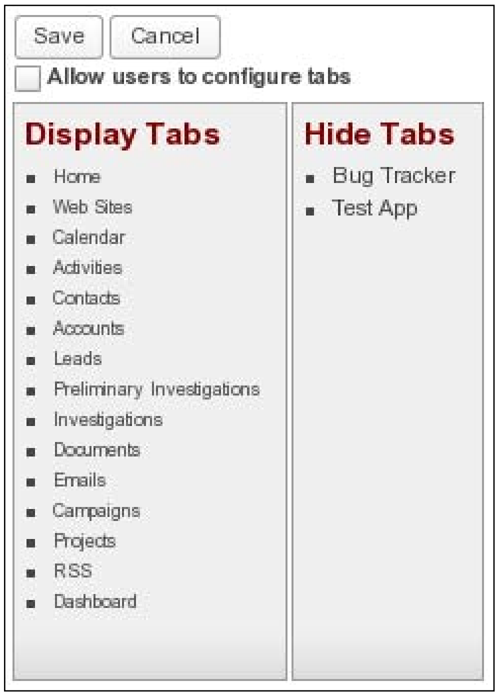
Finally edit `custom/include/language/en_us.lang.php` so that it contains the line:

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

for example:

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

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**:



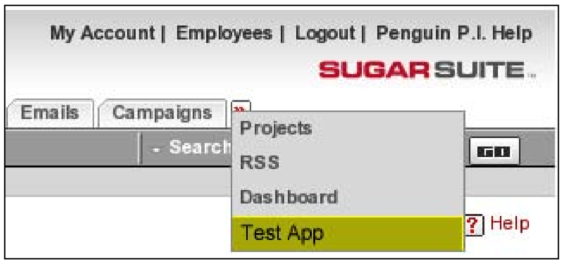
Display Tabs

- Home
- Web Sites
- Calendar
- Activities
- Contacts
- Accounts
- Leads
- Preliminary Investigations
- Investigations
- Documents
- Emails
- Campaigns
- Projects
- RSS
- Dashboard

Hide Tabs

- Bug Tracker
- Test App

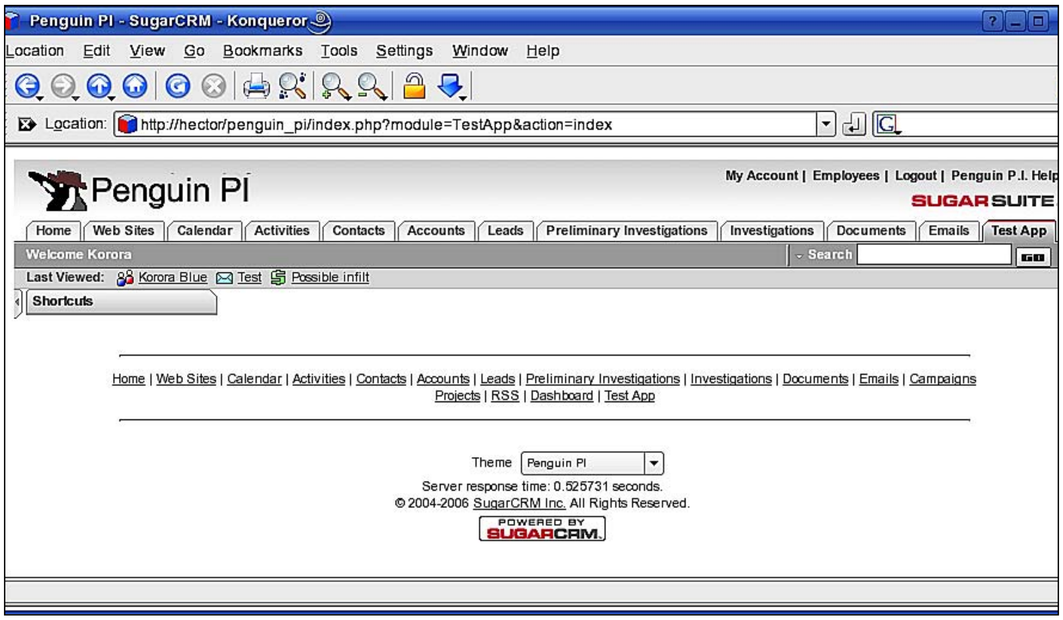
You'll need to drag your new tab into **Display Tabs**, and then it will be made available to your users:



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:


Penguin PI - SugarCRM - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: http://hector/penguin_pi/index.php?module=TestApp&action=index
Penguin PI
My Account | Employees | Logout | Penguin P.I. Help
SUGAR SUITE
Home
Web Sites
Calendar
Activities
Contacts
Accounts
Leads
Preliminary Investigations
Investigations
Documents
Emails
Test App
Welcome Korora
Search
GO
Last Viewed: Korora Blue Test Possible infilt
Shortcuts
Home | Web Sites | Calendar | Activities | Contacts | Accounts | Leads | Preliminary Investigations | Investigations | Documents | Emails | Campaigns
Projects | RSS | Dashboard | Test App
Theme Penguin PI
Server response time: 0.525731 seconds.
© 2004-2006 SugarCRM Inc. All Rights Reserved.
POWERED BY SUGARCRM

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`

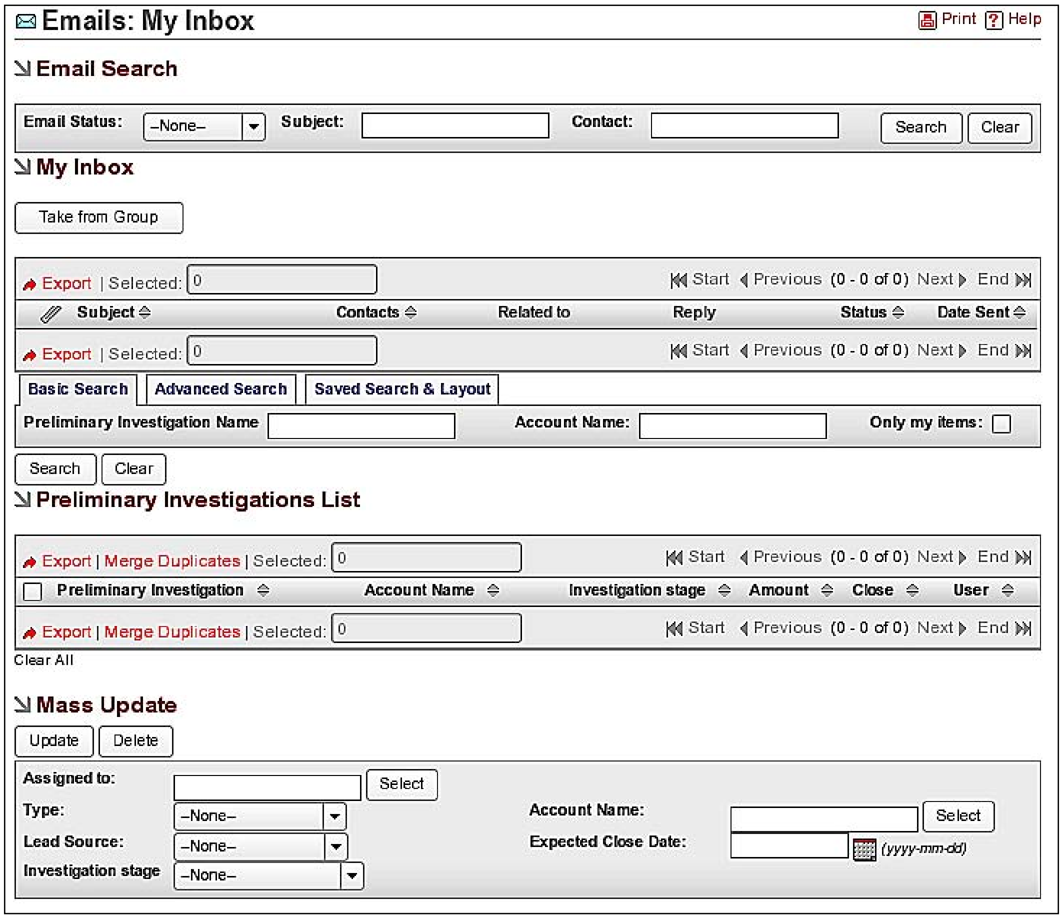
You will also remember that these files *do* need to exist, but they *don't* need to contain anything. Obviously the next stage is to edit these files in order to add contents to the tab. In fact we only have to edit one of the files—`modules/TestApp/index.php`. So, you could start by adding some HTML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

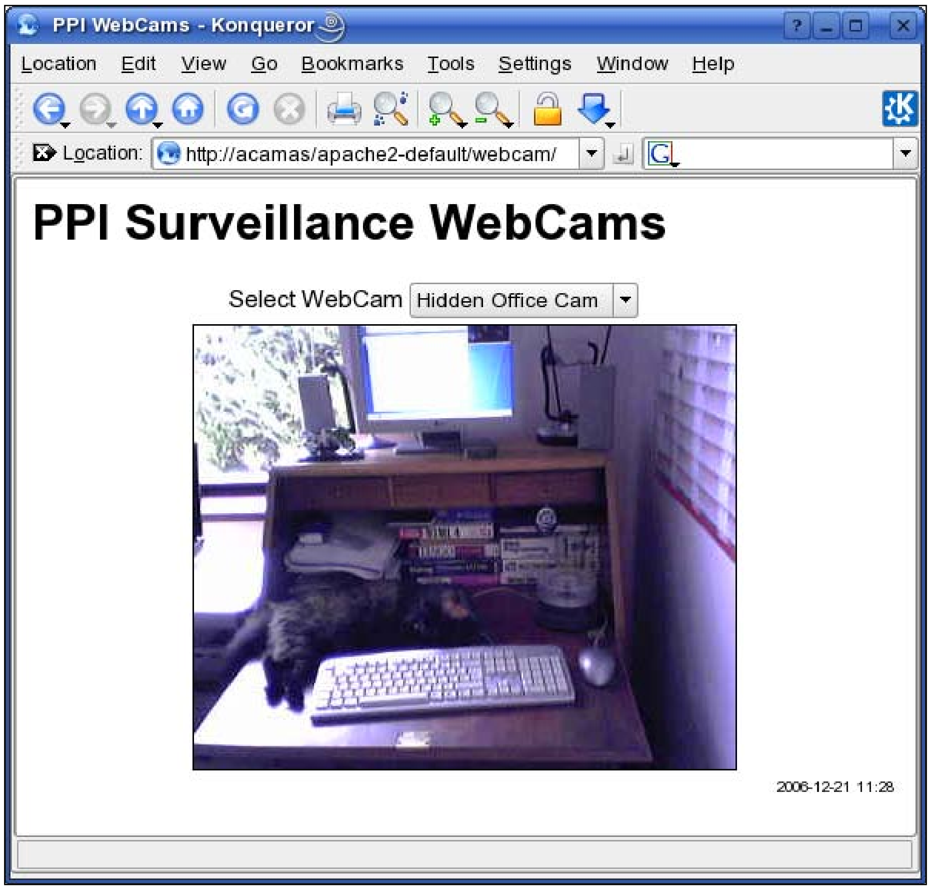


Having just said 'Don't use HTML, use PHP instead', I'm now going to say 'Actually, there is something interesting in HTML that you can make use of.' Why? Well, it's probable that you've got some useful applications that people are already using (and don't particularly want to lose). If these are web-based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


PPI WebCams - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: http://acamas/apache2-default/webcam/
Select WebCam Hidden Office Cam

PPI Surveillance WebCams

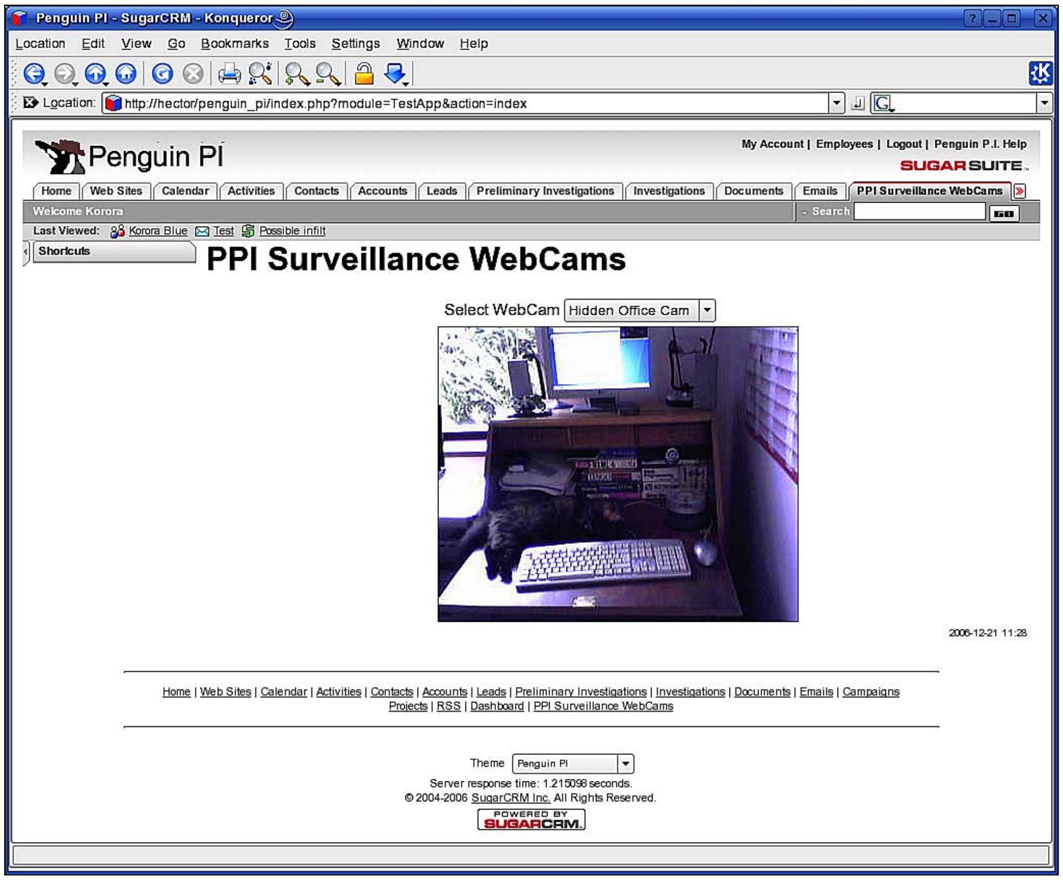
2006-12-21 11:28

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:

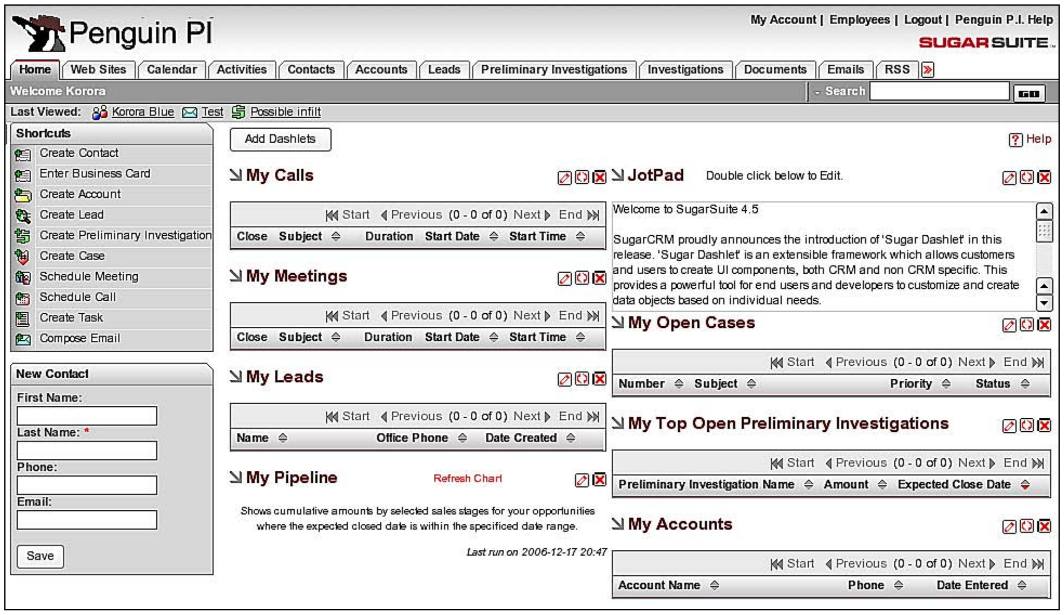
```
'TestApp' => 'PPI Surveillance WebCams',
```


Penguin PI - SugarCRM - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: http://hector/penguin_pi/index.php?module=TestApp&action=index
Penguin PI
My Account | Employees | Logout | Penguin P.I. Help
SUGARSUITE
Home Web Sites Calendar Activities Contacts Accounts Leads Preliminary Investigations Investigations Documents Emails PPI Surveillance WebCams
Welcome Korora
Search
Last Viewed: Korora Blue Test Possible infilt
Shortcuts
PPI Surveillance WebCams
Select WebCam Hidden Office Cam
2006-12-21 11:28
Home | Web Sites | Calendar | Activities | Contacts | Accounts | Leads | Preliminary Investigations | Investigations | Documents | Emails | Campaigns
Projects | RSS | Dashboard | PPI Surveillance WebCams
Theme Penguin PI
Server response time: 1.215098 seconds.
© 2004-2006 SugarCRM Inc. All Rights Reserved.
POWERED BY SUGARCRM

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:


Penguin PI
My Account | Employees | Logout | Penguin P.I. Help
SUGARSUITE
Home | Web Sites | Calendar | Activities | Contacts | Accounts | Leads | Preliminary Investigations | Investigations | Documents | Emails | RSS
Welcome Korora
Search
GO
Last Viewed: Korora Blue Test Possible infilt
Shortcuts
Create Contact
Enter Business Card
Create Account
Create Lead
Create Preliminary Investigation
Create Case
Schedule Meeting
Schedule Call
Create Task
Compose Email
New Contact
First Name:
Last Name: *
Phone:
Email:
Save
Add Dashlets
Help
My Calls
Start Previous (0 - 0 of 0) Next End
Close Subject Duration Start Date Start Time
My Meetings
Start Previous (0 - 0 of 0) Next End
Close Subject Duration Start Date Start Time
My Leads
Start Previous (0 - 0 of 0) Next End
Name Office Phone Date Created
My Pipeline
Refresh Chart
JotPad
Double click below to Edit.
My Open Cases
Start Previous (0 - 0 of 0) Next End
Number Subject Priority Status
My Top Open Preliminary Investigations
Start Previous (0 - 0 of 0) Next End
Preliminary Investigation Name Amount Expected Close Date
My Accounts
Start Previous (0 - 0 of 0) Next End
Account Name Phone Date Entered

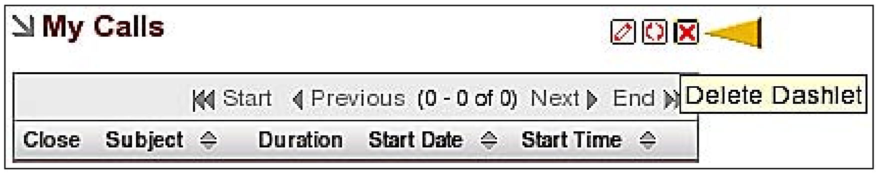
Welcome to SugarSuite 4.5

SugarCRM proudly announces the introduction of 'Sugar Dashlet' in this release. 'Sugar Dashlet' is an extensible framework which allows customers and users to create UI components, both CRM and non CRM specific. This provides a powerful tool for end users and developers to customize and create data objects based on individual needs.

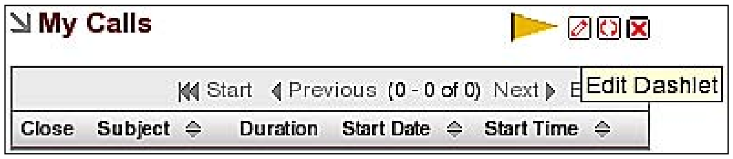
Shows cumulative amounts by selected sales stages for your opportunities where the expected closed date is within the specificed date range.

Last run on 2006-12-17 20:47

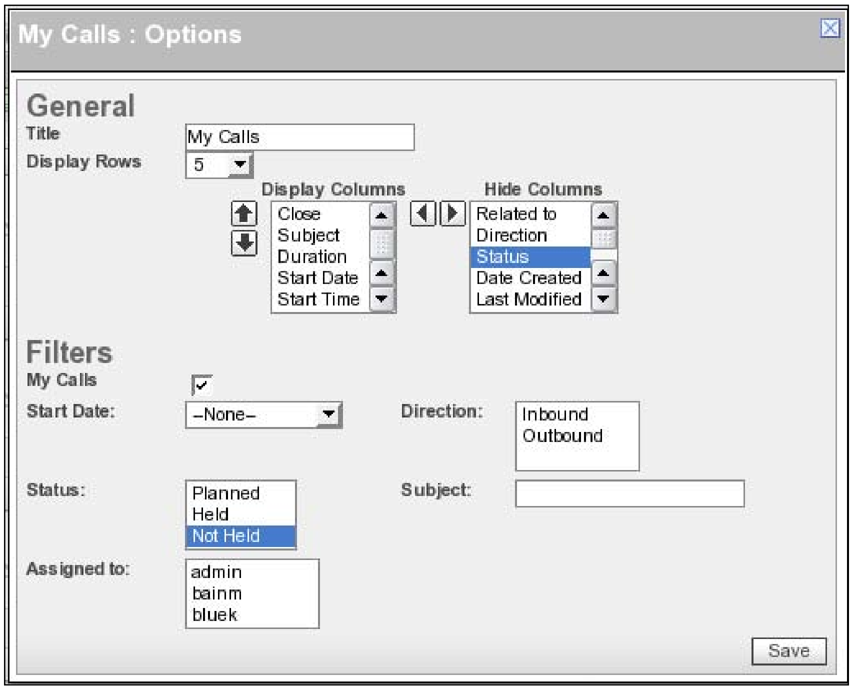
Users can edit the screen by removing Dashlets:


My Calls
Start Previous (0 - 0 of 0) Next End
Delete Dashlet
Close Subject Duration Start Date Start Time

Users can even customize each Dashlet themselves:


My Calls
Start Previous (0 - 0 of 0) Next E
Edit Dashlet
Close Subject Duration Start Date Start Time

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

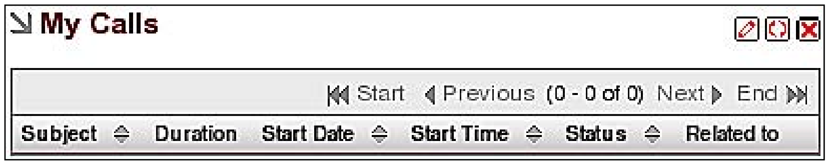
Status: Planned, Held, Not Held

Subject:

Assigned to: admin, bainm, bluek

Save

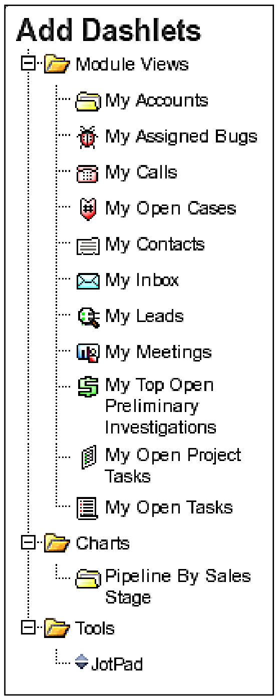
So that we see the tabs, and they can display exactly what they want:


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:



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

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

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');
?>
```

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.

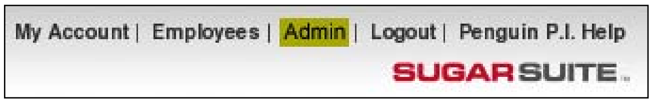
Next we need to create the dashlet file, `PPIDashlet.php`. In this case we're just going to get the dashlet to display some text:

```
<?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);
     }
  }
?>
```

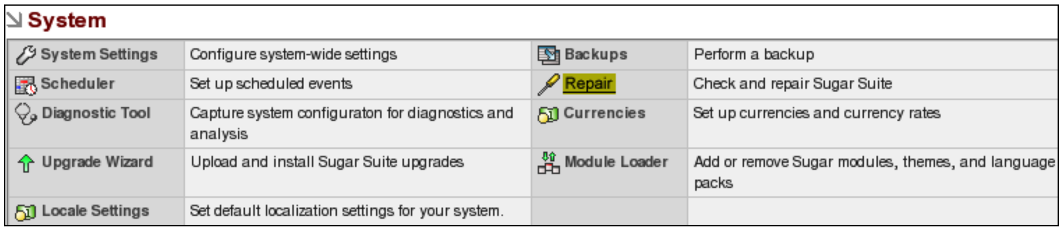
OK—not mind boggling functionality, but it's enough to show how to quickly create a Dashlet. As the book progresses we'll make the functionality more complicated. However, for the time being, let's look at how we make our new (simple) Dashlet available to our users.

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:



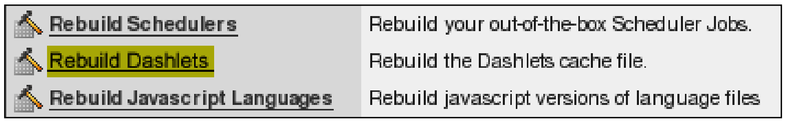
Having clicked on **Admin** you'll need to look for the **System** section, and then (even though this may seem strange) find the link marked **Repair**.



System

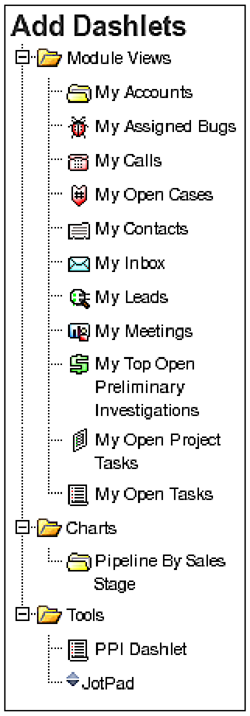
System Settings	Configure system-wide settings	Backups	Perform a backup
Scheduler	Set up scheduled events	Repair	Check and repair Sugar Suite
Diagnostic Tool	Capture system configuraton for diagnostics and analysis	Currencies	Set up currencies and currency rates
Upgrade Wizard	Upload and install Sugar 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

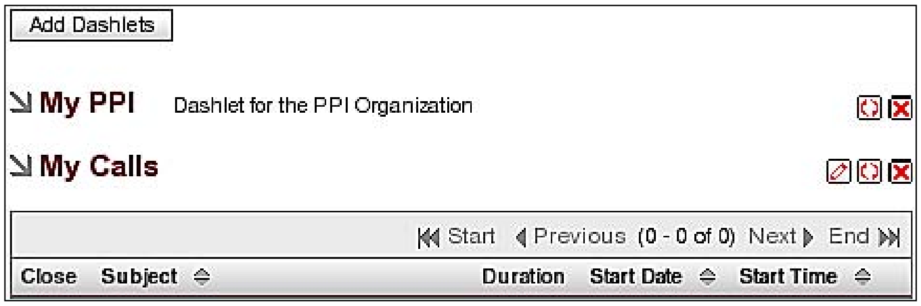
Your new Dashlet will now be available to all your users (in the **Tools** section):



Add Dashlets

- Module Views
  - My Accounts
  - My Assigned Bugs
  - My Calls
  - My Open Cases
  - My Contacts
  - My Inbox
  - My Leads
  - My Meetings
  - My Top Open Preliminary Investigations
  - My Open Project Tasks
  - My Open Tasks
- Charts
  - Pipeline By Sales Stage
- Tools
  - PPI Dashlet
  - JotPad

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



Add Dashlets

My PPI Dashlet for the PPI Organization

My Calls

Start Previous (0 - 0 of 0) Next End

Close Subject Duration Start Date Start Time

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

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.

3
Introducing Custom Fields

As you've worked your way through the first two chapters of this book, you have learned how to:

- Change the title of each of the module tabs that make up the SugarCRM front end.
- Change the terminology used in each of the module tabs, so that SugarCRM uses the same terms as your organization.
- Give SugarCRM a look and feel in keeping with your company's branding.
- Create your own module tabs and dashlets, either to add your own functionality, or to incorporate any existing web-based applications already used in your organization.

So, now you can provide a SugarCRM implementation that won't be completely alien to your intended users—hopefully they'll be able to use the application with minimum training.

Of course, there's more to it than just making SugarCRM look the way that you want. Let us imagine Korora Blue sat at her desk. The first thing that she does, every morning, is to evaluate all of the new preliminary investigations. When she does this then she decides if any surveillance needs to be carried out. As it stands, there is nowhere to store this in the SugarCRM application. So, obviously, we need to provide some extra fields for Korora so that she can do her job.

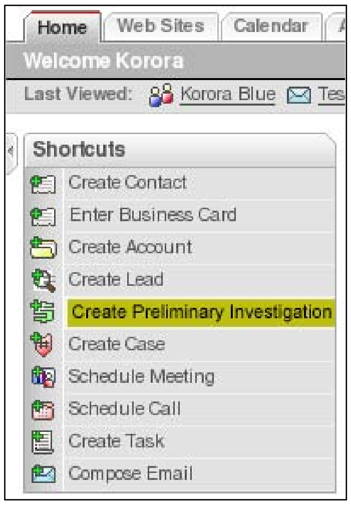
And that's the aim of this chapter—to show you how to add your own custom fields to SugarCRM. By the end of the chapter your implementation won't just look different to the standard, out-of-the-box SugarCRM application, it will actually start to behave differently.

Adding a Custom Field

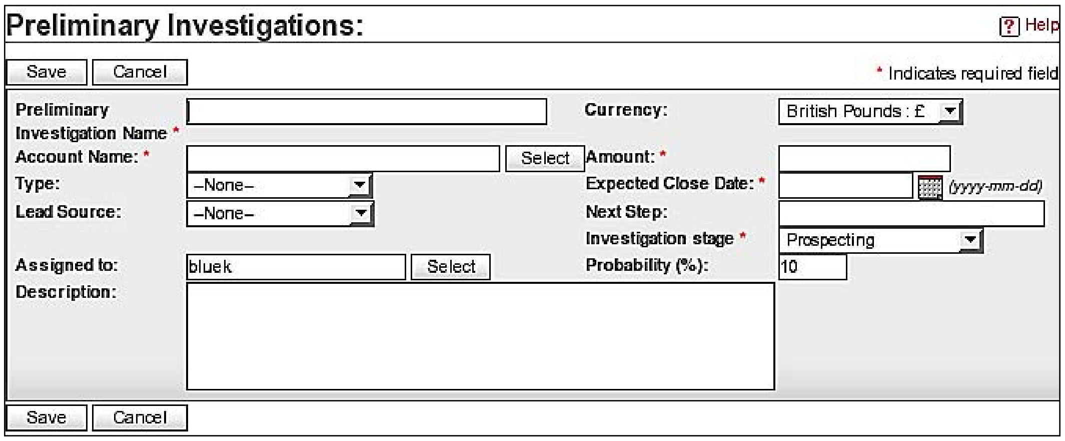
Before we jump in and add a new field let's have a look at what it is that Korora is trying to achieve.

The Standard Module Tab

When Korora decides that a new preliminary investigation is required she can do so by using the shortcuts on her **Home** tab:



Having clicked on **Create Preliminary Investigation** she just needs to fill in the appropriate details:



Preliminary Investigation Name *		Currency:	British Pounds : £
Account Name: *		Amount: *	
Type:	–None–	Expected Close Date: *	(yyyy-mm-dd)
Lead Source:	–None–	Next Step:	
		Investigation stage *	Prospecting
Assigned to:	bluek	Probability (%):	10
Description:			

So, is there a show stopper here? Well, in Korora's case, yes. She needs to be able to record whether or not any surveillance is required, but that's not possible with the standard form. What she actually needs is an extra drop-down field with a 'Yes/No' option.

The General Process for Creating a Custom Dropdown

Having decided that Korora needs a dropdown adding to **Preliminary Investigations** we need to go through three stages in order to add the custom field:

1. Create the options for the drop-down box
2. Create the custom field itself and link the options to it
3. Place the drop-down box on a module tab

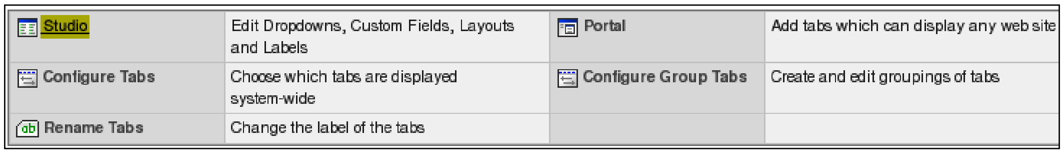
To start with there are two ways to create your own drop-down box:

- By using Studio
- Manually

We'll start by using Studio.

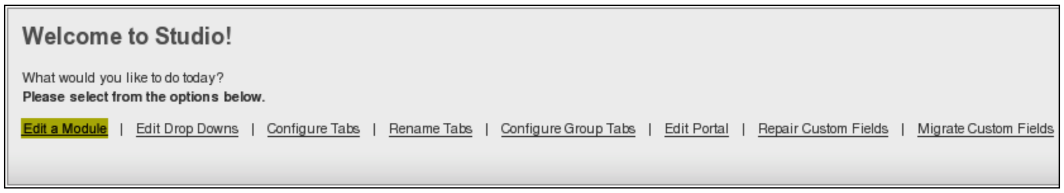
Using Studio to Create a Drop-down Box

As you would expect you will need to log on to SugarCRM as an administrator, and then go to the **Admin** screen where you'll find the link to the **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		

Once you're in the Studio, you need to find the **Edit a Module** link:

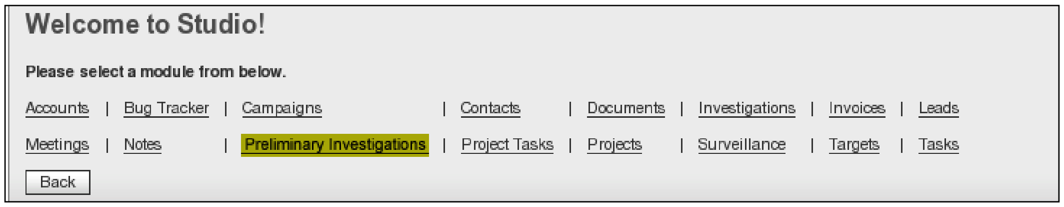


Welcome to Studio!

What would you like to do today?
Please select from the options below.

Edit a Module | Edit Drop Downs | Configure Tabs | Rename Tabs | Configure Group Tabs | Edit Portal | Repair Custom Fields | Migrate Custom Fields

And then choose the module that you want to edit. At the moment we're interested in **Preliminary Investigations** (you'll remember that this was formerly named **Opportunities**):


Welcome to Studio!

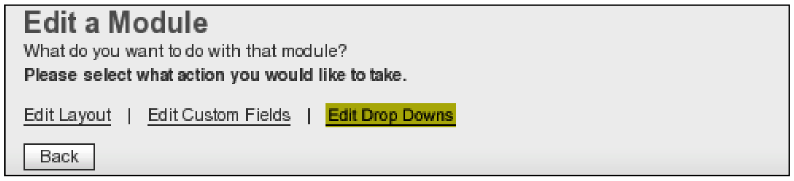
Please select a module from below.

Accounts | Bug Tracker | Campaigns | Contacts | Documents | Investigations | Invoices | Leads

Meetings | Notes | Preliminary Investigations | Project Tasks | Projects | Surveillance | Targets | Tasks

Back

At this point you have to select **Edit Drop Downs** from the available options.


Edit a Module

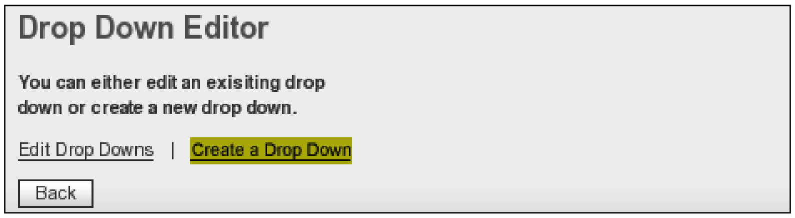
What do you want to do with that module?

Please select what action you would like to take.

Edit Layout | Edit Custom Fields | Edit Drop Downs

Back

Once you've done that you can edit existing dropdowns, but at the moment select **Create a Drop Down.**

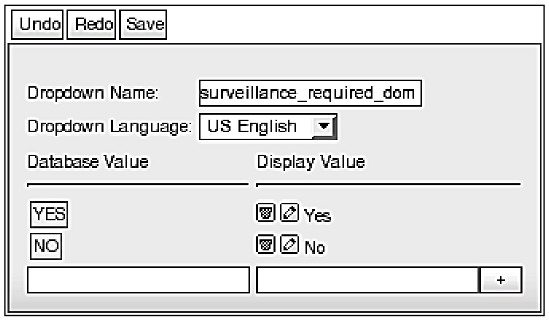

Drop Down Editor

You can either edit an exisiting drop down or create a new drop down.

Edit Drop Downs | Create a Drop Down

Back

You can now create your new dropdown by giving it a suitable name, and assigning options to it. As you create each option you'll find that you need to enter two values—the value to be displayed on the screen and the value to stored in the database:

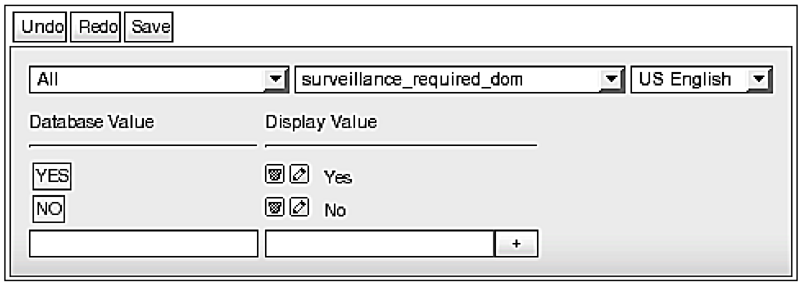

Undo | Redo | Save

Dropdown Name: surveillance_required_dom

Dropdown Language: US English

Database Value	Display Value
YES	Yes
NO	No

Interestingly, once you've clicked **Save** you'll find that the dropdown is actually available to **All** modules, and not just **Preliminary Investigations**:



Database Value	Display Value
YES	Yes
NO	No

Now, all of that may seem a little long-winded—especially if you have a number of dropdowns to create. In fact, you will probably find that it is quicker and easier to create a dropdown manually, and so that's what we'll look at next.

Manually Adding a Drop-down Box

You will find that adding a dropdown manually is much simpler than adding one using the Studio—as long as you're happy to edit files. Well, only one file. You need to edit a file that you have already used in Chapters 1 and 2: `custom/include/language/en_us.lang.php`.

Now we're going to use `custom/include/language/en_us.lang.php` to create a new dropdown. All you have to do is to add the definition for the dropdown:

```
$app_list_strings['surveillance_required_dom'] = array (
    'YES' => 'Yes',
    'NO' => 'No',
  );
```

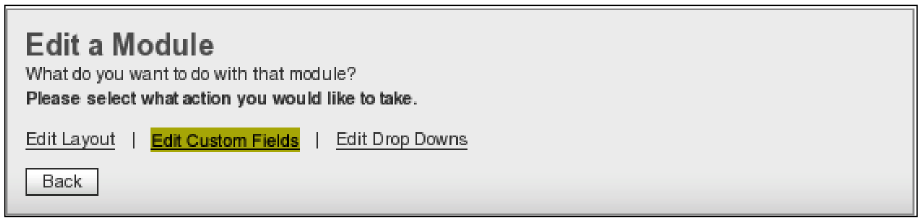
And that's all there is to creating a new dropdown. Next we need to look at the second stage of adding your custom field to a module tab—creating the custom field itself.

You have learned that we can create a dropdown either by using the Studio or by doing it manually. Custom field creation is exactly the same.

Using Studio to Create a Custom Field

Having defined a dropdown we now need to create a custom field. In actual fact this is a table column in the database. We will see how to do this manually, but first let's see how to use the Studio to do the job.

You'll remember that earlier we selected a module to edit (**Preliminary Investigations**) and then **Edit Drop Downs**—this time select **Edit Custom Fields**:

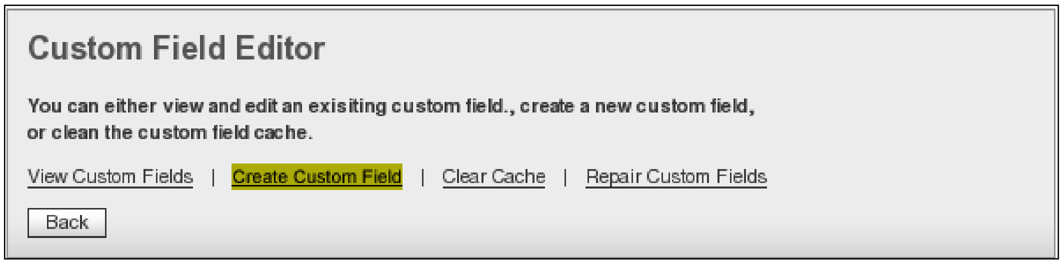


Edit a Module

What do you want to do with that module?

Please select what action you would like to take.

Next you'll need to click on **Create Custom Field**:

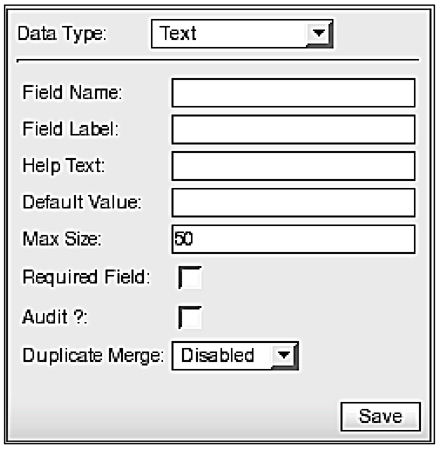


Custom Field Editor

You can either view and edit an exisiting custom field., create a new custom field, or clean the custom field cache.

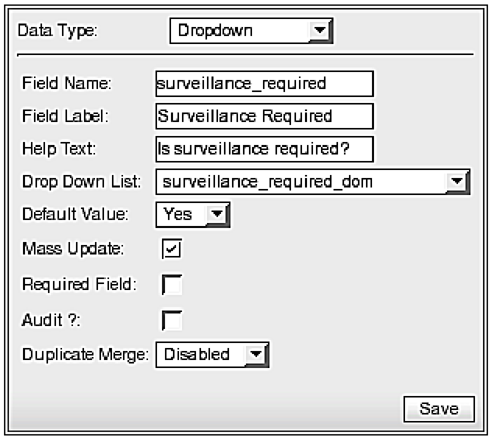
View Custom Fields | Create Custom Field | Clear Cache | Repair Custom Fields
Back

And then you'll be presented with the default field—the **Text** data type:



Data Type:	Text
Field Name:	
Field Label:	
Help Text:	
Default Value:	
Max Size:	50
Required Field:	☐
Audit ?:	☐
Duplicate Merge:	Disabled

Of course, we're not interested in the text field, we're interested in creating a dropdown. If you select **Dropdown** then you'll see that you'll be presented with a list of existing option lists. The one that we've already created should be available in this list:

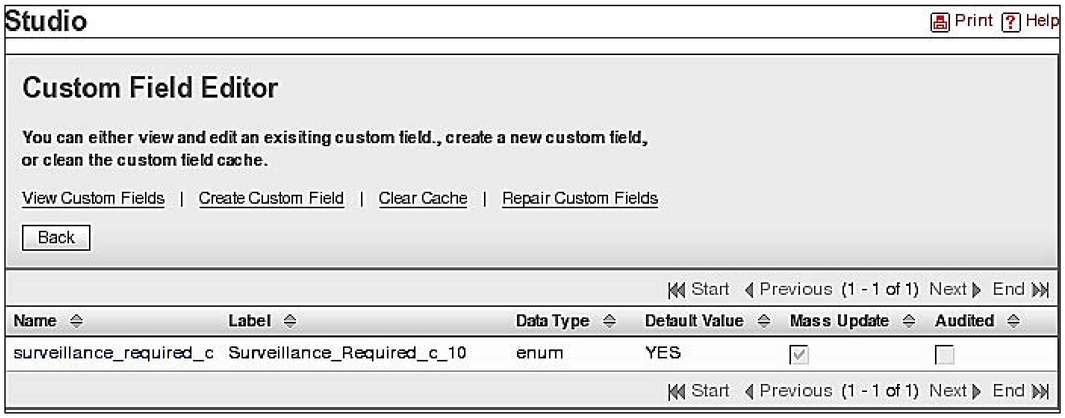


Data Type:	Dropdown
Field Name:	surveillance_required
Field Label:	Surveillance Required
Help Text:	Is surveillance required?
Drop Down List:	surveillance_required_dom
Default Value:	Yes
Mass Update:	✓
Required Field:	
Audit ?:	
Duplicate Merge:	Disabled

Most of the fields are self explanatory, but it's worth just looking at a couple of them before we move on:

- **Mass Update**—If you enable this then you can include your field in the mass updates in the module tab screen.
- **Audit**—This tells SugarCRM to track any change that you make to your field.
- **Duplicate Merge**—This allows you to merge any duplicate records.

Once you've entered all of the details then press **Save**, and the Studio will update the SugarCRM database for you:



Studio

Custom Field Editor

You can either view and edit an exisiting custom field., create a new custom field, or clean the custom field cache.

View Custom Fields | Create Custom Field | Clear Cache | Repair Custom Fields
Back
Start Previous (1 - 1 of 1) Next End

Name	Label	Data Type	Default Value	Mass Update	Audited
surveillance_required_c	Surveillance_Required_c_10	enum	YES	✓	

You will notice that the details are not saved exactly as you entered them:

- **_c** is appended onto the end of your field name.
- **_c_10** is appended onto the end of your field label.

- Spaces in the field name and label are replaced by underscores.
- The data type is no longer shown as **Dropdown**, it is defined as **enum**—this is because there isn't a data type of 'dropdown'; the data is actually stored as an **enum** (i.e. a list).

Now you're ready to add the dropdown to your module tab. However, first you may want to consider how to create the field definitions manually.

Creating the Custom Field Manually

We've already established that a custom field is simply a reference stored in a database. All we have to do is to insert the appropriate information into the database ourselves:

```
insert into fields_meta_data
    (      id,
           name,
           label,
           help,
           custom_module,
           data_type,
           ext1,
           default_value,
           date_modified
    )
values
    (      'Opportunitiessurv1_req_c',
           'surv1_req_c',
           'surv1_req_c_10',
           'Surveillance?',
           'Opportunities',
           'enum',
           'surveillance_required_dom',
           'YES',
           now()
    )
;
```

There is, of course, a major advantage here, you can create a simple script to create all of the custom fields that you need—especially useful when you come to migrating from your development environment to your live environment (we'll discuss that further in Chapter 7).

Now, we're not quite finished with the database yet. If you've used Studio to add a custom field for the Opportunities module then you will find that you have a table named `opportunities_cstm` in the database (and don't forget—the renaming of the module is only for the browser—the SugarCRM structure remains the same). If not then you'll need to create it yourself. This table requires a new field for every custom field that you add (to `opportunities`, of course).

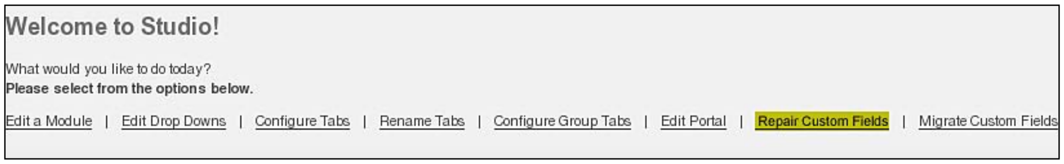
So, depending on whether `opportunities_cstm` exists or not, you'll need to do one of the following:

```
create table  opportunities_cstm (surv1_req_c varchar(150));
```

or:

```
alter table opportunities_cstm add surv1_req_c varchar(150);
```

On the other hand, once you've created your fields, then you may prefer to let the Studio set up `opportunities_cstm` for you. To do that just uses the Studio's **Repair Custom Fields** facility:

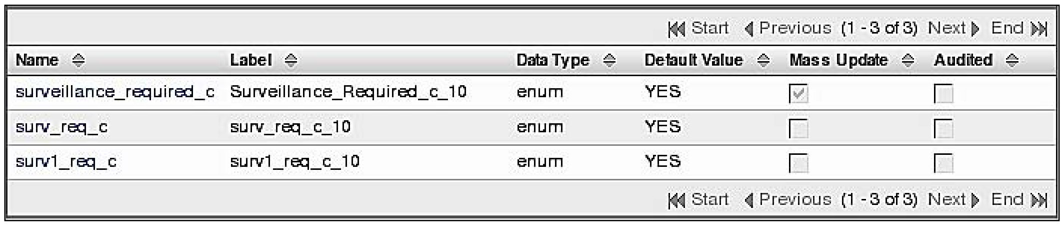


Welcome to Studio!

What would you like to do today?
Please select from the options below.

Edit a Module | Edit Drop Downs | Configure Tabs | Rename Tabs | Configure Group Tabs | Edit Portal | Repair Custom Fields | Migrate Custom Fields

Once you've added all of the fields that you want then you can view them via the Studio:

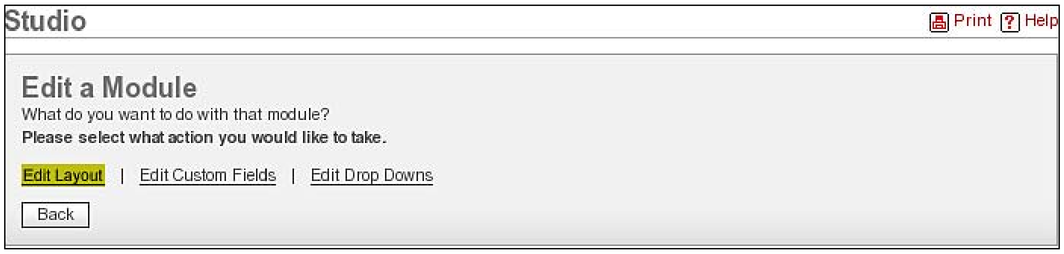


Name	Label	Data Type	Default Value	Mass Update	Audited
surveillance_required_c	Surveillance_Required_c_10	enum	YES	✓	☐
surv_req_c	surv_req_c_10	enum	YES	☐	☐
surv1_req_c	surv1_req_c_10	enum	YES	☐	☐

With your custom fields defined you can add them to your module tabs.

Adding the Dropdown to a Module Tab

You'll need to return to the module editor in the Studio in order to add your newly created custom field, but this time go to the **Edit Layout** link:

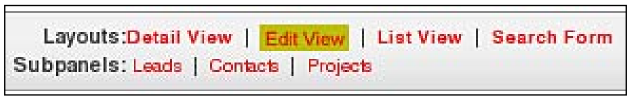

Studio
Print Help

Edit a Module
What do you want to do with that module?
Please select what action you would like to take.

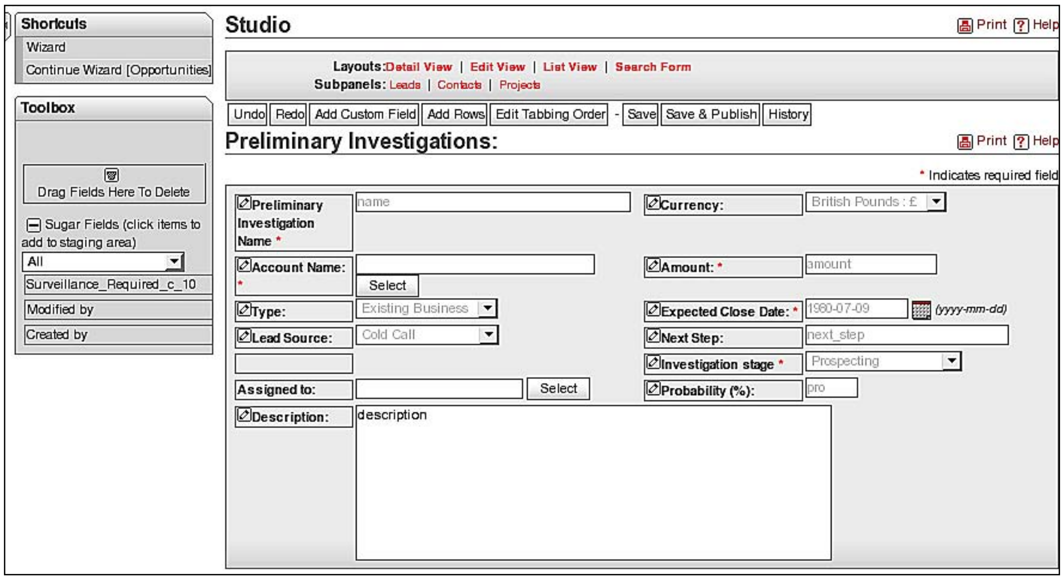
Edit Layout | Edit Custom Fields | Edit Drop Downs

Back

At this point you'll be presented with the list the layouts that you're able to modify:


Layouts: Detail View | Edit View | List View | Search Form
Subpanels: Leads | Contacts | Projects

Now, I'm sure you'll agree that so far the development environment hasn't been exactly WYSIWYG. However, if you click on **Edit View** (for example), you'll find that you see the layout to be modified, and a toolbox:


Shortcuts
Wizard
Continue Wizard [Opportunities]

Toolbox
Drag Fields Here To Delete
Sugar Fields (click items to add to staging area)
All
Surveillance_Required_c_10
Modified by
Created by

Studio
Print Help

Layouts: Detail View | Edit View | List View | Search Form
Subpanels: Leads | Contacts | Projects

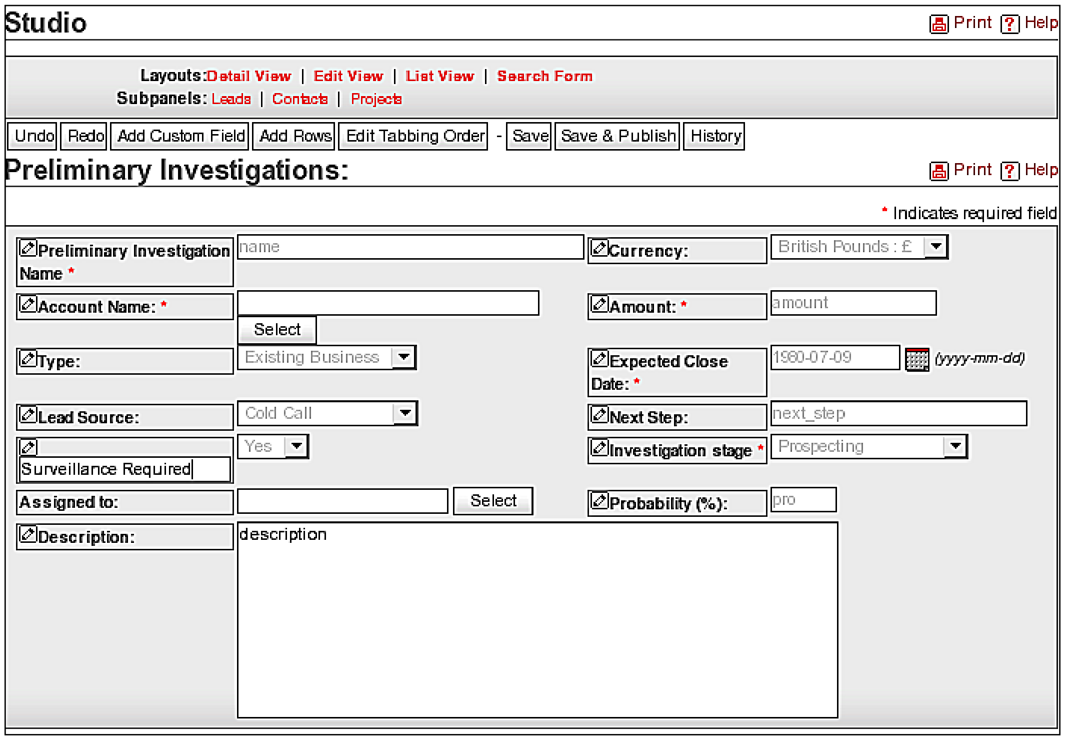
Undo | Redo | Add Custom Field | Add Rows | Edit Tabbing Order | - | Save | Save & Publish | History

Preliminary Investigations:
Print Help

* Indicates required field

Preliminary Investigation Name *	name	Currency:	British Pounds : £
Account Name: *	Select	Amount: *	amount
Type:	Existing Business	Expected Close Date: *	1960-07-09 (yyyy-mm-dd)
Lead Source:	Cold Call	Next Step:	next_step
		Investigation stage *	Prospecting
Assigned to:	Select	Probability (%):	pro
Description:	description		

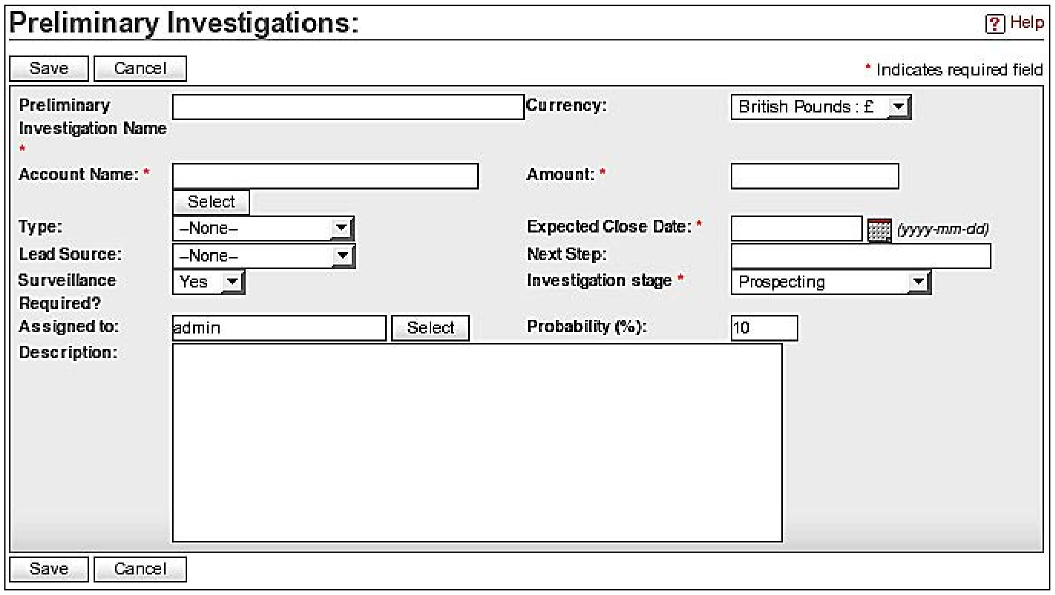
You'll see that the toolbox contains the field that we created, and you can drag and drop it into an appropriate location (you'll find a space between **Lead Source** and **Assigned to**):



All that is left to define is the text to be displayed next to your new drop-down box. You can either do this by using the Studio, or by adding a line to `custom/modules/ Opportunities/ language/en_us.lang.php`:

```
$mod_strings['Surveillance_Required_c_10'] = "Surveillance Required?";
```

Once you've done all of that then you can click on **Save & Publish** to make the new layout available to all of your users:



Preliminary Investigations:

Help

Save Cancel

* Indicates required field

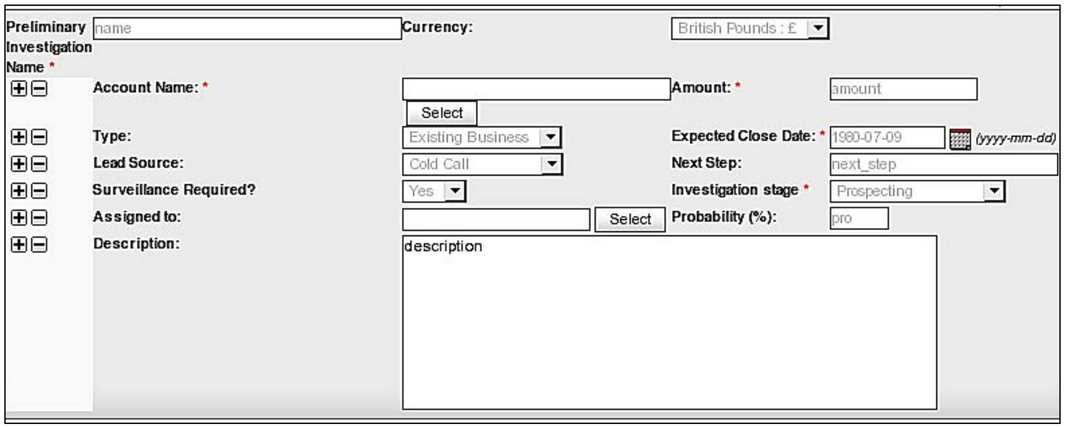
Preliminary Investigation Name *		Currency:	British Pounds : £
Account Name: *	Select	Amount: *	
Type:	–None–	Expected Close Date: *	(yyyy-mm-dd)
Lead Source:	–None–	Next Step:	
Surveillance Required?	Yes	Investigation stage *	Prospecting
Assigned to:	admin Select	Probability (%):	10
Description:			

Save Cancel

Of course that's fine for a single field (there's a gap for it), but you're going to have to make space for any additional fields.

Adding Rows

If you need to add more that one field then you'll need to add additional rows. You may have already noticed that one of the buttons on the layout screen is entitled **Add Rows**—click on this and you can add as many rows as you need:

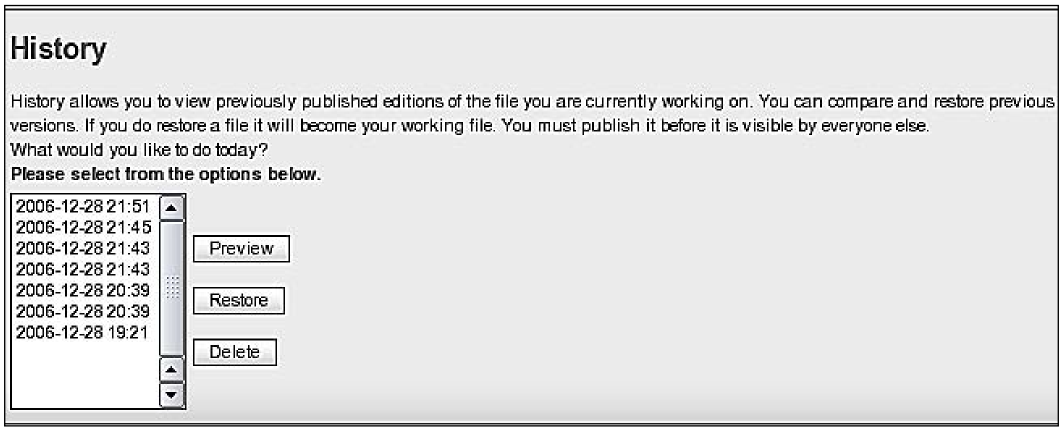


Preliminary Investigation Name *	name	Currency:	British Pounds : £
Account Name: *	Select	Amount: *	amount
Type:	Existing Business	Expected Close Date: *	1980-07-09 (yyyy-mm-dd)
Lead Source:	Cold Call	Next Step:	next_step
Surveillance Required?	Yes	Investigation stage *	Prospecting
Assigned to:	Select	Probability (%):	pro
Description:	description		

Adding rows is very easy (just press one of the **+** buttons). However, you need to be careful if you want to delete a row. If you accidentally delete a row containing important fields, you will find that there is no **Undo** button. You will also find that any fields that you delete by doing this do not appear in the toolbox when you return to the layout editing screen.

Recovering Previous Versions of a Layout

If, for any reason, you decide that you need to roll back to a previous layout then click on the **History** button. You can then view (and restore) the layout that you require:



History

History allows you to view previously published editions of the file you are currently working on. You can compare and restore previous versions. If you do restore a file it will become your working file. You must publish it before it is visible by everyone else.
What would you like to do today?
Please select from the options below.

2006-12-28 21:51
2006-12-28 21:45
2006-12-28 21:43
2006-12-28 21:43
2006-12-28 20:39
2006-12-28 20:39
2006-12-28 19:21

Manually Modifying Layouts

We've seen that we can use the Studio in order to modify the layout of the SugarCRM screens, but, as you'd expect, we can do this manually as well. The views that we can edit in the Studio are:

- Display
- Edit
- List
- Search

We've already established that the files for **Preliminary Investigations** are stored in the `modules/Opportunities` directory. All we have to do is to find the right files to edit. In the directory you'll find:

- `DisplayView.html`
- `EditView.html`

- `ListView.html`
- `SearchView.html`

If you have already modified the edit view, and now look at `EditView.html`, then you'll find that it contains something like:

```
<tr><!-- BEGIN: open_source -->
<td  class="dataLabel">
  <span sugar='slot11'>
    {MOD.Surveillance_Required_c_10}
  </span sugar='slot'>
</td>
<td class="dataField">
  <span sugar='slot11b'>
    <select title='{SURVEILLANCE_REQUIRED_C_HELP}'
    name="surveillance_required_c">{OPTIONS_SURVEILLANCE_REQUIRED_C}
    </select>
  </span sugar='slot'>
</td>
<td  class="dataLabel">
  <span sugar='slot12'>
    {MOD.LBL_SALES_STAGE}
    <span class="required">{APP.LBL_REQUIRED_SYMBOL}</span>
  </span sugar='slot'>
</td>
<td class="dataField">
  <span sugar='slot12b'>
    <select tabindex='2'
     name='sales_stage'>{SALES_STAGE_OPTIONS}
    </select>
  </span sugar='slot'>
</td>
<!-- END: open_source --></tr>
```

You'll realize that this is simple HTML code for adding lines to a table, but that it also makes use of some of our SugarCRM variables. Now, if we look back at the field that we manually created in this chapter then we'll see that its details are:

- name—surv1_req_c
- label—surv1_req_c_10

All we have to do is to add another line for our extra field at the end of the table of details in `EditView.html`:

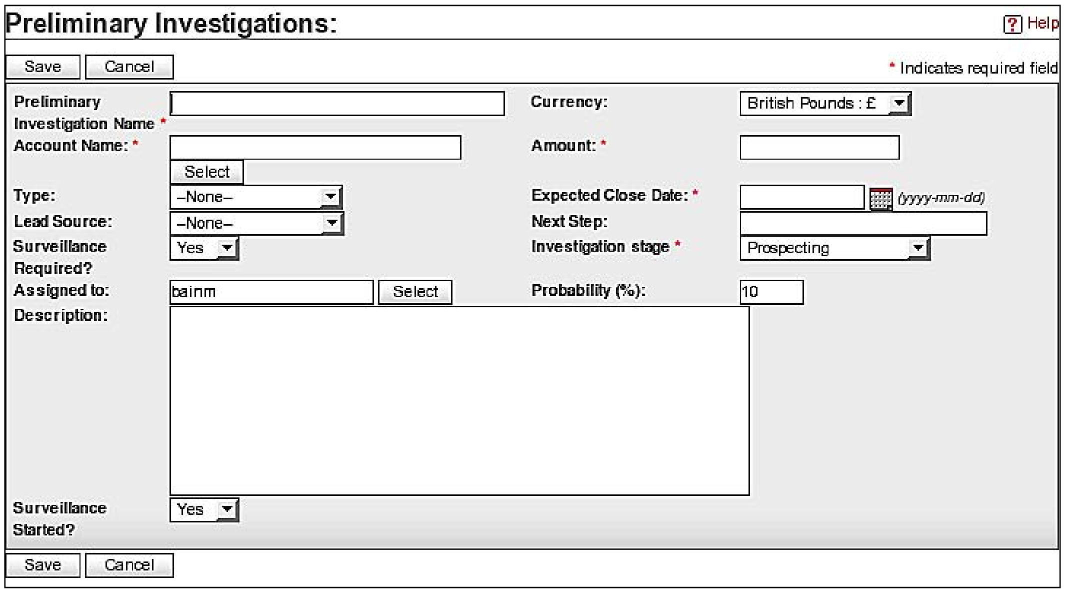
```
<tr><!-- BEGIN: open_source -->
<td  class="dataLabel">
```

```
    <span sugar='slot16'>
      {MOD.SURV1_REQ_C_10}
    </span sugar='slot'>
  </td>
  <td class="dataField">
    <span sugar='slot16b'>
      <select title='{SURV1_REQ_C_HELP}'
      name="surv1_req_c">{OPTIONS_SURV1_REQ_C}
      </select>
    </span sugar='slot'>
  </td>
  <!-- END: open_source --></tr>
```

Again, we're just using simple HTML to add a line to the table, but including references to the new field that we've created. And don't forget to modify `custom/modules/Opportunities/language/en_us.lang.php` to add a label for the dropdown:

```
  $mod_strings['surv1_req_c_10'] = "Surveillance Started?";
```

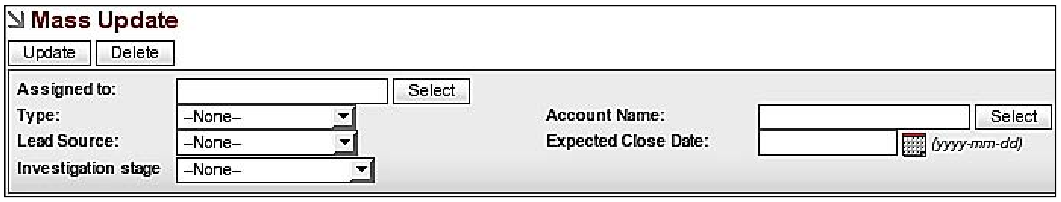
The end result is a screen containing two new dropdowns:



Preliminary Investigation Name *		Currency:	British Pounds : £
Account Name: *	Select	Amount: *	
Type:	–None–	Expected Close Date: *	(yyyy-mm-dd)
Lead Source:	–None–	Next Step:	
Surveillance Required?	Yes	Investigation stage *	Prospecting
Assigned to:	bainm Select	Probability (%):	10
Description:			
Surveillance Started?	Yes		

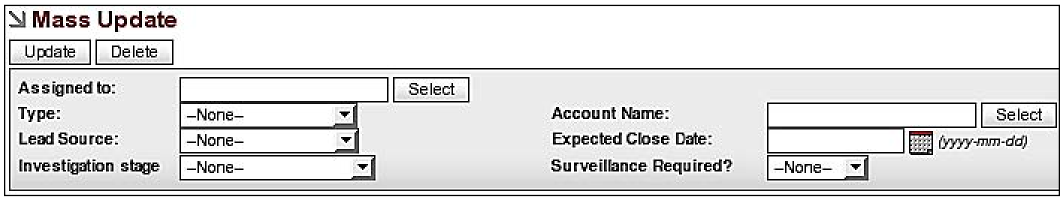
Including Custom Fields in Mass Updates

I'm sure that you're already aware of the mass update function built into SugarCRM. This allows you to view a number of opportunities, cases, project tasks, etc., and then update key fields at the same time. So, for example, if you go to our **Primary Investigations** tab then you'll find that the default mass-update panel for the module looks like:



Assigned to:			
Type:	–None–	Account Name:	
Lead Source:	–None–	Expected Close Date:	
Investigation stage	–None–		

The mass-update function is very useful, and you will, of course, want to use your custom fields with it. In fact, if you've been following the examples in this chapter then you may find that one of the fields is already there:



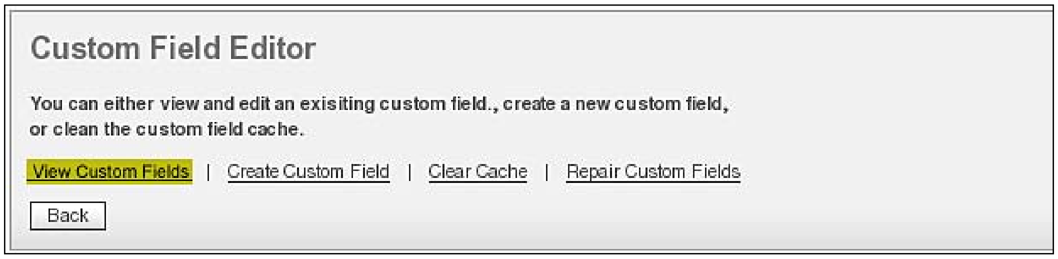
Assigned to:			
Type:	–None–	Account Name:	
Lead Source:	–None–	Expected Close Date:	
Investigation stage	–None–	Surveillance Required?	–None–

So, how do we add fields to the **Mass Update** sub-screen? Actually it is very easy. You may remember that earlier we saw how to create a new field using the Studio. On the Studio screen there's a box named **Mass Update**—tick this and your field will be automatically included in the sub-screen.

We have seen how to create the custom field manually using SQL, and, as you'd expect, it's just a matter of including a value for the appropriate field in the SQL statement:

```
insert into fields_meta_data
   (      id,
          name,
          label,
          help,
          custom_module,
          data_type,
          ext1,
          default_value,
          date_modified,
          mass_update
   )
values
   (       'Opportunitiessurv2_req_c',
          'surv2_req_c',
          'surv2_req_c_10',
          'Surveillance?',
          'Opportunities',
          'enum',
          'surveillance_required_dom',
          'YES',
          now(),
          1
   )
;
```

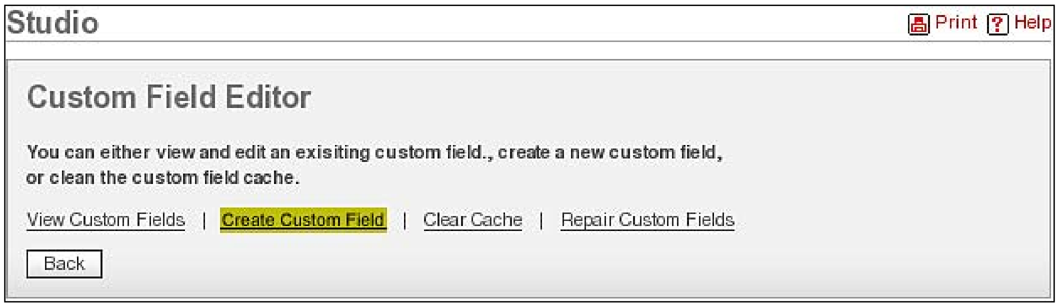
But, what about fields that we've already created? Hopefully, you'll remember that back then we used the Studio to select the screen for creating new fields. This time select **View Custom Fields**:



Custom Field Editor

You can either view and edit an exisiting custom field., create a new custom field, or clean the custom field cache.

View Custom Fields | Create Custom Field | Clear Cache | Repair Custom Fields
Back



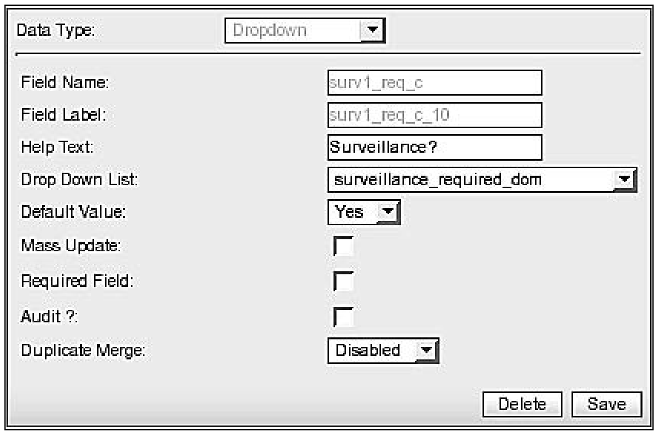
Studio

Custom Field Editor

You can either view and edit an exisiting custom field., create a new custom field, or clean the custom field cache.

Print
Help
View Custom Fields | Create Custom Field | Clear Cache | Repair Custom Fields
Back

Then you can choose the field that you're interested in and check the **Mass Update** box:



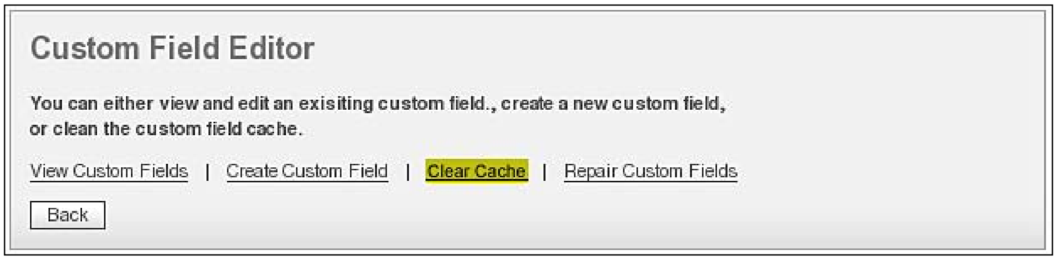
Data Type:	Dropdown
Field Name:	surv1_req_c
Field Label:	surv1_req_c_10
Help Text:	Surveillance?
Drop Down List:	surveillance_required_dom
Default Value:	Yes
Mass Update:	☐
Required Field:	☐
Audit ?:	☐
Duplicate Merge:	Disabled

Or, you can achieve the same by running SQL directly on the database:

```
update fields_meta_data
set mass_update=1
where id='Opportunitiessurv1_req_c';
```

Making Sure that Your Changes are Visible

Occasionally you'll make changes that aren't automatically passed through to all of your users. This is because SugarCRM uses a caching system for any custom fields (similar to, but not the same as, your web browser's caching). So, if you do change the *mass_update* field, make sure that you clear the cache via Studio's **Custom Field Editor**:



Custom Field Editor

You can either view and edit an exisiting custom field., create a new custom field, or clean the custom field cache.

View Custom Fields | Create Custom Field | Clear Cache | Repair Custom Fields
Back

Or you can do it manually by clearing the contents of the `cache/dynamic_fields` directory.

Limitations of the Mass Update

Now, before you run off and create loads of fields, it's worth noting that not all fields can be used for mass updating. The only field types that have these capabilities are:

- Dropdown (as we already know)
- Multiple Select
- Radio Buttons
- Date

That means that you *can't* use:

- Text
- Text Area
- Integer
- Decimal
- Checkbox
- Email
- Web Link
- HTML

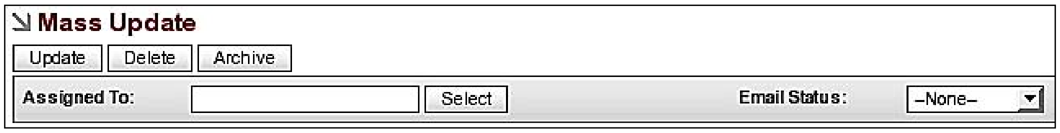
Adding Built-in SugarCRM Fields to the Mass Update

At this point you may be wondering if any other built-in fields can be added into the mass update. The answer is yes, but like custom fields not all types can be used. So, your next questions will be—which ones are they, and how do you do it?

The built-in fields are handled differently to custom fields. In each module directory you'll find a `vardefs.php` file. Each `vardefs.php` file contains the details of fields to be used by the SugarCRM application. Any fields that can be used in the mass update have a *massupdate* property. Not all modules have fields that you can add to the mass update, but if you look in `modules/Emails/vardefs.php` (for example) then you'll find:

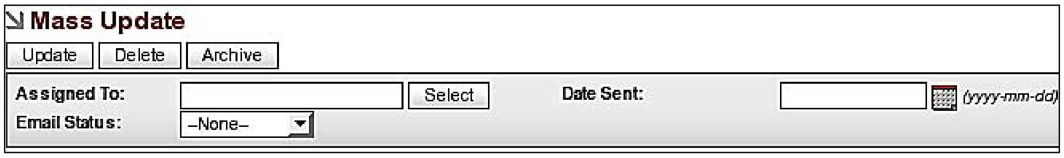
```
'date_start' => array (
                'name' => 'date_start',
                'vname' => 'LBL_DATE',
                'type' => 'date',
                'len' => '255',
                'rel_field' => 'time_start',
                'massupdate'=>false,
        ),
```

So, the standard mass update for emails looks like:



Assigned To:
Email Status: –None–

However, change `massupdate` to `true`, and you'll see:

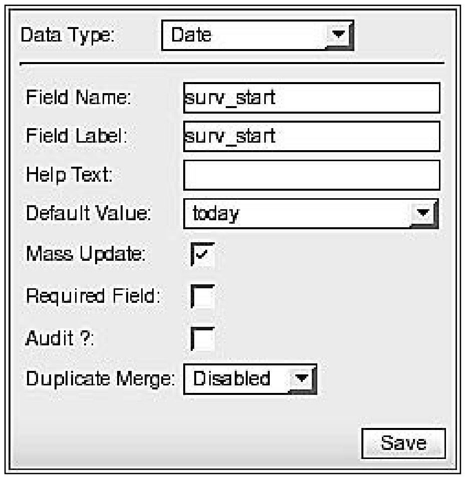


Assigned To:
Date Sent: (yyyy-mm-dd)
Email Status: –None–

As you can see the built-in fields have the same limitations as custom ones when it comes to mass updates—in this case `date_start` can only be included because it is a date field.

Creating other Field Types

We've seen how to create a drop-down field, both by using the Studio and manually, but you're probably wondering how to create other field types. Some (such as radio buttons) follow the same route as drop downs. Other types (such as dates) are simpler to create, since the process is the same except that you don't have to start by creating the drop-down box itself. So, if you want to create a date box (for instance) then go straight to the custom field editor:



Data Type:	Date
Field Name:	surv_start
Field Label:	surv_start
Help Text:	
Default Value:	today
Mass Update:	✓
Required Field:	
Audit ?:	
Duplicate Merge:	Disabled

From there on the process is exactly the same as creating the dropdown that we've already dealt with.

And if you're going to do this manually (using SQL) then you just need to know the data type—obviously in this case it would be *date,* so use an SQL insert query to do that:

```
insert into fields_meta_data
    (      id,
           name,
           label,
           help,
           custom_module,
           data_type,
           date_modified,
           mass_update
    )
values
    (      'Opportunitiessurv_start_c',
           'surv_start_c',
           'surv_start_c_10',
           'Surveillance Start',
           'Opportunities',
           'date',
           now(),
           1
    )
;
```

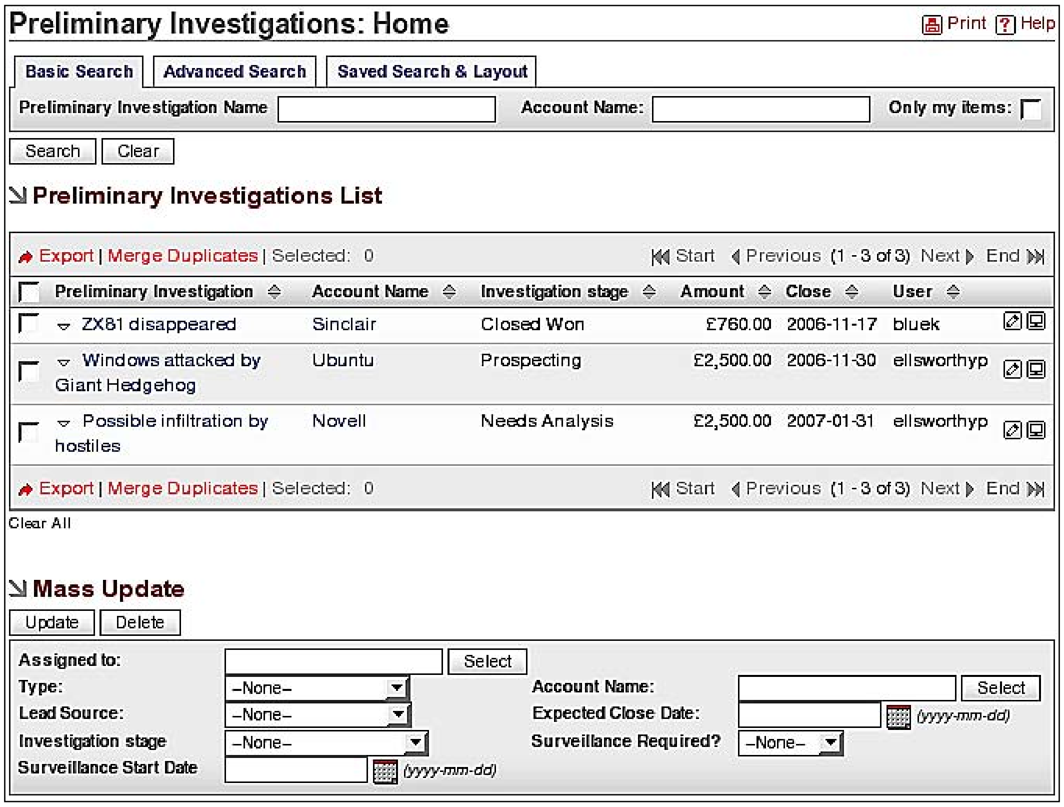
Having inserted the data, don't forget to add a new field to `opportunities_ctsm`, and a simple SQL alter statement will do that:

```
alter table opportunities_cstm add surv_start_c date;
```

Update `custom/modules/Opportunities/ language/en_us.lang.php` :

```
$mod_strings['surv_start_c_10'] = "Surveillance Start Date";
```

And once you've used the Studio or edited the `.html` files you'll have a date field on your screen as well as the dropdowns that we've already created—and Korora's life will suddenly become much easier:


Preliminary Investigations: Home

Preliminary Investigation Name
Account Name:
Only my items:

Preliminary Investigations List

Export | Merge Duplicates | Selected: 0 — Start Previous (1 - 3 of 3) Next End

Preliminary Investigation	Account Name	Investigation stage	Amount	Close	User
ZX81 disappeared	Sinclair	Closed Won	£760.00	2006-11-17	bluek
Windows attacked by Giant Hedgehog	Ubuntu	Prospecting	£2,500.00	2006-11-30	ellsworthyp
Possible infiltration by hostiles	Novell	Needs Analysis	£2,500.00	2007-01-31	ellsworthyp

Export | Merge Duplicates | Selected: 0 — Start Previous (1 - 3 of 3) Next End

Clear All

Mass Update

Assigned to:	Select		
Type:	–None–	Account Name:	Select
Lead Source:	–None–	Expected Close Date:	(yyyy-mm-dd)
Investigation stage	–None–	Surveillance Required?	–None–
Surveillance Start Date	(yyyy-mm-dd)		

Field Data Types

So far we've looked at the dropdown (enum) and date field types. Now, if you are going to use the Studio to create your new fields then all you have to do is select the data type from the drop-down list. However, if you want to create the fields manually then you'll need to know what to enter in the `data_type` field in `fields_meta_data`:

- Text—varchar
- Text Area—text
- Integer—int
- Decimal—float
- Checkbox—bool
- Email—email

- Dropdown—enum
- Multiple Select—multienum
- Radio Buttons—radioenum
- Date—date
- Web Link—url
- HTML—html

With all this information at your fingertips you can now create whatever new fields your users require, and you can do it by using the methods that you feel most comfortable with—whether it be through the Studio, or by editing files and using SQL on the command line.

Summary

In this chapter we've seen how to create and make use of our own custom fields in SugarCRM modules. We've also seen how to include some of our fields (and some of the built-in SugarCRM fields) in the mass-update sub-panels.

We saw that the process for creating a custom field manually is the same as in the Studio; it's just that you'll be doing all of the things that the Studio would do for you.

In Chapter 4 we'll start to look at SugarCRM in more depth as we start to understand the structure of the application itself.

4
Interfacing with SugarCRM

Hopefully, you are feeling very confident about customizing SugarCRM. Therefore, this seems an appropriate point to take a step back from the customization process, and have a deeper look into the structure of SugarCRM itself. So, what we'll do now is:

- See how the SugarCRM application is put together as we examine the user and data interfaces in this chapter.
- In the next chapter we'll see how the SugarCRM database is put together.

What Have we Learned so Far?

Over the past three chapters we've actually learned quite a bit about the application architecture. To start with:

- The application consists of a number of PHP files on a web server.
- The application requires a database in the background.

So, if we think about the PHP files we know that:

- We always access the SugarCRM application via a central PHP file—`index.php`.
- We have the `custom` directory for storing any language customizations.
- We have a `themes` directory where we store the files for customizing colors, fonts, icons, and images for the application.
- SugarCRM consists of a number of `module` directories, which provide the actual SugarCRM functionality. They're all stored in the `modules` directory.

Let us just remind ourselves about the files that each of these directories needs to contain.

The Include Directory

The `include` directory contains module-independent files such as:

- `modules.php`

The Custom Directory

The `custom` directory contains:

- `custom/include/language/en_us.lang.php`
- `custom/modules/<module>/language/en_us.lang.php`

The Themes Directory

The `themes` directory contains a directory for each theme to be used by your application. Each of these directories must have:

- `config.php`
- `style.css`

The Modules Directory

The `modules` directory contains a directory for each module to be used by your application. Each of these directories *must* have:

- `index.php`
- `Forms.php`
- `<module name>.php`
- `language/en_us.lang.php`

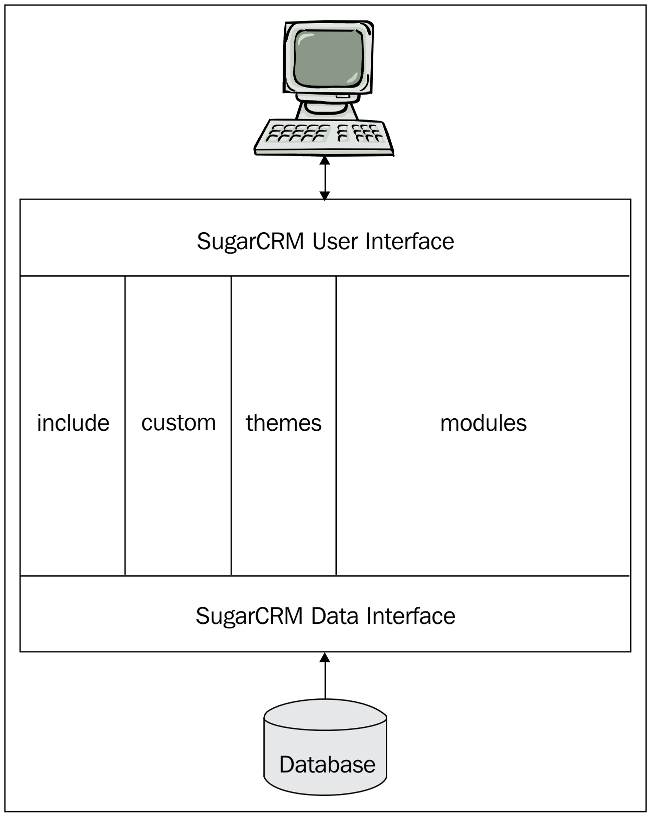
We've also learned that the SugarCRM modules have an interface to the database, and in particular:

- Custom fields can be defined on the database, but the application caches details about them in `cache/dynamic_fields`.
- Each module has its own data field definitions in a file named `vardefs.php`.

From all of this we can already build ourselves a general picture of the SugarCRM application architecture.

Overview of the SugarCRM Application Architecture

The SugarCRM application architecture is simple, but effective:



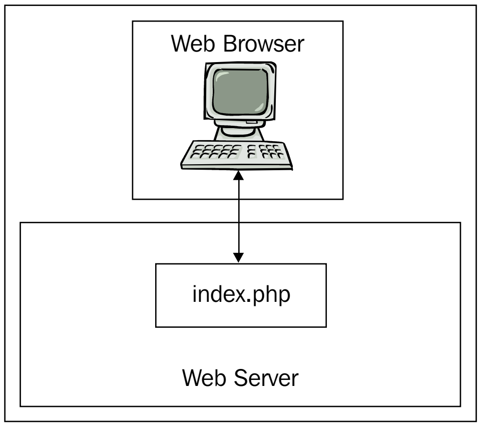
As you can see form the diagram above, and, as you may well have worked out for yourself already from the last three chapters, our users use their computers (i.e. their web browsers) to access the Sugar User Interface—this then governs all interactions between the user and the SugarCRM functionality.

You will also see that there is another interface (the SugarCRM data interface) between the SugarCRM functionality and the underlying database (as you would expect).

In the remainder of the chapter we'll concentrate on the user and data interfaces, and then in Chapter 5 we'll look at the database itself.

The SugarCRM User Interface

You probably have worked out that the SugarCRM user interface is actually generated by the `index.php` file in your main SugarCRM directory:



So, there's nothing here that you don't already know. It is, therefore, worth having a look at what the interface actually does for us.

The user interface (or if you prefer—the UI layer):

- Decodes information posted (via the HTML forms) to the SugarCRM forms
- Authenticates users' log on details and active sessions
- Provides a wrapper around the modules files

In other words all a user has to do is to call up `index.php` on the web server, and it will do all of the work for them.

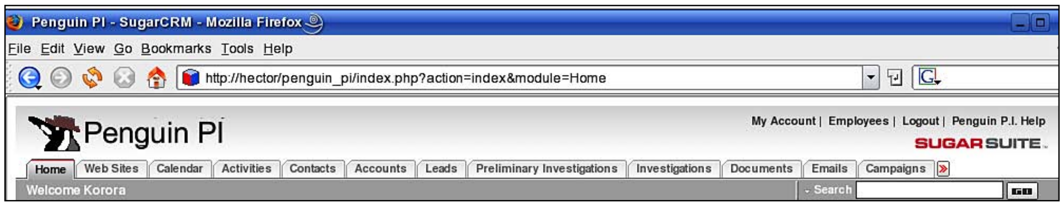
Calling Modules

Having identified that `index.php` handles all of our interactions with SugarCRM, it's worth just looking at how we can use the user interface to guide us to particular modules, and, perhaps more importantly, how we can use it to carry out actions.

There are two key parameters that you can pass to `index.php`:

- **module**—This, obviously, is the module that you want to call. However, to be completely correct, it is the *directory* in which the module is stored.
- **action**—This is the PHP file in the module directory to be used. By default its `index.php` (i.e. the `index.php` file in the module directory, not `index.php` in the top level of the web server). However, you can call other PHP files in the directory.

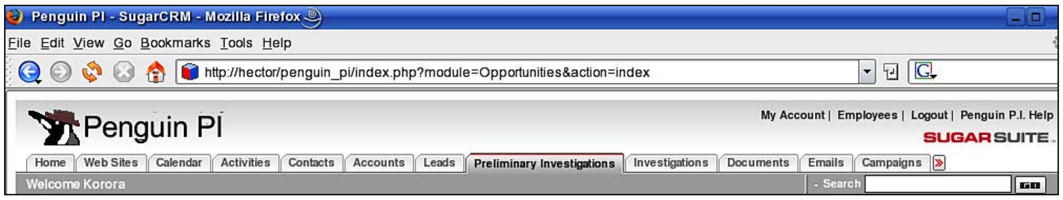
So, let us imagine Korora logging on—she'll start by typing in the SugarCRM URL (in her case `http://hector/penguin_pi`) however, once she's finished typing in her user name and password then she'll see:


Penguin PI - SugarCRM - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Penguin PI
My Account | Employees | Logout | Penguin P.I. Help
SUGARSUITE
Home | Web Sites | Calendar | Activities | Contacts | Accounts | Leads | Preliminary Investigations | Investigations | Documents | Emails | Campaigns
Welcome Korora
Search

http://hector/penguin_pi/index.php?action=index&module=Home

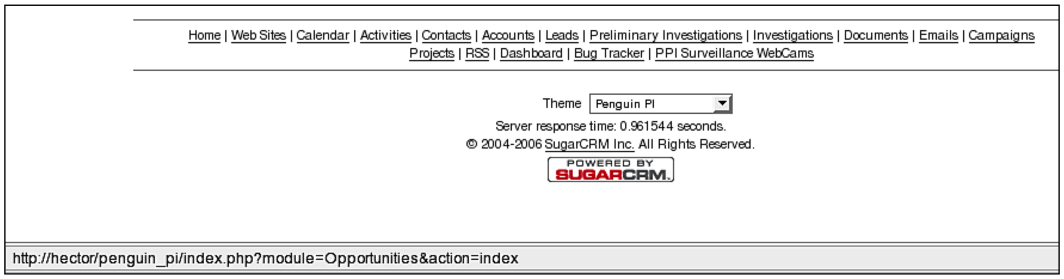
As you can see the user interface has set the module to **Home**, and the action to **index**.

If Korora then clicks on one of the tabs (for example **Preliminary Investigations**) then the user interface handles this change for her, and you can see that this has been done by setting the module to **Opportunities** and the action to **index**:


Penguin PI - SugarCRM - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Penguin PI
My Account | Employees | Logout | Penguin P.I. Help
SUGARSUITE
Home | Web Sites | Calendar | Activities | Contacts | Accounts | Leads | Preliminary Investigations | Investigations | Documents | Emails | Campaigns
Welcome Korora
Search

http://hector/penguin_pi/index.php?module=Opportunities&action=index

In fact, you'll find that each of the tab titles is actually a link, and each of the links simply passes the appropriate module and index back to the main `index.php`. For example, if you place the mouse pointer over the **Preliminary Investigations** tab title then you'll see that the link address is:


Home | Web Sites | Calendar | Activities | Contacts | Accounts | Leads | Preliminary Investigations | Investigations | Documents | Emails | Campaigns
Projects | RSS | Dashboard | Bug Tracker | PPI Surveillance WebCams
Theme Penguin PI
POWERED BY SUGARCRM

Server response time: 0.961544 seconds.
© 2004-2006 SugarCRM Inc. All Rights Reserved.

http://hector/penguin_pi/index.php?module=Opportunities&action=index

We can now use this knowledge to manage the way in which we use SugarCRM. For example, if we return to the module tab that we created in Chapter 2, then we can change it so that it contains a list of key 'jobs' to be done.

We could start by changing the title of the module by going to `custom/include/language`, editing `en_us.lang.php`, and changing:

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

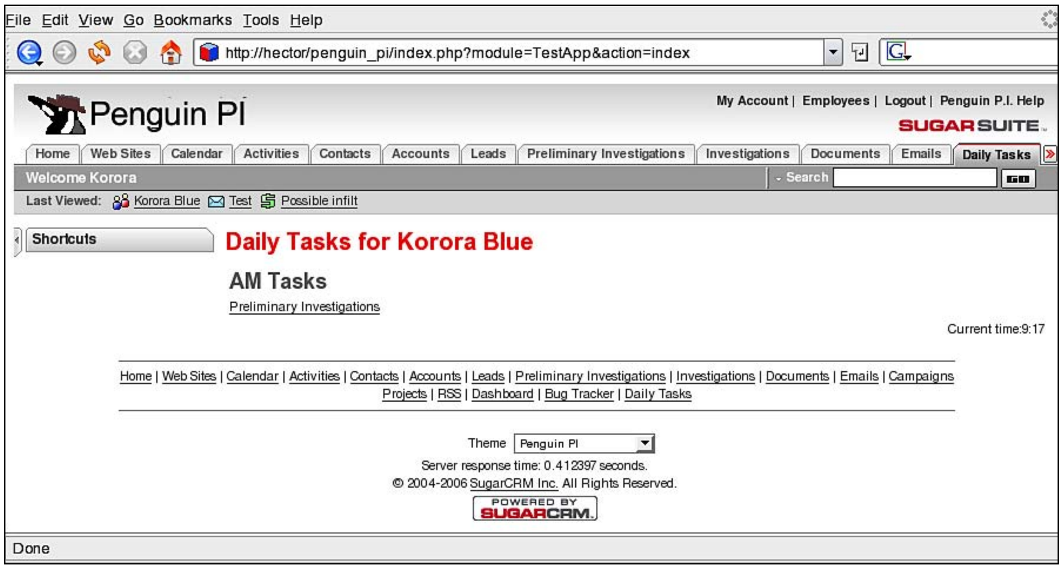
to:

```
'TestApp' => 'Daily Tasks',
```

Next we can think about editing `modules/TestApp/index.php` so that Korora's daily tasks are displayed. And, to make it even more useful, we can make use of the `strftime` function (which formats the local time) to display different tasks at different times:

```
<?php
global $current_user;
#Get the local time (from the server)
$h = strftime("%H");
$m = strftime("%M");
?>
<h1>Daily Tasks for
<?php
#Display the users name (to be more personal just use the first name)
echo $current_user->first_name . " " . $current_user->last_name;
?></h1>

<table width=100%>

<?php
#Display tasks for the morning
if ( $h >= 9) { ?>
<tr><td><h2>AM Tasks</h2></td></tr>
<tr><td>
<a href=index.php?module=Opportunities&action=index>
Preliminary Investigations
</a></td></tr>
<?php } ?>

<?php
#Display tasks for the afternoon
if ( $h >= 12) { ?>
<tr><td><hr></td></tr>
<tr><td><h2>PM Tasks</h2></td></tr>
<tr><td>
<a href=index.php?module=Cases&action=index>
Investigations
</a></td></tr>
<?php } ?>
<tr><td align=right>
<?php
#And finally show the current (server) time
echo "Current time:" . $h . ":" . $m; ?>
</td></tr>
</table>
```

The end result (in the morning) is:


File Edit View Go Bookmarks Tools Help
http://hector/penguin_pi/index.php?module=TestApp&action=index
Penguin PI
My Account | Employees | Logout | Penguin P.I. Help
SUGARSUITE
Home Web Sites Calendar Activities Contacts Accounts Leads Preliminary Investigations Investigations Documents Emails Daily Tasks
Welcome Korora
Search
GO
Last Viewed: Korora Blue Test Possible infilt
Shortcuts

Daily Tasks for Korora Blue

AM Tasks

Preliminary Investigations

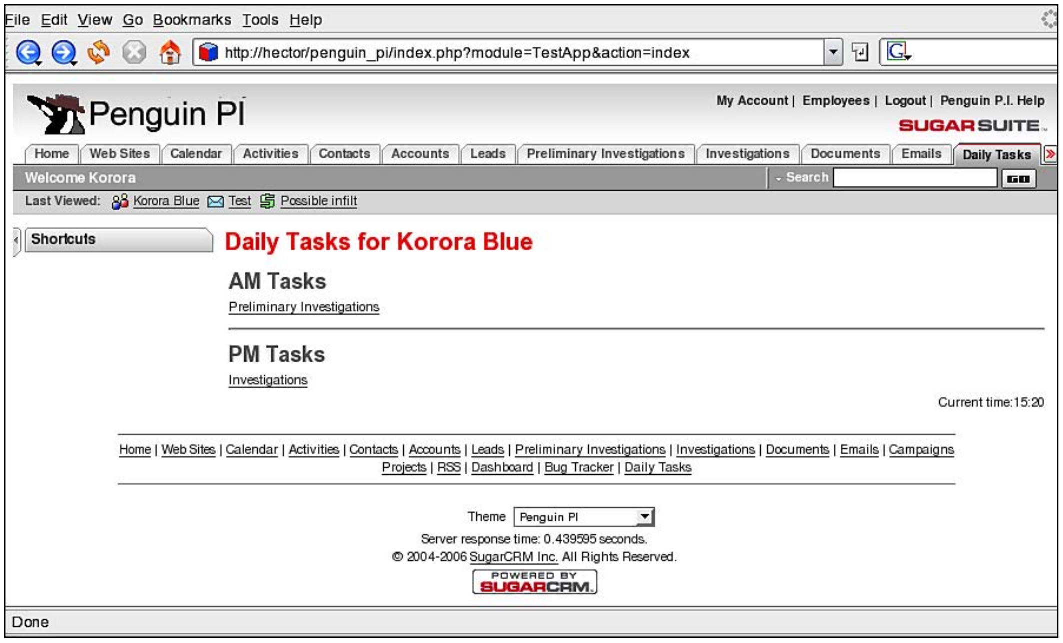
Current time:9:17

Home | Web Sites | Calendar | Activities | Contacts | Accounts | Leads | Preliminary Investigations | Investigations | Documents | Emails | Campaigns
Projects | RSS | Dashboard | Bug Tracker | Daily Tasks
Theme Penguin PI

Server response time: 0.412397 seconds.

© 2004-2006 SugarCRM Inc. All Rights Reserved.

And in the afternoon:


File Edit View Go Bookmarks Tools Help
http://hector/penguin_pi/index.php?module=TestApp&action=index
Penguin PI
My Account | Employees | Logout | Penguin P.I. Help
SUGARSUITE
Home Web Sites Calendar Activities Contacts Accounts Leads Preliminary Investigations Investigations Documents Emails Daily Tasks
Welcome Korora
Search
GO
Last Viewed: Korora Blue Test Possible infilt
Shortcuts

Daily Tasks for Korora Blue

AM Tasks

Preliminary Investigations

PM Tasks

Investigations

Current time:15:20

Home | Web Sites | Calendar | Activities | Contacts | Accounts | Leads | Preliminary Investigations | Investigations | Documents | Emails | Campaigns
Projects | RSS | Dashboard | Bug Tracker | Daily Tasks
Theme Penguin PI

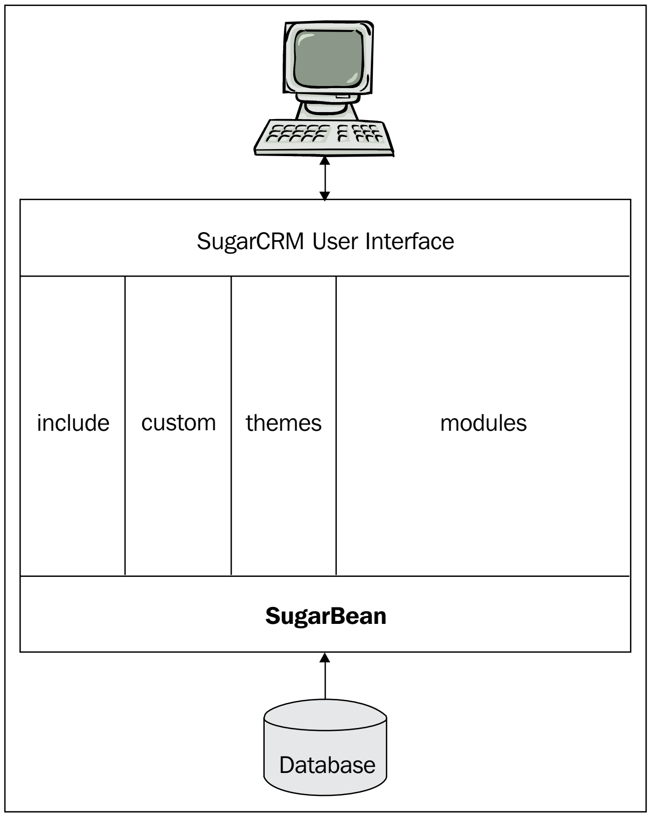
Server response time: 0.439595 seconds.

© 2004-2006 SugarCRM Inc. All Rights Reserved.

Now that we've had a look at the SugarCRM user interface it's time to move on to the SugarCRM Data Interface—otherwise known as SugarBean.

SugarBean—The SugarCRM Data Interface

We'll be looking at the structure of the database in Chapter 5, but it's possible that you will never have to access it, and that's because of SugarCRM's SugarBean:



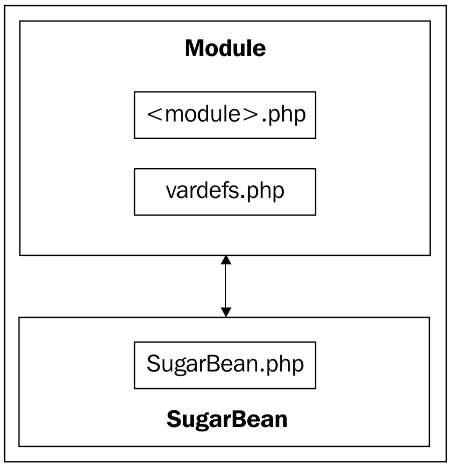
So, what is the SugarBean? At its simplest level it's another PHP file, but it does a very important job—it's a high-level API that allows you to manipulate your business data without having to worry (too much) about what's going on in the database. The SugarBean:

- Is the base class for the entire SugarCRM business object that you need to use. This means that the **Opportunity** object (for example) is just an extension of the SugarBean.
- It supplies all of the key functions for your business objects, such as creating records, retrieving records, updating and deleting.

And, as you would expect, the SugarBean consists of a set of PHP files.

The SugarBean Files

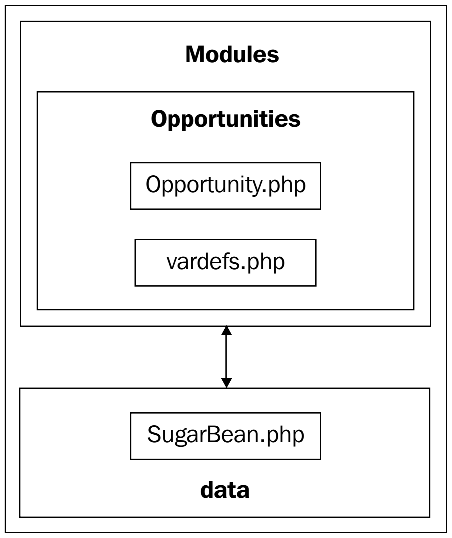
As we've already learned the SugarBean is the data interface between our modules and our database, and it consists of a number of PHP files on the SugarCRM web server:



You can see that there are three key files for the SugarBean:

- `SugarBean.php`—This is located in the `data` directory and is (as we've already learned) the base class file.
- `vardefs.php`—This is the schema for the business object. There is one for each module that uses the SugarBean.
- `<module>.php`—Each module using the SugarBean must contain this file, and it is used to extend the base class for the particular module. It is not actually named the same as the module, but takes the singular form, e.g. the **Opportunities** module would contain `Opportunity.php`.

So, if we continue to think about Opportunities for a moment then we'd see the following set up:



In order to better understand the SugarBean let's start by examining `vardefs.php` in a little bit more detail.

vardefs.php

You'll hopefully remember that we have already worked with the `vardefs.php` file. In Chapter 3 we saw that it is possible to add SugarCRM fields into the mass update sub-screen by editing this file—we modfied `modules/Emails/vardefs.php`, and updated the `massupdate` property:

```
'date_start' => array (
                'name' => 'date_start',
                'vname' => 'LBL_DATE',
                'type' => 'date',
                'len' => '255',
                'rel_field' => 'time_start',
                'massupdate'=>true,
                'comment' => 'Date of last inbound email check', ),
```

Last time we just made the changes and moved on, but this time we'll look each of the properties, although now that you know that this is the database schema file, I'm sure that you can work out most of the details yourself.

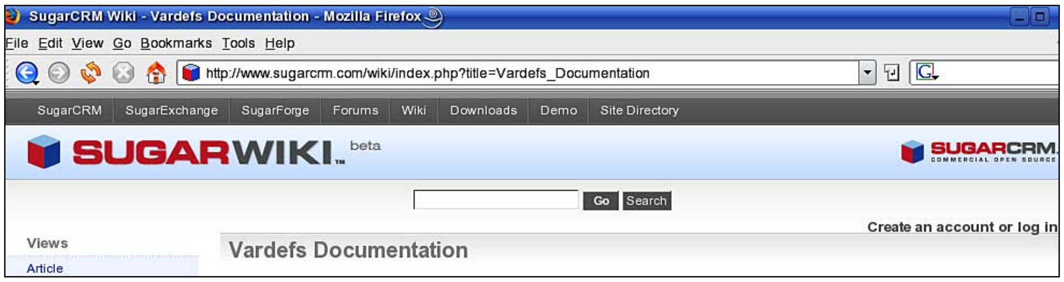
In case you haven't worked out what's going on here—this array represents a single field in the database schema, and you'll see that each field has a set of parameters. In this case `'date_start'` has:

- `'name'`—Unsurprisingly this is the name of the field.
- `'vname'`—The field label ID for the module's `en_us.lang.php` file.
- `'type'`—The data type of the property.
- `'len'`—The length of the field.
- `'rel_field'`—Since this is a date field it has a related time field.
- `'massupdate'`—You already know what this does (but in case you've forgotten—you set this if you want to be able to update a group of records all at the same time).
- `'comment'`—That would be a comment then.

There are actually a lot more parameters that are available to you, but this is still a fairly fluid area of SugarCRM, and these are liable to change. For that reason I'm not going to give you an exhaustive list. Instead it's time to look at some of SugarCRM's on-line documentation.

vardefs On-line Documentation

- You'll find current details about `vardefs` at `http://www.sugarcrm.com/wiki/index.php?title=Vardefs_Documentation`:


SugarCRM Wiki - Vardefs Documentation - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.sugarcrm.com/wiki/index.php?title=Vardefs_Documentation
SugarCRM SugarExchange SugarForge Forums Wiki Downloads Demo Site Directory
SUGARWIKI beta
SUGARCRM COMMERCIAL OPEN SOURCE
Go Search
Create an account or log in
Views
Article
Vardefs Documentation

Having just said that you should refer to the on-line SugarWiki to obtain an up to date list of all of the available parameters for the vardef fields, it is still worth looking at one parameter—the type.

vardefs Field Types

There are a number of different field types available to you for use in the data schema—some of which you'll recognize if you've worked databases, and some of which are specific to SugarCRM:

- `'assigned_user_name'`—Contains a SugarCRM user name
- `'blob'`—the **B**inary **L**arge **OB**ject—Normally used when you want to store a large amount of data in a single field
- `'bool'`—A boolean value, although it uses a 1 or 0 rather than true or false and in fact it maps to tiny integer on a MySQL database
- `'char'`—An array of characters—although you'd never use this, since varchar is available
- `currency`
- `'date'`
- `'datetime'`
- `'email'`
- `'enum'`—Enumeration—normally used for dropdown lists
- `'float'`—A decimal number—normally used to store currency
- `'id'`—A 36 character SugarCRM ID number
- `'int'`—Integer
- `'link'`—A relationship link
- `'nondb'`—A derived value—not from database (and not technically a type), which could come from a PHP function
- `'num'`—Interesting one—this is actually stored in the MySQL database as a varchar
- `'phone'`—a phone number
- `'relate'`—Related Bean, i.e. related to a field in another table
- `'text'`—text field. Basically a 'char' that holds 65,535 characters
- `'varchar'`—A variable sized string, the length of which is set by the 'len' field

So, nothing really contentious here—the list contains all the common field types that you will need for your project.

The Complete vardefs File

So far we've only looked at an individual field within `vardefs.php`; however, that's not the end of the story. Each field is defined as an array of parameters, but these are just part of a larger array of fields, and are stored in another array—the dictionary:

```
$dictionary['Opportunity'] = array(
  'table' => 'opportunities',
  'audited'=>true,
  'unified_search' => true,
  'duplicate_merge'=>true,
  'comment' => 'An opportunity is the target of selling activities',
  'fields' => array )
```

We're now going to leave `vardefs.php` again, but we will be coming back to it—in Chapter 8, when we'll be looking at developing a complete module. In the meantime we'll have a quick look at the `<module>.php` file.

The <module>.php File

Although I've referred to the `<module>.php` file don't forget that the file must actually be given the singular name for the module, so for Opportunities use `Opportunity.php`, for Emails use `Email.php`, and so on.

Our file contains a class that extends the basic SugarBean class (`SugarBean.php`), and it's used to define:

- Variables mapped to the database schema (`vardefs.php`)
- Any additional functionality specific to the particular business object

The class files all have the same format, and so if we look at `Opportunity.php`, we'll see that the base class file is loaded, along with any others that are required:

```
require_once('data/SugarBean.php');
require_once('modules/Contacts/Contact.php');
require_once('modules/Tasks/Task.php');
require_once('modules/Notes/Note.php');
require_once('modules/Calls/Call.php');
require_once('modules/Leads/Lead.php');
require_once('modules/Emails/Email.php');
require_once('include/utils.php');
```

And then the new class is defined:

```
class Opportunity extends SugarBean
{
  var $field_name_map;
  // Stored fields
  var $id;
  var $lead_source;
  var $date_entered;
  var $date_modified;
  var $modified_user_id;
}
```

And, of course, it will need a constructor function:

```
function Opportunity()
{
   parent::SugarBean();
   global $sugar_config;
   if(!$sugar_config['require_accounts'])
   {
      unset($this->required_fields['account_name']);
   }
   global $current_user;
 }
```

As well as any extra functions required for the Opportunity class:

```
function get_list_view_data()
{
  global $locale, $current_language, $current_user, $mod_strings,
  $app_list_strings, $sugar_config;
  $app_strings = return_application_language($current_language);
  require_once('modules/Currencies/Currency.php');
  $temp_array = $this->get_list_view_array();
  #Set the sales state
  $temp_array['SALES_STAGE'] =
    empty($temp_array['SALES_STAGE']) ? '' :
                                   $temp_array['SALES_STAGE'];
  #Set the ammount
  $temp_array['AMOUNT'] = currency_format_number($this->amount);
  #Set the name
  $temp_array["ENCODED_NAME"]=$this->name;
  #Return the result
  return $temp_array;
}
```

Now, just like `vardefs.php`, we're going to leave the class file behind for now, and return to it in Chapter 8. However, you, no doubt, want to see the SugarBean in action—so we'll turn our attention to SugarCRM's logic hooks.

SugarBean in Action—SugarCRM's Logic Hooks

You may not have heard of logic hooks before—if not then, quite simply, they provide us with the ability to add in our own custom business logic into the SugarCRM applications. These logic hooks may take the form of some kind of validation, or they may take the form of a more involved business operation.

If we look at the Penguin P.I. office for a moment, we might see Korora sat at her desk. One of her tasks is to evaluate any new preliminary investigations and then assign them to someone. However, when she does this she must ensure that:

- Only people with certain roles can receive preliminary investigations.
- Each preliminary investigation must got to the correct office covering the geographical region in which the investigation is to be carried out.
- Where more than one person qualifies for receiving the preliminary investigation then the person with the least amount of work must be chosen.

We've therefore got two options:

- Let Korora work it all out for herself—regardless of how long it's going to take.
- Add a logic hook that will do all of this automatically.

We're not going to do all of that at the moment; we'll save that for Chapter 9 when we'll look at developing custom workflows within SugarCRM. However, what we will do is create a logic hook that records changes in assigned users for any Opportunity.

Now, if you're already used to working with databases such as Oracle then you'll be used to the concept of a *trigger*. Triggers are simply pieces of code that are run when particular events (such as update or insert) occur on the database—and that's exactly what the logic hook does—it runs a PHP file when the SugarBean carries out certain database operations. These key events are:

- `after_retrieve`
- `before_save`
- `before_delete`
- `after_delete`
- `before_undelete`
- `after_undelete`

So, in this case we want the logic hook to operate on the `before_save` event. We're also going to be writing to a log file in this instance, and so the first thing to do is to write the code for that. We want to keep this separate from the standard SugarCRM code and so we'll place it `custom/include` and call the file `penguin_pi_functions.php`:

```
#File:  penguin_pi_functions.php
<?php
function WriteToLogFile($strText) {
  $File = '/www/penguin_pi/test.log'; #Choose a suitable file name
  $Handle = fopen($File, 'a'); #Open the file
  $Data = $strText . "\n"; #Add a carriage return to the text
  if($Handle) { // avoid further errors on file access failure
  fwrite($Handle, $Data); #Write the text to the file
  fclose($Handle); #Close the file
  }
}
?>
```

[And just a note—you may need to manually create (and set the permissions for) the log file before you run the code.]

Next we'll need the code file that's going to be run by the logic hook. Again, we'll put it in `custom/include`, but this time we'll call the file `ppi_prelim_change.php`, and it's another class file:

```
#File:  ppi_prelim_change.php
<?php
require_once('data/SugarBean.php');
require_once('modules/Opportunities/Opportunity.php');
require_once('custom/include/penguin_pi_functions.php');

class ppi_prelim_change {
  function ppi_prelim_change (&$bean, $event, $arguments) {
    global $sugar_config;

    if ($bean->fetched_row['assigned_user_id']!=
                                          $bean->assigned_user_id)
    {
      #Obtain the information for the old user
      $old_user = new User(); #Create a user object
      $old_user->retrieve($bean->fetched_row['assigned_user_id']);
      $old_assigned_user_name =
      $old_user->first_name.' '.$old_user->last_name;

      #Obtain the information for the new user
      $new_user = new User();
      $new_user->retrieve($bean->assigned_user_id);
      $new_assigned_user_name =
      $new_user->first_name.' '.$new_user->last_name;
      #Write the information to the log file
      WriteToLogFile
        ($old_assigned_user_name . " -> " . $new_assigned_user_name );
    }
  }
}
?>
```

You'll notice from the code that both the current (i.e. changed) data and the original data are available to the function by making use of the *$bean* object:

- `$bean->assigned_user_id` provides the new user ID
- `$bean->fetched_row['assigned_user_id']` provides the old user ID.

You'll also notice that the code makes use of:

- The SugarBean base class file (`SugarBean.php`)
- The Opportunity class file (`Opportunity.php`)
- Our own custom functions file (`penguin_pi_functions.php`)

One very useful function worth taking note of is *retrieve*—you'll see from the code that this obtains the assigned user details with the minimum of effort on your part.

Finally, we just need to create the logic hook file itself. However, unlike the last two files, this *must* be placed in a specific location. You might expect that the Opportunities logic hook should be placed in `modules/Opportunities`, and you'd be nearly correct—it actually needs to be be placed in `custom/modules/Opportunities`, and it also has to be named `logic_hooks.php`:

```
#File: logic_hooks.php
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
$hook_array = Array(); #Create an array

$hook_array['before_save'] = Array(); #Create a sub-array

#Write the required information to the array
$hook_array['before_save'][] = Array(1, 'ppi_prelim_change',
  'custom/include/ppi_prelim_change.php',
  'ppi_prelim_change', 'ppi_prelim_change');
?>
```

If you examine the code then you'll see that we have to define an array (called `$hook_array`). This array then contains a sub-array, and it's this array that defines the logic hook itself.

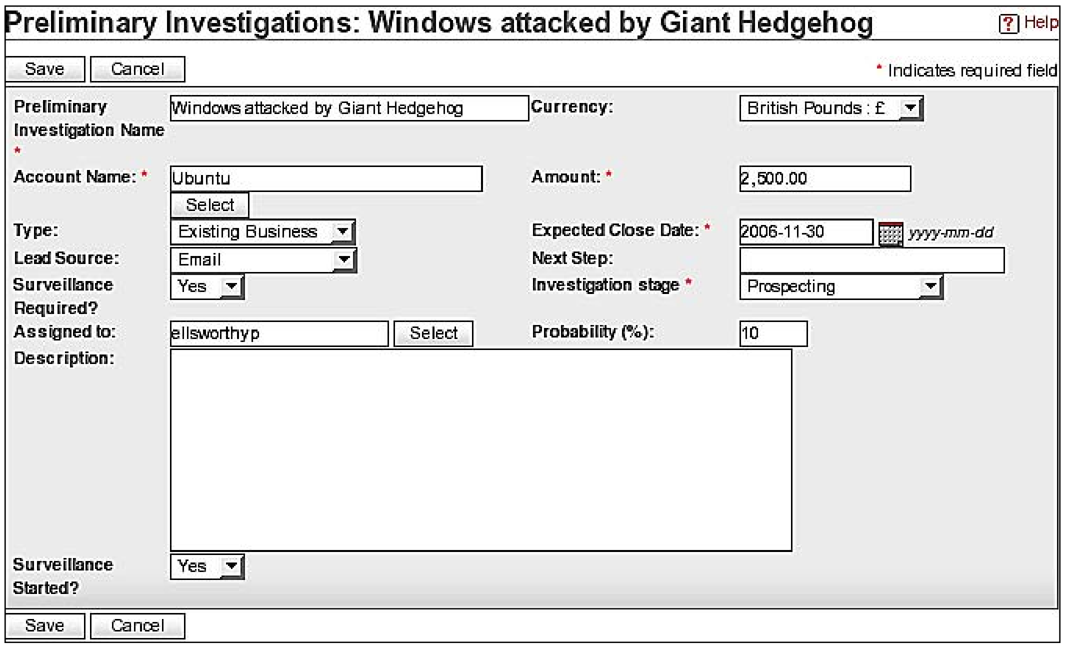
You'll notice that the array has the same name as the event on which the trigger is to be set, and it has a number of elements:

- Logic hook order—we can define a number of hooks in the same file, and this element defines the order in which they should be used.
- Name—this is just a placeholder to store the name of the hook.
- PHP code file location.
- PHP class to be called.
- PHP function that is to be run by the hook.

If you have these three files in place then your logic hook is up and running, and just waiting for your users to do something.

The End Result of Using the Logic Hook

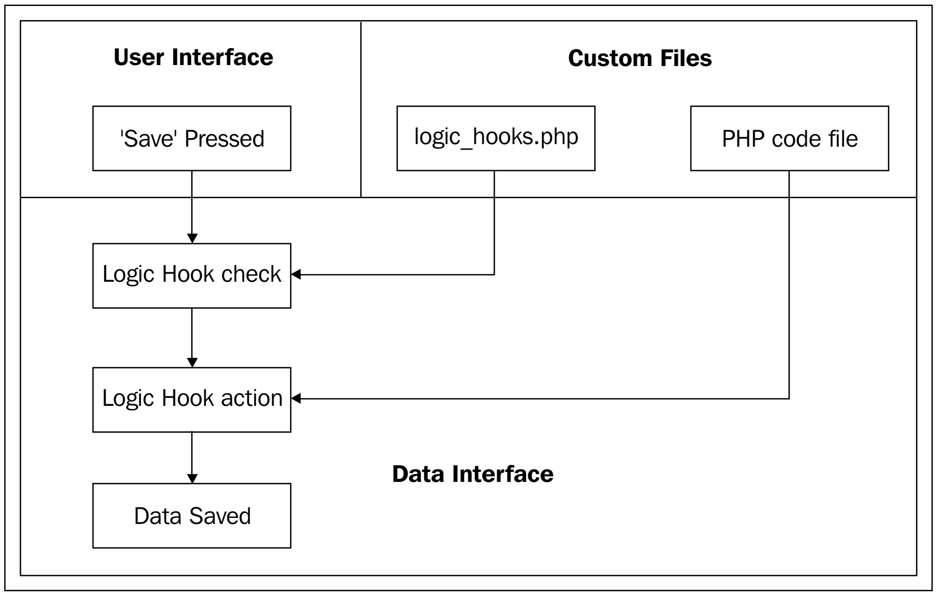
Back to Korora—she now needs to edit one of the preliminary investigations and change the assigned user:



Preliminary Investigations: Windows attacked by Giant Hedgehog

Preliminary Investigation Name *	Windows attacked by Giant Hedgehog	Currency:	British Pounds : £
Account Name: *	Ubuntu	Amount: *	2,500.00
Type:	Existing Business	Expected Close Date: *	2006-11-30 yyyy-mm-dd
Lead Source:	Email	Next Step:	
Surveillance Required?	Yes	Investigation stage *	Prospecting
Assigned to:	ellsworthyp	Probability (%):	10
Description:			
Surveillance Started?	Yes		

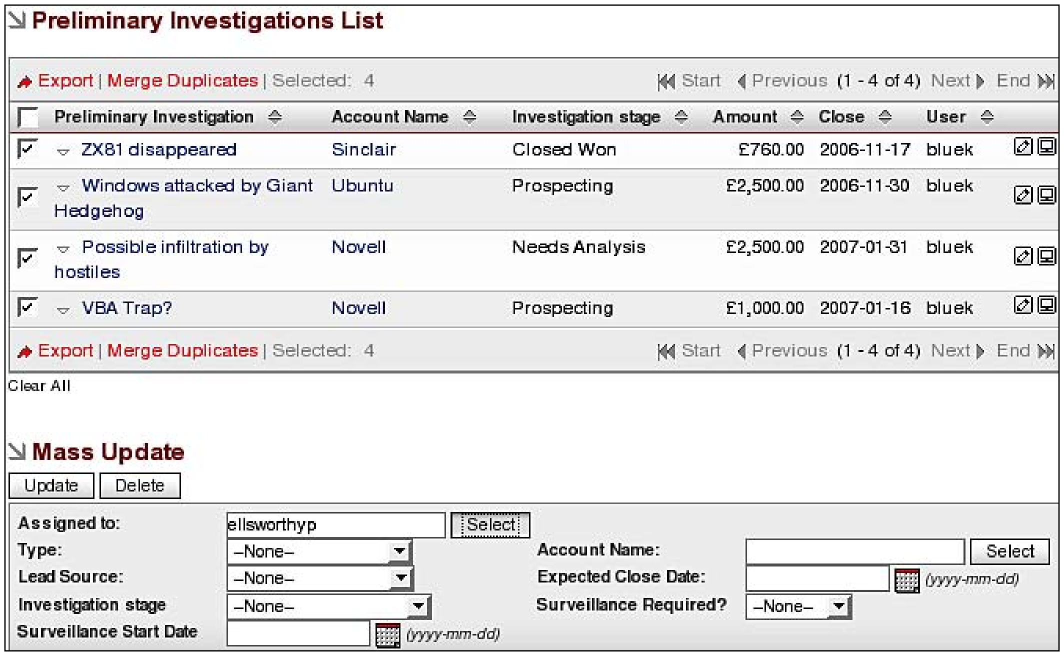
When she clicks the **Save** button then she'll be unaware of any differences in SugarCRM; however, in the background something will have changed:



Although your users will see nothing, SugarCRM will check to see if a Logic Hook exists. If it does then the associated PHP code file will be run, and (in this case) the data will be saved. And, of course, if Korora was to look on the web server she'd find that the `www/penguin_pi/test.log` file would contain a new entry:

```
Korora Blue -> Pygoscelis Ellsworthy
```

And this doesn't only work for individual instances—this will also work for the mass update. So, if you look back on the tab screen and actually carry out a mass update:



Preliminary Investigations List

Export | Merge Duplicates | Selected: 4 — Start · Previous (1 - 4 of 4) Next · End

	Preliminary Investigation	Account Name	Investigation stage	Amount	Close	User
✓	ZX81 disappeared	Sinclair	Closed Won	£760.00	2006-11-17	bluek
✓	Windows attacked by Giant Hedgehog	Ubuntu	Prospecting	£2,500.00	2006-11-30	bluek
✓	Possible infiltration by hostiles	Novell	Needs Analysis	£2,500.00	2007-01-31	bluek
✓	VBA Trap?	Novell	Prospecting	£1,000.00	2007-01-16	bluek

Export | Merge Duplicates | Selected: 4 — Start · Previous (1 - 4 of 4) Next · End

Clear All

Mass Update

Update | Delete

Assigned to:	ellsworthyp [Select]		
Type:	–None–	Account Name:	[Select]
Lead Source:	–None–	Expected Close Date:	(yyyy-mm-dd)
Investigation stage	–None–	Surveillance Required?	–None–
Surveillance Start Date	(yyyy-mm-dd)		

You'll find that the logic hook fires for each record that you update.

Obviously this has been a very simple example, but we'll look at this more extensively in Chapter 9, and then we'll see how to use logic hooks as part of a workflow system.

Summary

In this chapter we've spent some time looking at the SugarCRM user interface and the data interface. We've seen how to use these effectively within our SugarCRM customizations.

The SugarBean is SugarCRM's high-level API that handles all our interactions with the database.

Logic hooks enable us to add in our own business logic into SugarCRM with the minimum effort. They are similar to database triggers—except that it's the SugarBean that does all the work and not the database.

Thus we've looked at the interfaces, and how to use them effectively in our customizations. We'll look at the files in more detail in Chapter 8 (when we'll develop a complete module), and Chapter 9 (when we'll look at custom workflows). However, before we do all that we'll examine the structure of the database itself.

5 SugarCRM Database Schematics

All our work so far has mainly been with the SugarCRM PHP files on the web server, and we've only dipped a little bit into the database. However, we're going to rectify that in Chapters 5 and 6.

The aim of this chapter is to map out the key areas of the database, and to see how the tables of the database are linked together. And so, we won't actually be doing practical work in this chapter—this is all reference material.

The areas covered by this chapter and Chapter 6 are:

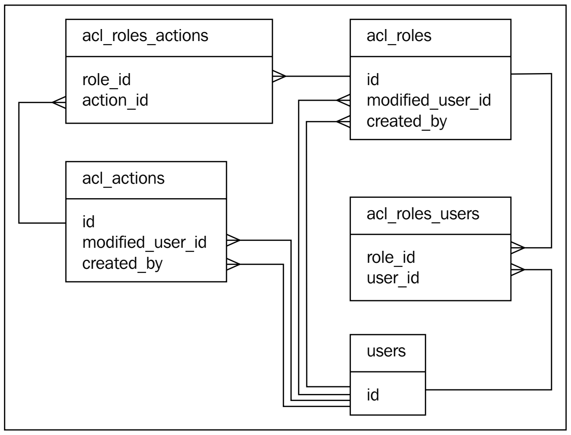
- Database schematic diagrams—We will show you how the tables are related to each other in the database in this chapter.
- SugarCRM table definitions—We will show you the actual structure of each table in the database in the next chapter.

You can then use the contents of the chapters to build your own SQL statements. These are useful if you want to add SQL statements into your new modules and create your own custom reports or run batch programs external to SugarCRM—for example using cron to run regular shell scripts.

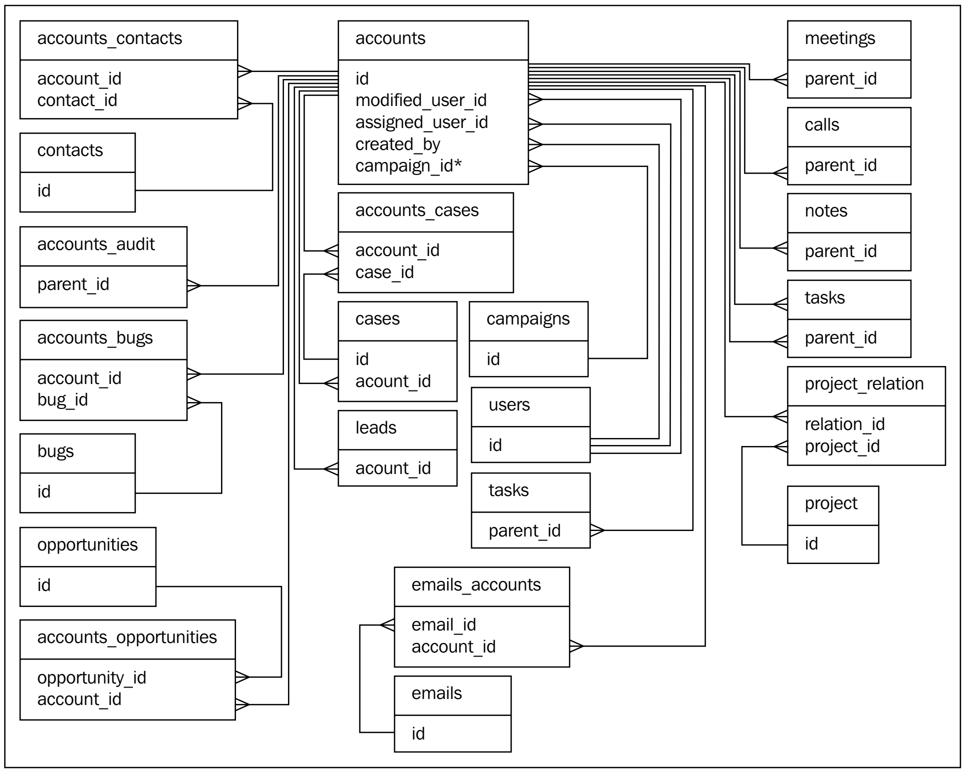
Database Schematic Diagrams

The database schematic diagrams show the relationships between tables in the database. They don't show all the fields in each table—just the fields that can be used in join statements. Refer to the table definitions in Chapter 6 for details of all of the fields in any particular table.

Access Control List

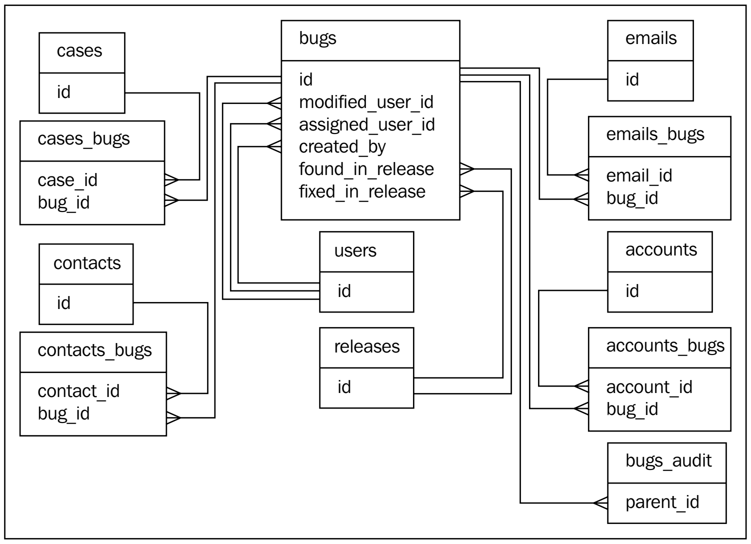


Accounts

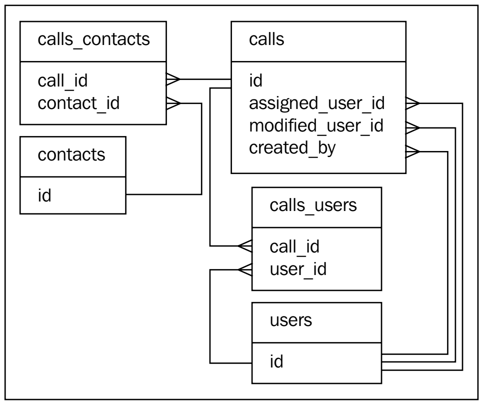


*Added in version 4.5.1

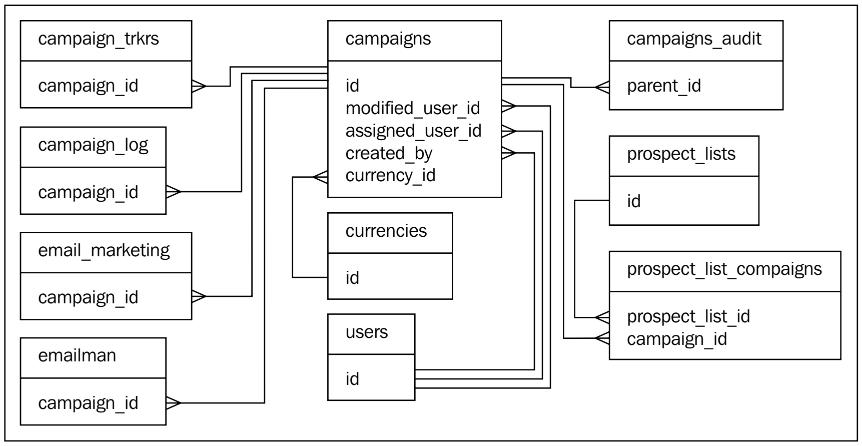
Bugs


cases
id
cases_bugs
case_id
bug_id
contacts
id
contacts_bugs
contact_id
bug_id
bugs
id
modified_user_id
assigned_user_id
created_by
found_in_release
fixed_in_release
users
id
releases
id
emails
id
emails_bugs
email_id
bug_id
accounts
id
accounts_bugs
account_id
bug_id
bugs_audit
parent_id

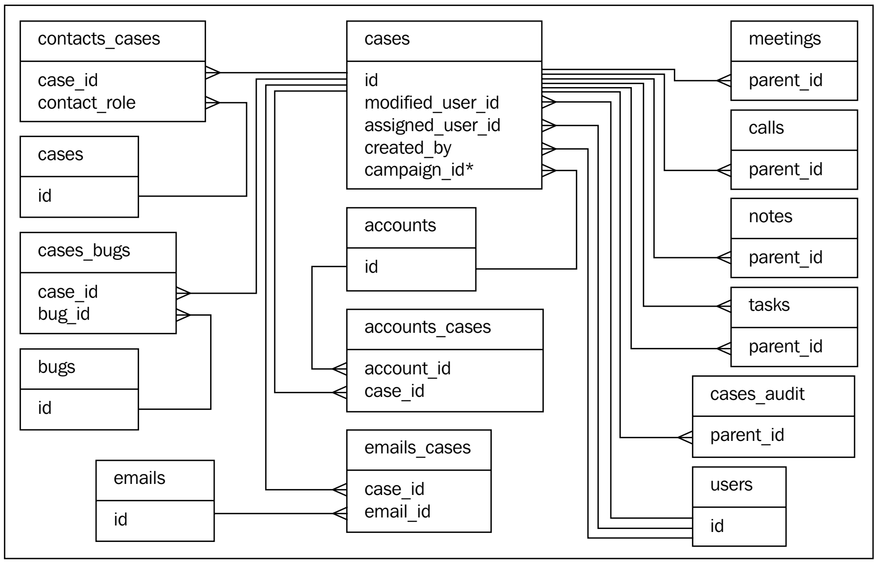
Calls


calls_contacts
call_id
contact_id
contacts
id
calls
id
assigned_user_id
modified_user_id
created_by
calls_users
call_id
user_id
users
id

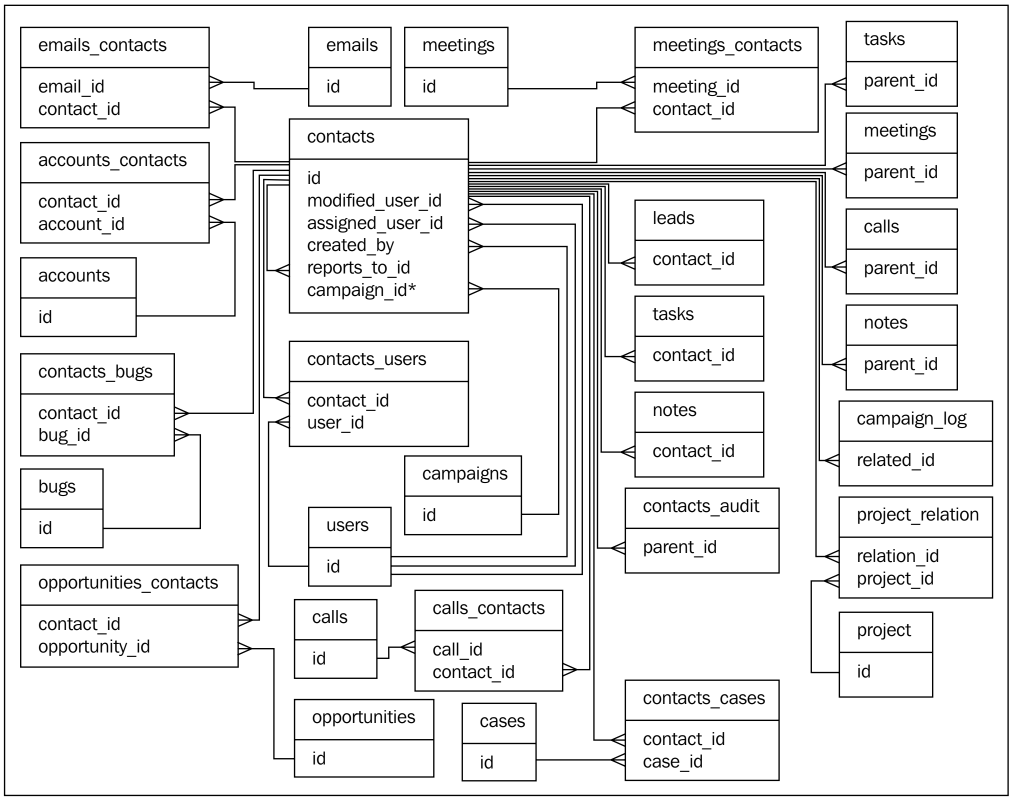
Campaigns



Cases

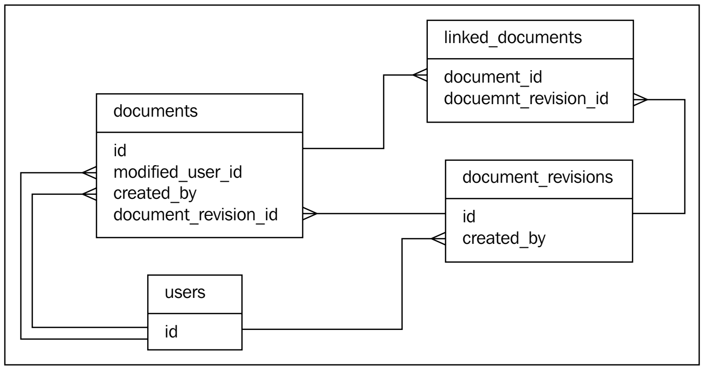


Contacts

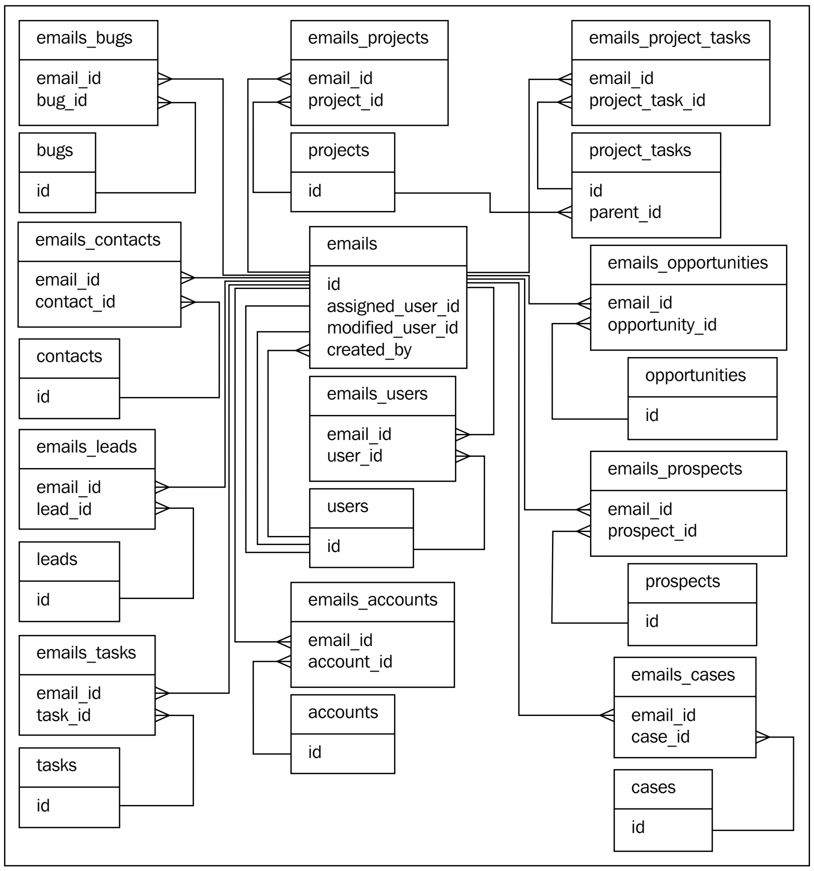


*Added in version 4.5.1

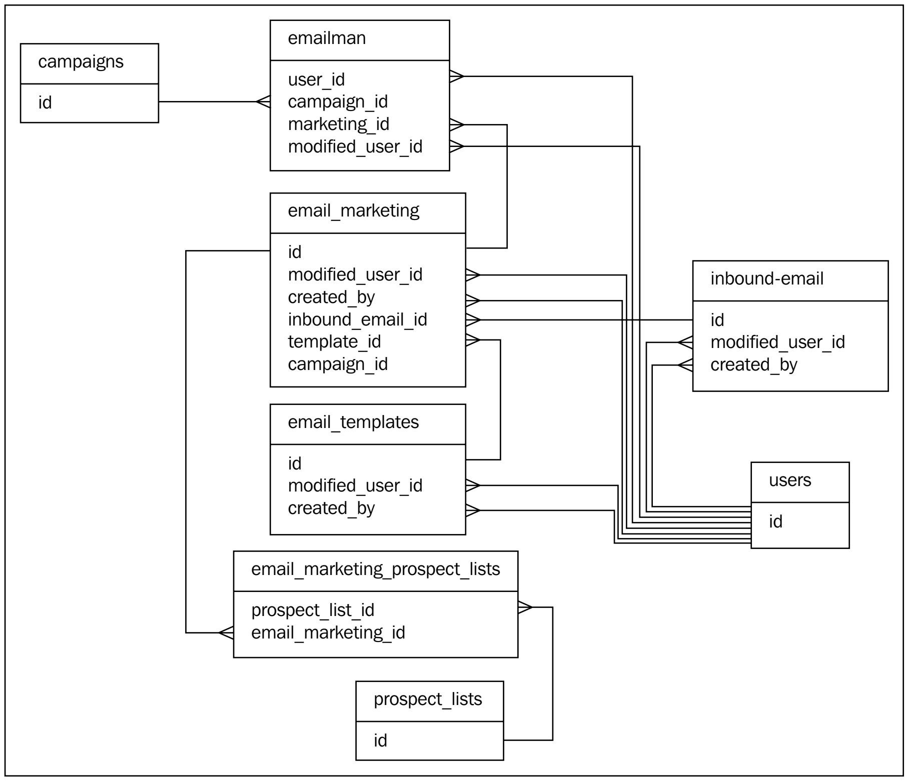
Documents



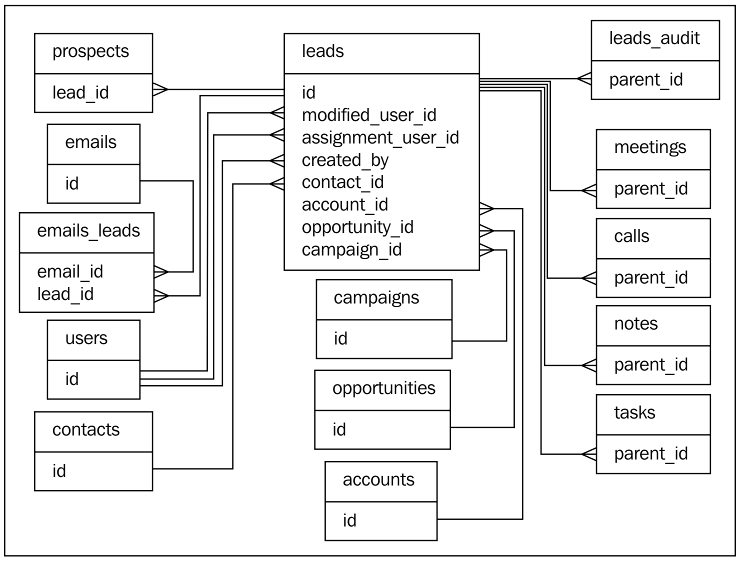
Emails



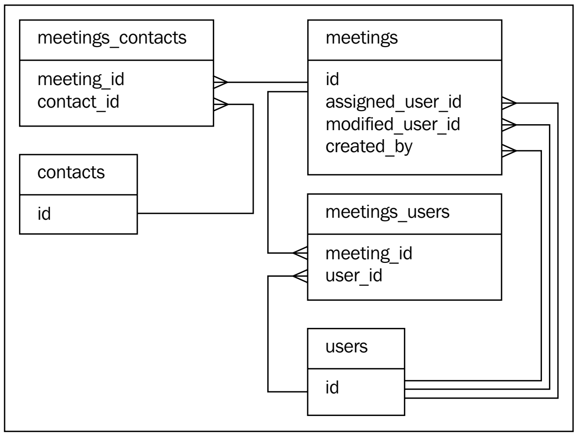
Email Management



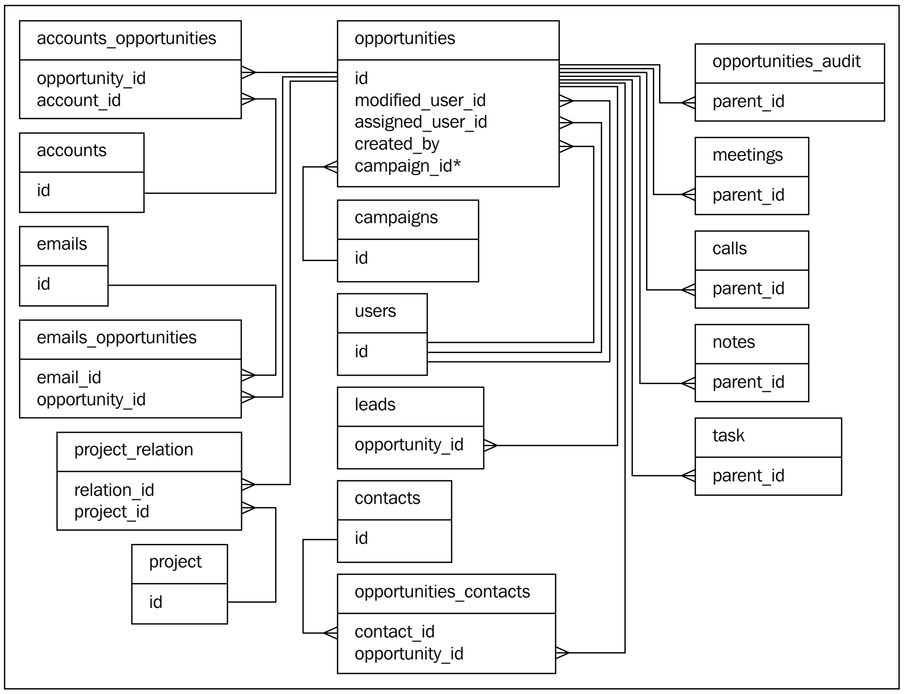
Leads



Meetings

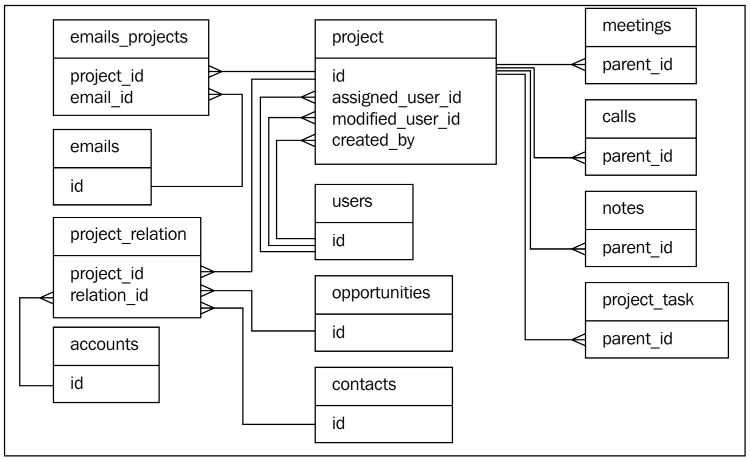


Opportunities

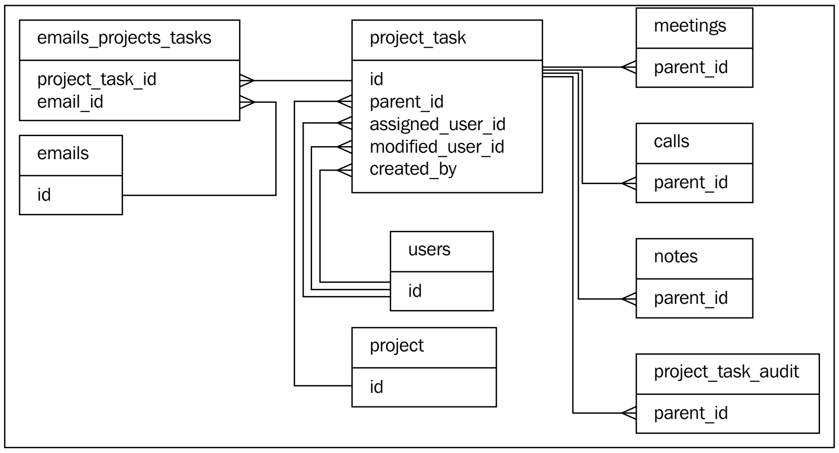


*Added in version 4.5.1

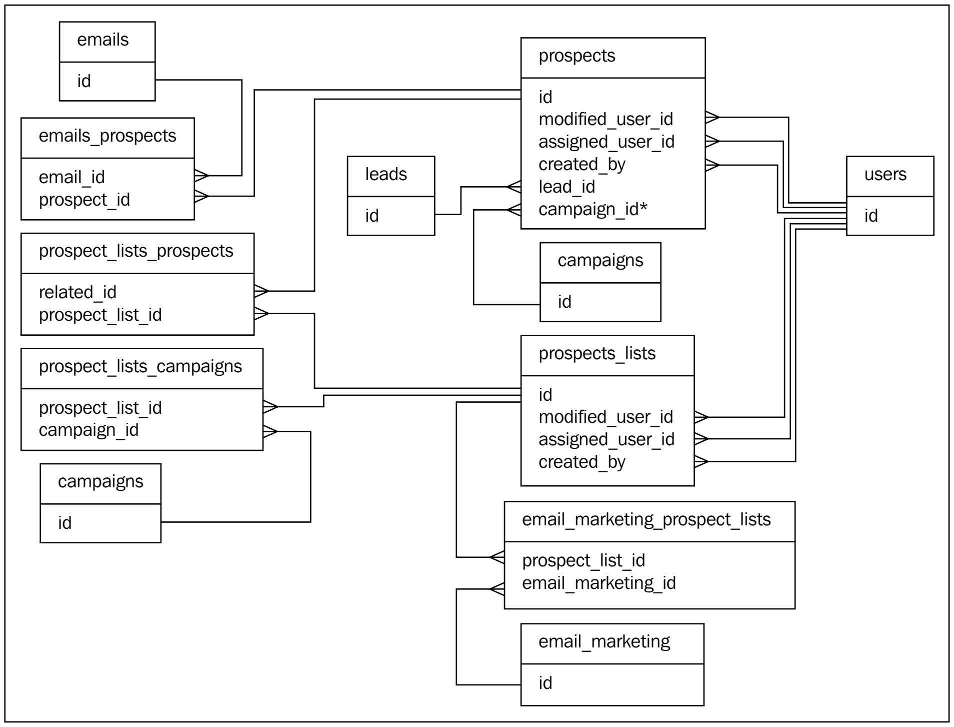
Projects



Project Tasks

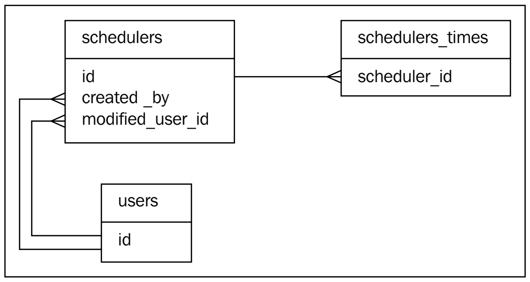


Prospects

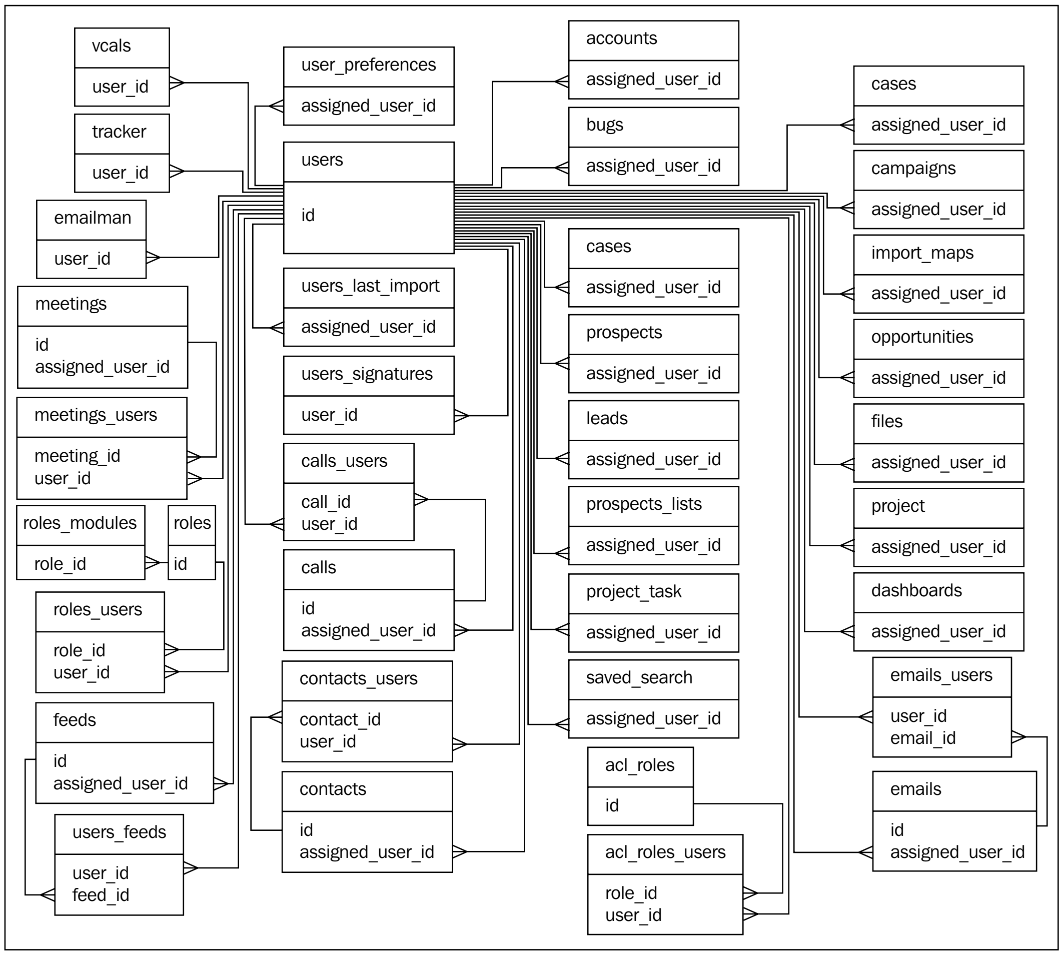


*Added in version 4.5.1

Schedulers



Users



Differences Between Versions 4.5.0 and 4.5.1

There are a few additional relationships that have been added into SugarCRM version 4.5.1—these are all marked in the diagrams.

Summary

It's unlikely that you will have read this chapter from end to end. However, you will (hopefully) find it invaluable when you start to build SQL statements. Remember to use the schematic diagram to understand how the tables fit together. So having seen the database schematics you want to see what fields are available in each table; we will see this in the next chapter.

6
SugarCRM Data Dictionary

In Chapter 5 we looked at the SugarCRM database schematics—this showed us all of the relationships between the tables in the database. We'll now turn our attention to the individual tables themselves.

Differences between Versions 4.5.0 and 4.5.1

You'll find that some minor changes in the table structures between SugarCRM version 4.5.0 and 4.5.1. The changes are:

- One or two additional fields in 4.5.1—these are all marked in the table definitions.
- The data type for each *id* field has been changed from varchar to char. This has no effect on the operation of the database, but can potentially save some of the space required to store the data.

SugarCRM Table Definitions

Having seen how the tables are related together, and the fields that are used to do that, we can now look at the other fields that make up each table. Each table definition contains:

- Field name
- Field type, width and any additional information (such as auto_increment)
- If a null value is allowed
- Index details—this will either be PRI—primary index, or MUL—duplicate entries allowed
- The default value

Accounts

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)			
assigned_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
name	varchar(150)	YES		NULL
parent_id	char(36)	YES	MUL	NULL
account_type	varchar(25)	YES		NULL
industry	varchar(25)	YES		NULL
annual_revenue	varchar(25)	YES		NULL
phone_fax	varchar(25)	YES		NULL
billing_address_street	varchar(150)	YES		NULL
billing_address_city	varchar(100)	YES		NULL
billing_address_state	varchar(100)	YES		NULL
billing_address_ postalcode	varchar(20)	YES		NULL
billing_address_country	varchar(100)	YES		NULL
description	text	YES		NULL
rating	varchar(25)	YES		NULL
phone_office	varchar(25)	YES		NULL
phone_alternate	varchar(25)	YES		NULL
email1	varchar(100)	YES		NULL
email2	varchar(100)	YES		NULL
website	varchar(255)	YES		NULL
ownership	varchar(100)	YES		NULL
employees	varchar(10)	YES		NULL
sic_code	varchar(10)	YES		NULL

Field Name	Field Type	Null Allowed	Index	Default Value
ticker_symbol	varchar(10)	YES		NULL
shipping_address_street	varchar(150)	YES		NULL
shipping_address_city	varchar(100)	YES		NULL
shipping_address_state	varchar(100)	YES		NULL
shipping_address_postalcode	varchar(20)	YES		NULL
shipping_address_country	varchar(100)	YES		NULL
deleted	tinyint(1)		MUL	0
campaign_id	char(36)	YES		NULL

This table is used in the database schematics:

- Accounts
- Bugs
- Cases
- Contacts
- Emails
- Leads
- Opportunities
- Projects
- Users

Field campaign_id added in version 4.5.1

accounts_audit

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)			
parent_id	char(36)			
date_created	datetime	YES		NULL
created_by	char(36)	YES		NULL
field_name	varchar(100)	YES		NULL

Field Name	Field Type	Null Allowed	Index	Default Value
data_type	varchar(100)	YES		NULL
before_value_ string	varchar(255)	YES		NULL
after_value_ string	varchar(255)	YES		NULL
before_value_ text	text	YES		NULL
after_value_text	text	YES		NULL

This table is used in the database schematic:

- Accounts

accounts_bugs

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
account_id	char(36)	YES	MUL	NULL
bug_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Accounts
- Bugs

accounts_cases

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
account_id	char(36)	YES	MUL	NULL
case_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Accounts
- Cases

accounts_contacts

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
contact_id	char(36)	YES	MUL	NULL
account_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Accounts
- Contacts

accounts_opportunities

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
opportunity_id	char(36)	YES	MUL	NULL
account_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Accounts
- Opportunities

acl_actions

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)			
created_by	char(36)	YES		NULL
name	varchar(150)	YES		NULL
category	varchar(100)	YES		NULL
acltype	varchar(100)	YES		NULL
aclaccess	int(3)	YES		NULL
deleted	tinyint(1)	YES		0

This table is used in the database schematic:

- Access Control List

acl_roles

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)			
created_by	char(36)	YES		NULL
name	varchar(150)	YES		NULL
description	text	YES		NULL
deleted	tinyint(1)	YES		0

This table is used in the database schematics:

- Access Control List
- Users

acl_roles_actions

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
role_id	char(36)	YES	MUL	NULL
action_id	char(36)	YES	MUL	NULL
access_override	int(3)	YES		NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)	YES		0

This table is used in the database schematic:

- Access Control List

acl_roles_users

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
role_id	char(36)	YES	MUL	NULL
user_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)	YES		0

This table is used in the database schematics:

- Access Control List
- Users

Bugs

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
bug_number	int(11) auto_ increment		MUL	NULL
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)			
assigned_user_id	char(36)	YES		NULL
deleted	tinyint(1)			0
name	varchar(255)	YES	MUL	NULL
status	varchar(25)	YES		NULL
priority	varchar(25)	YES		NULL
description	text	YES		NULL
created_by	char(36)	YES		NULL
resolution	varchar(255)	YES		NULL
found_in_release	varchar(255)	YES		NULL
type	varchar(255)	YES		NULL
fixed_in_release	varchar(255)	YES		NULL
work_log	text	YES		NULL
source	varchar(255)	YES		NULL
product_category	varchar(255)	YES		NULL

This table is used in the database schematics:

- Accounts
- Bugs
- Cases
- Contacts
- Emails
- Users

bugs_audit

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)			
parent_id	char(36)			
date_created	datetime	YES		NULL
created_by	char(36)	YES		NULL
field_name	varchar(100)	YES		NULL
data_type	varchar(100)	YES		NULL
before_value_ string	varchar(255)	YES		NULL
after_value_ string	varchar(255)	YES		NULL
before_value_ text	text	YES		NULL
after_value_text	text	YES		NULL

This table is used in the database schematic:

- Bugs

Calls

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
assigned_user_id	char(36)	YES		NULL
modified_user_ id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
name	varchar(50)	YES	MUL	NULL
duration_hours	int(2)	YES		NULL
duration_ minutes	int(2)	YES		NULL
date_start	date	YES		NULL
time_start	time	YES		NULL

Field Name	Field Type	Null Allowed	Index	Default Value
date_end	date	YES		NULL
parent_type	varchar(25)	YES		NULL
status	varchar(25)	YES		NULL
direction	varchar(25)	YES		NULL
parent_id	char(36)	YES		NULL
description	text	YES		NULL
deleted	tinyint(1)			0
reminder_time	int(4)	YES		-1
outlook_id	varchar(255)	YES		NULL

This table is used in the database schematics:

- Accounts
- Calls
- Cases
- Contacts
- Leads
- Opportunities
- Projects
- Project Tasks
- Users

calls_contacts

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
call_id	char(36)	YES	MUL	NULL
contact_id	char(36)	YES	MUL	NULL
required	char(1)	YES		1
accept_status	varchar(25)	YES		none
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Calls
- Contacts

calls_users

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
call_id	char(36)	YES	MUL	NULL
user_id	char(36)	YES	MUL	NULL
required	char(1)	YES		1
accept_status	varchar(25)	YES		none
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Calls
- Users

campaign_log

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
campaign_id	char(36)	YES	MUL	NULL
target_tracker_key	char(36)	YES	MUL	NULL
target_id	char(36)	YES		NULL
target_type	varchar(25)	YES		NULL
activity_type	varchar(25)	YES		NULL
activity_date	datetime	YES		NULL
related_id	char(36)	YES		NULL
related_type	varchar(25)	YES		NULL
archived	tinyint(1)	YES		0
hits	int(11)	YES		0

Field Name	Field Type	Null Allowed	Index	Default Value
list_id	char(36)	YES		NULL
deleted	tinyint(1)	YES		0
date_modified	datetime	YES		NULL
more_information	varchar(100)	YES	MUL	NULL
marketing_id	char(36)	YES		NULL

This table is used in the database schematics:

- Campaigns
- Contacts

Added in version 4.5.1

campaign_trkrs

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
tracker_name	varchar(30)	YES		NULL
tracker_url	varchar(255)	YES		http://
tracker_key	int(11) auto_ increment		MUL	NULL
campaign_id	char(36)	YES		NULL
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
is_optout	tinyint(1)			0
deleted	tinyint(1)			0

This table is used in the database schematic:

- Campaigns

Campaigns

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
tracker_key	int(11) auto_ increment		MUL	NULL
tracker_count	int(11)	YES		0
name	varchar(50)	YES	MUL	NULL
refer_url	varchar(255)	YES		http://
tracker_text	varchar(255)	YES		NULL
date_entered	datetime	YES		NULL
date_modified	datetime	YES		NULL
modified_user_ id	char(36)	YES		NULL
assigned_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
deleted	tinyint(1)			0
start_date	date	YES		NULL
end_date	date	YES		NULL
status	varchar(25)	YES		NULL
currency_id	char(36)	YES		NULL
budget	double	YES		NULL
expected_cost	double	YES		NULL
actual_cost	double	YES		NULL
expected_ revenue	double	YES		NULL
campaign_type	varchar(25)	YES		NULL
objective	text	YES		NULL
content	text	YES		NULL
impressions	int(11)	YES		0
frequency	varchar(25)	YES		NULL

This table is used in the database schematics:

- Campaigns
- Email Management
- Leads
- Prospects
- Users

Added in version 4.5.1

campaigns_audit

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)			
parent_id	char(36)			
date_created	datetime	YES		NULL
created_by	char(36)	YES		NULL
field_name	varchar(100)	YES		NULL
data_type	varchar(100)	YES		NULL
before_value_string	varchar(255)	YES		NULL
after_value_string	varchar(255)	YES		NULL
before_value_text	text	YES		NULL
after_value_text	text	YES		NULL

This table is used in the database schematic:

- Campaigns

Cases

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
case_number	int(11) auto_ increment		MUL	NULL
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)			
assigned_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
deleted	tinyint(1)			0
name	varchar(255)	YES	MUL	NULL
account_id	char(36)	YES		NULL
status	varchar(25)	YES		NULL
priority	varchar(25)	YES		NULL
description	text	YES		NULL
resolution	text	YES		NULL

This table is used in the database schematics:

- Accounts
- Bugs
- Cases
- Contacts
- Emails
- Users

cases_audit

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)			
parent_id	char(36)			
date_created	datetime	YES		NULL
created_by	char(36)	YES		NULL
field_name	varchar(100)	YES		NULL
data_type	varchar(100)	YES		NULL
before_value_ string	varchar(255)	YES		NULL
after_value_ string	varchar(255)	YES		NULL
before_value_ text	text	YES		NULL
after_value_text	text	YES		NULL

This table is used in the database schematic:

- Cases

cases_bugs

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
case_id	char(36)	YES	MUL	NULL
bug_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Bugs
- Cases

Config

Field Name	Field Type	Null Allowed	Index	Default Value
category	varchar(32)	YES	MUL	NULL
name	varchar(32)	YES		NULL
value	text	YES		NULL

Contacts

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
deleted	tinyint(1)		MUL	0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)	YES		NULL
assigned_user_id	char(36)	YES	MUL	NULL
created_by	char(36)	YES		NULL
salutation	varchar(5)	YES		NULL
first_name	varchar(100)	YES		NULL
last_name	varchar(100)	YES	MUL	NULL
lead_source	varchar(100)	YES		NULL
title	varchar(50)	YES		NULL
department	varchar(100)	YES		NULL
reports_to_id	char(36)	YES		NULL
birthdate	date	YES		NULL
do_not_call	varchar(3)	YES		0
phone_home	varchar(25)	YES		NULL
phone_mobile	varchar(25)	YES		NULL
phone_work	varchar(25)	YES		NULL
phone_other	varchar(25)	YES		NULL
phone_fax	varchar(25)	YES		NULL
email1	varchar(100)	YES	MUL	NULL
email2	varchar(100)	YES	MUL	NULL
assistant	varchar(75)	YES		NULL
assistant_phone	varchar(25)	YES		NULL
email_opt_out	varchar(3)	YES		0
primary_address_street	varchar(150)	YES		NULL
primary_address_city	varchar(100)	YES		NULL
primary_address_state	varchar(100)	YES		NULL

Field Name	Field Type	Null Allowed	Index	Default Value
primary_address_ postalcode	varchar(20)	YES		NULL
primary_address_ country	varchar(100)	YES		NULL
alt_address_street	varchar(150)	YES		NULL
alt_address_city	varchar(100)	YES		NULL
alt_address_state	varchar(100)	YES		NULL
alt_address_postalcode	varchar(20)	YES		NULL
alt_address_country	varchar(100)	YES		NULL
description	text	YES		NULL
portal_name	varchar(255)	YES		NULL
portal_active	tinyint(1)			0
portal_app	varchar(255)	YES		NULL
invalid_email	tinyint(1)	YES		0
campaign_id	char(36)	YES		NULL

This table is used in the database schematics:

- Accounts
- Bugs
- Calls
- Cases
- Contacts
- Emails
- Leads
- Meetings
- Opportunities
- Projects
- Users

Added in version 4.5.1

contacts_audit

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)			
parent_id	char(36)			
date_created	datetime	YES		NULL
created_by	char(36)	YES		NULL
field_name	varchar(100)	YES		NULL
data_type	varchar(100)	YES		NULL
before_value_string	varchar(255)	YES		NULL
after_value_string	varchar(255)	YES		NULL
before_value_text	text	YES		NULL
after_value_text	text	YES		NULL

This table is used in the database schematic:

- Contacts

contacts_bugs

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
contact_id	char(36)	YES	MUL	NULL
bug_id	char(36)	YES	MUL	NULL
contact_role	varchar(50)	YES		NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Bugs
- Contacts

contacts_cases

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
contact_id	char(36)	YES	MUL	NULL
case_id	char(36)	YES	MUL	NULL
contact_role	varchar(50)	YES		NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Cases
- Contacts

contacts_users

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
contact_id	char(36)	YES	MUL	NULL
user_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Contacts
- Users

Currencies

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
name	char(36)		MUL	
symbol	char(36)			
iso4217	varchar(3)			
conversion_rate	double			0
status	varchar(25)	YES		NULL
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
created_by	char(36)			

This table is used in the database schematic:

- Campaigns

custom_fields

Field Name	Field Type	Null Allowed	Index	Default Value
bean_id	char(36)	YES	MUL	NULL
set_num	int(11)	YES		0
field0	varchar(255)	YES		NULL
field1	varchar(255)	YES		NULL
field2	varchar(255)	YES		NULL
field3	varchar(255)	YES		NULL
field4	varchar(255)	YES		NULL
field5	varchar(255)	YES		NULL
field6	varchar(255)	YES		NULL
field7	varchar(255)	YES		NULL
field8	varchar(255)	YES		NULL
field9	varchar(255)	YES		NULL
deleted	tinyint(1)	YES		0

Dashboards

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_ id	char(36)			
assigned_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
name	varchar(100)	YES	MUL	NULL
description	text	YES		NULL
content	text	YES		NULL

This table is used in the database schematic:

- Users

document_revisions

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
change_log	varchar(255)	YES		NULL
document_id	char(36)	YES		NULL
date_entered	datetime	YES		NULL
created_by	char(36)	YES		NULL
filename	varchar(255)			
file_ext	varchar(25)	YES		NULL
file_mime_type	varchar(100)	YES		NULL
revision	varchar(25)	YES		NULL
deleted	tinyint(1)	YES		0
date_modified	datetime	YES		NULL

This table is used in the database schematic:

- Documents

Documents

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
document_name	varchar(255)			
active_date	date	YES		NULL
exp_date	date	YES		NULL
description	text	YES		NULL
category_id	varchar(25)	YES	MUL	NULL
subcategory_id	varchar(25)	YES		NULL
status_id	varchar(25)	YES		NULL
date_entered	datetime	YES		NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)	YES		0
modified_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
document_ revision_id	char(36)	YES		NULL
mail_merge_ document	varchar(3)	YES		off
related_doc_id	char(36)	YES		NULL
related_doc_rev_id	char(36)	YES		NULL
is_template	tinyint(1)	YES		0
template_type	varchar(25)	YES		NULL

This table is used in the database schematic:

- Documents

email_marketing

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
deleted	tinyint(1)		MUL	0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
name	varchar(255)	YES	MUL	NULL
from_addr	varchar(100)	YES		NULL
from_name	varchar(100)	YES		NULL
inbound_email_id	char(36)	YES		NULL
date_start	date	YES		NULL
time_start	time	YES		NULL
template_id	char(36)			
status	varchar(25)			
campaign_id	char(36)	YES		NULL
all_prospect_lists	tinyint(1)	YES		0

This table is used in the database schematics:

- Email Management
- Prospects

email_marketing_prospect_lists

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
prospect_list_id	char(36)	YES		NULL
email_marketing_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)	YES		0

This table is used in the database schematics:

- Email Management
- Prospects

email_templates

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
published	varchar(3)	YES		NULL
name	varchar(255)	YES	MUL	NULL
description	text	YES		NULL
subject	varchar(255)	YES		NULL
body	text	YES		NULL
body_html	text	YES		NULL
deleted	tinyint(1)			0
campaign_id	tinyint(1)	YES		0

This table is used in the database schematic:

- Email Management

Added in version 4.5.1

emailman

Field Name	Field Type	Null Allowed	Index	Default Value
date_entered	datetime	YES		NULL
date_modified	datetime	YES		NULL
user_id	char(36)	YES		NULL
id	int(11) auto_increment		PRI	NULL
campaign_id	char(36)	YES	MUL	NULL
marketing_id	char(36)	YES		NULL
list_id	char(36)	YES	MUL	NULL
send_date_time	datetime	YES		NULL

Field Name	Field Type	Null Allowed	Index	Default Value
modified_user_id	char(36)	YES		NULL
in_queue	tinyint(1)	YES		0
in_queue_date	datetime	YES		NULL
send_attempts	int(11)	YES		0
deleted	tinyint(1)	YES		0
related_id	char(36)	YES		NULL
related_type	varchar(100)	YES		NULL

This table is used in the database schematics:

- Campaigns
- Email Management
- Users

Emails

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
assigned_user_id	char(36)	YES	MUL	NULL
modified_user_ id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
name	varchar(255)	YES	MUL	NULL
date_start	date	YES		NULL
time_start	time	YES		NULL
parent_type	varchar(25)	YES		NULL
parent_id	char(36)	YES	MUL	NULL
description	longtext	YES		NULL
description_html	longtext	YES		NULL
from_addr	varchar(100)	YES		NULL
from_name	varchar(100)	YES		NULL

Field Name	Field Type	Null Allowed	Index	Default Value
to_addrs	text	YES		NULL
cc_addrs	text	YES		NULL
bcc_addrs	text	YES		NULL
to_addrs_ids	text	YES		NULL
to_addrs_names	text	YES		NULL
to_addrs_emails	text	YES		NULL
cc_addrs_ids	text	YES		NULL
cc_addrs_names	text	YES		NULL
cc_addrs_emails	text	YES		NULL
bcc_addrs_ids	text	YES		NULL
bcc_addrs_names	text	YES		NULL
bcc_addrs_emails	text	YES		NULL
type	varchar(25)	YES		NULL
status	varchar(25)	YES		NULL
message_id	varchar(100)	YES	MUL	NULL
reply_to_name	varchar(100)	YES		NULL
reply_to_addr	varchar(100)	YES		NULL
intent	varchar(25)	YES		pick
mailbox_id	char(36)	YES		NULL
raw_source	longtext	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Accounts
- Bugs
- Cases
- Contacts
- Emails
- Leads
- Opportunities

- Projects
- Project Tasks
- Prospects
- Users

emails_accounts

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
email_id	char(36)	YES	MUL	NULL
account_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Accounts
- Emails

emails_bugs

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
email_id	char(36)	YES	MUL	NULL
bug_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Bugs
- Emails

emails_cases

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
email_id	char(36)	YES	MUL	NULL
case_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Cases
- Emails

emails_contacts

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
email_id	char(36)	YES	MUL	NULL
contact_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0
campaign_data	text	YES		NULL

This table is used in the database schematics:

- Contacts
- Emails

Added in version 4.5.1

emails_leads

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
email_id	char(36)	YES	MUL	NULL
lead_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0
campaign_data	text	YES		NULL

Added in version 4.5.1

emails_opportunities

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
email_id	char(36)	YES	MUL	NULL
opportunity_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Emails
- Leads

emails_project_tasks

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
email_id	char(36)	YES	MUL	NULL
project_task_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Emails
- Project Tasks

emails_projects

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
email_id	char(36)	YES	MUL	NULL
project_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Emails
- Projects

emails_prospects

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
email_id	char(36)	YES	MUL	NULL
prospect_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0
campaign_data	text	YES		NULL

This table is used in the database schematics:

- Emails
- Prospects

Added in version 4.5.1

emails_tasks

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
email_id	char(36)	YES	MUL	NULL
task_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematic:

- Emails

emails_users

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
email_id	char(36)	YES	MUL	NULL
user_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0
campaign_data	text	YES		NULL

This table is used in the database schematics:

- Emails
- Users

Added in version 4.5.1

Feeds

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00

Field Name	Field Type	Null Allowed	Index	Default Value
modified_user_id	char(36)			
assigned_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
title	varchar(100)	YES	MUL	NULL
description	text	YES		NULL
url	varchar(255)	YES		NULL

This table is used in the database schematic:

- Users

fields_meta_data

Field Name	Field Type	Null Allowed	Index	Default Value
id	varchar(255)		PRI	
name	varchar(255)	YES		NULL
label	varchar(255)	YES		NULL
help	varchar(255)	YES		NULL
custom_module	varchar(255)	YES		NULL
data_type	varchar(255)	YES		NULL
max_size	int(11)	YES		NULL
required_option	varchar(255)	YES		NULL
default_value	varchar(255)	YES		NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)	YES		0
audited	tinyint(1)	YES		0
mass_update	tinyint(1)	YES		0
duplicate_merge	smallint(6)	YES		0
ext1	varchar(255)	YES		
ext2	varchar(255)	YES		
ext3	varchar(255)	YES		
ext4	text	YES		NULL

Files

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
name	char(36)	YES		NULL
content	blob	YES		NULL
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
assigned_user_id	char(36)	YES		NULL

This table is used in the database schematic:

- Users

iframes

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
name	varchar(255)		MUL	
url	varchar(255)			
type	varchar(255)			
placement	varchar(255)			
status	tinyint(1)			0
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
created_by	char(36)			

import_maps

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
name	char(36)			
source	char(36)			
module	char(36)			
content	blob	YES		NULL
has_header	tinyint(1)			1

Field Name	Field Type	Null Allowed	Index	Default Value
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
assigned_user_id	char(36)	YES	MUL	NULL
is_published	varchar(3)			no

This table is used in the database schematic:

- Users

inbound_email

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
name	varchar(255)	YES		NULL
status	varchar(25)			Active
server_url	varchar(100)			
email_user	varchar(100)			
email_password	varchar(100)			
port	int(5)			0
service	varchar(50)			
mailbox	varchar(50)			
delete_seen	tinyint(1)	YES		0
mailbox_type	varchar(10)	YES		NULL
template_id	char(36)	YES		NULL
stored_options	text	YES		NULL
group_id	char(36)	YES		NULL

This table is used in the database schematic:

- Email Management

inbound_email_autoreply

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
autoreplied_to	varchar(100)		MUL	

Leads

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
deleted	tinyint(1)			0
converted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)	YES		NULL
assigned_user_id	char(36)	YES	MUL	NULL
created_by	char(36)	YES		NULL
salutation	varchar(5)	YES		NULL
first_name	varchar(25)	YES		NULL
last_name	varchar(25)	YES	MUL	NULL
title	varchar(100)	YES		NULL
refered_by	varchar(100)	YES		NULL
lead_source	varchar(100)	YES		NULL
lead_source_description	text	YES		NULL
status	varchar(100)	YES		NULL
status_description	text	YES		NULL
department	varchar(100)	YES		NULL
reports_to_id	char(36)	YES		NULL
do_not_call	varchar(3)	YES		0
phone_home	varchar(25)	YES		NULL
phone_mobile	varchar(25)	YES		NULL
phone_work	varchar(25)	YES		NULL

Field Name	Field Type	Null Allowed	Index	Default Value
phone_other	varchar(25)	YES		NULL
phone_fax	varchar(25)	YES		NULL
email1	varchar(100)	YES	MUL	NULL
email2	varchar(100)	YES	MUL	NULL
email_opt_out	varchar(3)	YES		0
primary_address_street	varchar(150)	YES		NULL
primary_address_city	varchar(100)	YES		NULL
primary_address_state	varchar(100)	YES		NULL
primary_address_ postalcode	varchar(20)	YES		NULL
primary_address_country	varchar(100)	YES		NULL
alt_address_street	varchar(150)	YES		NULL
alt_address_city	varchar(100)	YES		NULL
alt_address_state	varchar(100)	YES		NULL
alt_address_postalcode	varchar(20)	YES		NULL
alt_address_country	varchar(100)	YES		NULL
description	text	YES		NULL
account_name	varchar(150)	YES		NULL
account_description	text	YES		NULL
contact_id	char(36)	YES	MUL	NULL
account_id	char(36)	YES	MUL	NULL
opportunity_id	char(36)	YES	MUL	NULL
opportunity_name	varchar(255)	YES		NULL
opportunity_amount	varchar(50)	YES		NULL
campaign_id	char(36)	YES		NULL
portal_name	varchar(255)	YES		NULL
portal_app	varchar(255)	YES		NULL
invalid_email	tinyint(1)	YES		0

This table is used in the database schematics:

- Accounts
- Contacts
- Emails
- Leads

- Opportunities
- Prospects
- Users

leads_audit

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)			
parent_id	char(36)			
date_created	datetime	YES		NULL
created_by	char(36)	YES		NULL
field_name	varchar(100)	YES		NULL
data_type	varchar(100)	YES		NULL
before_value_string	varchar(255)	YES		NULL
after_value_string	varchar(255)	YES		NULL
before_value_text	text	YES		NULL
after_value_text	text	YES		NULL

This table is used in the database schematic:

- Leads

linked_documents

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
parent_id	char(36)	YES		NULL
parent_type	varchar(25)	YES		NULL
document_id	char(36)	YES		NULL
document_revision_id	char(36)	YES		NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematic:

- Documents

Meetings

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
assigned_user_id	char(36)	YES		NULL
modified_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
name	varchar(50)	YES	MUL	NULL
location	varchar(50)	YES		NULL
duration_hours	int(2)	YES		NULL
duration_minutes	int(2)	YES		NULL
date_start	date	YES		NULL
time_start	time	YES		NULL
date_end	date	YES		NULL
parent_type	varchar(25)	YES		NULL
status	varchar(25)	YES		NULL
parent_id	char(36)	YES	MUL	NULL
description	text	YES		NULL
deleted	tinyint(1)			0
reminder_time	int(11)	YES		-1
outlook_id	varchar(255)	YES		NULL

This table is used in the database schematics:

- Accounts
- Cases
- Contacts
- Leads
- Meetings
- Opportunities
- Projects
- Project Tasks
- Users

meetings_contacts

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
meeting_id	char(36)	YES	MUL	NULL
contact_id	char(36)	YES	MUL	NULL
required	char(1)	YES		1
accept_status	varchar(25)	YES		none
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Contacts
- Meetings

meetings_users

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
meeting_id	char(36)	YES	MUL	NULL
user_id	char(36)	YES	MUL	NULL
required	char(1)	YES		1
accept_status	varchar(25)	YES		none
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Meetings
- Users

Notes

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
name	varchar(255)	YES	MUL	NULL
filename	varchar(255)	YES		NULL
file_mime_type	varchar(100)	YES		NULL
parent_type	varchar(25)	YES		NULL
parent_id	char(36)	YES	MUL	NULL
contact_id	char(36)	YES	MUL	NULL
portal_flag	tinyint(1)			0
description	text	YES		NULL
deleted	tinyint(1)			0
embed_flag	tinyint(1)			0

This table is used in the database schematics:

- Accounts
- Cases
- Contacts
- Leads
- Opportunities
- Projects
- Project Tasks

Added in version 4.5.1

Opportunities

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)	YES		NULL
assigned_user_id	char(36)	YES	MUL	NULL
created_by	char(36)	YES		NULL
deleted	tinyint(1)			0
name	varchar(50)	YES	MUL	NULL
opportunity_type	varchar(255)	YES		NULL
lead_source	varchar(50)	YES		NULL
amount	double	YES		NULL
amount_backup	varchar(25)	YES		NULL
amount_usdollar	double	YES		NULL
currency_id	char(36)	YES		NULL
date_closed	date	YES		NULL
next_step	varchar(100)	YES		NULL
sales_stage	varchar(25)	YES		NULL
probability	double	YES		NULL
description	text	YES		NULL
campaign_id	char(36)	YES		NULL

This table is used in the database schematics:

- Accounts
- Contacts
- Emails
- Leads
- Opportunities
- Projects
- Users

Added in version 4.5.1

opportunities_audit

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)			
parent_id	char(36)			
date_created	datetime	YES		NULL
created_by	char(36)	YES		NULL
field_name	varchar(100)	YES		NULL
data_type	varchar(100)	YES		NULL
before_value_ string	varchar(255)	YES		NULL
after_value_ string	varchar(255)	YES		NULL
before_value_ text	text	YES		NULL
after_value_text	text	YES		NULL

This table is used in the database schematic:

- Opportunities

opportunities_contacts

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
contact_id	char(36)	YES	MUL	NULL
opportunity_id	char(36)	YES	MUL	NULL
contact_role	varchar(50)	YES		NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Contacts
- Opportunities

Project

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
assigned_user_id	char(36)	YES		NULL
modified_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
name	varchar(50)			
description	text	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Accounts
- Contacts
- Emails
- Opportunities
- Projects
- Project Tasks
- Users

project_relation

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
project_id	char(36)			
relation_id	char(36)			
relation_type	varchar(255)			
deleted	tinyint(1)			0
date_modified	datetime			0000-00-00 00:00:00

This table is used in the database schematics:

- Accounts
- Contacts
- Opportunities
- Projects

project_task

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
assigned_user_id	char(36)	YES		NULL
modified_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
name	varchar(50)			
status	varchar(255)	YES		NULL
date_due	date	YES		NULL
time_due	time	YES		NULL
date_start	date	YES		NULL
time_start	time	YES		NULL
parent_id	char(36)			
priority	varchar(255)	YES		NULL
description	text	YES		NULL
order_number	int(11)	YES		1
task_number	int(11)	YES		NULL
depends_on_id	char(36)	YES		NULL
milestone_flag	varchar(255)	YES		NULL
estimated_effort	int(11)	YES		NULL
actual_effort	int(11)	YES		NULL
utilization	int(11)	YES		100
percent_complete	int(11)	YES		0
deleted	tinyint(1)			0

This table is used in the database schematics:

- Emails
- Projects
- Project Tasks
- Users

project_task_audit

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)			
parent_id	char(36)			
date_created	datetime	YES		NULL
created_by	char(36)	YES		NULL
field_name	varchar(100)	YES		NULL
data_type	varchar(100)	YES		NULL
before_value_string	varchar(255)	YES		NULL
after_value_string	varchar(255)	YES		NULL
before_value_text	text	YES		NULL
after_value_text	text	YES		NULL

This table is used in the database schematic:

- Project Tasks

prospect_list_campaigns

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
prospect_list_id	char(36)	YES	MUL	NULL
campaign_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)	YES		0

This table is used in the database schematics:

- Campaigns
- Prospects

prospect_lists

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
name	varchar(50)	YES	MUL	NULL
list_type	varchar(25)	YES		NULL
date_entered	datetime	YES		NULL
date_modified	datetime	YES		NULL
modified_user_id	char(36)	YES		NULL
assigned_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
deleted	tinyint(1)			0
description	text	YES		NULL
domain_name	varchar(255)	YES		NULL

This table is used in the database schematics:

- Campaigns
- Email Management
- Prospects
- Users

prospect_lists_prospects

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
prospect_list_id	char(36)	YES	MUL	NULL
related_id	char(36)	YES	MUL	NULL
related_type	varchar(25)	YES		NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)	YES		0

This table is used in the database schematic:

- Prospects

Prospects

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
tracker_key	int(11) auto_ increment		MUL	NULL
deleted	tinyint(1)	YES		0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)	YES		NULL
assigned_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
salutation	varchar(5)	YES		NULL
first_name	varchar(100)	YES		NULL
last_name	varchar(100)	YES	MUL	NULL
title	varchar(25)	YES		NULL
department	varchar(255)	YES		NULL
birthdate	date	YES		NULL
do_not_call	varchar(3)	YES		0
phone_home	varchar(25)	YES		NULL
phone_mobile	varchar(25)	YES		NULL
phone_work	varchar(25)	YES		NULL
phone_other	varchar(25)	YES		NULL
phone_fax	varchar(25)	YES		NULL
email1	varchar(100)	YES		NULL
email2	varchar(100)	YES		NULL
assistant	varchar(75)	YES		NULL
assistant_phone	varchar(25)	YES		NULL
email_opt_out	varchar(3)	YES		0
primary_address_street	varchar(150)	YES		NULL
primary_address_city	varchar(100)	YES		NULL

Field Name	Field Type	Null Allowed	Index	Default Value
primary_address_state	varchar(100)	YES		NULL
primary_address_ postalcode	varchar(20)	YES		NULL
primary_address_ country	varchar(100)	YES		NULL
alt_address_street	varchar(150)	YES		NULL
alt_address_city	varchar(100)	YES		NULL
alt_address_state	varchar(100)	YES		NULL
alt_address_postalcode	varchar(20)	YES		NULL
alt_address_country	varchar(100)	YES		NULL
description	text	YES		NULL
invalid_email	tinyint(1)	YES		0
lead_id	char(36)	YES		NULL
account_name	varchar(150)	YES		NULL
campaign_id	char(36)	YES		NULL

This table is used in the database schematics:

- Emails
- Leads
- Prospects
- Users

Added in version 4.5.1

Relationships

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
relationship_name	varchar(150)		MUL	
lhs_module	varchar(100)			
lhs_table	varchar(64)			
lhs_key	varchar(64)			

Field Name	Field Type	Null Allowed	Index	Default Value
rhs_module	varchar(100)			
rhs_table	varchar(64)			
rhs_key	varchar(64)			
join_table	varchar(64)	YES		NULL
join_key_lhs	varchar(64)	YES		NULL
join_key_rhs	varchar(64)	YES		NULL
relationship_type	varchar(64)	YES		NULL
relationship_role_column	varchar(64)	YES		NULL
relationship_role_column_value	varchar(50)	YES		NULL
reverse	tinyint(1)	YES		0
deleted	tinyint(1)	YES		0

Releases

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)			
created_by	char(36)	YES		NULL
name	varchar(50)		MUL	
list_order	int(4)	YES		NULL
status	varchar(25)	YES		NULL

This table is used in the database schematic:

- Bugs

Roles

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_ id	char(36)			
created_by	char(36)	YES		NULL
name	varchar(150)	YES		NULL
description	text	YES		NULL
modules	text	YES		NULL
deleted	tinyint(1)	YES		0

This table is used in the database schematics:

- Access Control List
- Users

roles_modules

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
role_id	char(36)	YES	MUL	NULL
module_id	char(36)	YES	MUL	NULL
allow	tinyint(1)	YES		0
date_modified	datetime	YES		NULL
deleted	tinyint(1)	YES		0

This table is used in the database shematic

- Users

roles_users

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
role_id	char(36)	YES	MUL	NULL
user_id	char(36)	YES	MUL	NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)	YES		0

This table is used in the database schematics:

- Access Control List
- Users

saved_search

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
name	varchar(150)	YES	MUL	NULL
search_module	varchar(150)	YES		NULL
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
assigned_user_id	char(36)	YES		NULL
contents	text	YES		NULL
description	text	YES		NULL

This table is used in the database schematic:

- Users

Schedulers

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
created_by	char(36)	YES		NULL
modified_user_id	char(36)	YES		NULL
name	varchar(255)			
job	varchar(255)			
date_time_start	datetime		MUL	0000-00-00 00:00:00
date_time_end	datetime	YES		NULL
job_interval	varchar(100)			
time_from	time	YES		NULL
time_to	time	YES		NULL
last_run	datetime	YES		NULL
status	varchar(25)	YES		NULL
catch_up	tinyint(1)	YES		1

This table is used in the database schematic:

- Schedulers

schedulers_times

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
scheduler_id	char(36)		MUL	
execute_time	datetime			0000-00-00 00:00:00
status	varchar(25)			ready

This table is used in the database schematic:

- Schedulers

Tasks

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
assigned_user_id	char(36)	YES	MUL	NULL
modified_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
name	varchar(50)	YES	MUL	NULL
status	varchar(25)	YES		NULL
date_due_flag	varchar(5)	YES		on
date_due	date	YES		NULL
time_due	time	YES		NULL
date_start_flag	varchar(5)	YES		on
date_start	date	YES		NULL
time_start	time	YES		NULL
parent_type	varchar(25)	YES		NULL
parent_id	char(36)	YES	MUL	NULL
contact_id	char(36)	YES	MUL	NULL
priority	varchar(25)	YES		NULL
description	text	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematics:

- Accounts
- Cases
- Contacts
- Emails
- Leads
- Opportunities
- Project Tasks

Tracker

Field Name	Field Type	Null Allowed	Index	Default Value
id	int(11) auto_ increment		PRI	NULL
user_id	char(36)	YES		NULL
module_name	varchar(25)	YES		NULL
item_id	char(36)	YES		NULL
item_summary	varchar(255)	YES		NULL
date_modified	datetime	YES		NULL

This table is used in the database schematic:

- Users

upgrade_history

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
filename	varchar(255)	YES		NULL
md5sum	varchar(32)	YES	MUL	NULL
type	varchar(30)	YES		NULL
status	varchar(50)	YES		NULL
version	varchar(10)	YES		NULL
name	varchar(255)	YES		NULL
description	text	YES		NULL
id_name	varchar(255)	YES		NULL
manifest	datetime			NULL
date_entered	datetime			0000-00-00 00:00:00

user_preferences

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
category	varchar(50)	YES		NULL
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
assigned_user_id	char(36)	YES	MUL	NULL
contents	text	YES		NULL

This table is used in the database schematic:

- Users

Users

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
user_name	varchar(60)	YES	MUL	NULL
user_hash	varchar(32)	YES		NULL
authenticate_id	varchar(100)	YES		NULL
sugar_login	tinyint(1)	YES		1
first_name	varchar(30)	YES		NULL
last_name	varchar(30)	YES		NULL
reports_to_id	char(36)	YES		NULL
is_admin	tinyint(1)	YES		0
receive_ notifications	tinyint(1)	YES		1
description	text	YES		NULL
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)	YES		NULL
created_by	char(36)	YES		NULL
title	varchar(50)	YES		NULL

Field Name	Field Type	Null Allowed	Index	Default Value
department	varchar(50)	YES		NULL
phone_home	varchar(50)	YES		NULL
phone_mobile	varchar(50)	YES		NULL
phone_work	varchar(50)	YES		NULL
phone_other	varchar(50)	YES		NULL
phone_fax	varchar(50)	YES		NULL
email1	varchar(100)	YES		NULL
email2	varchar(100)	YES		NULL
status	varchar(25)	YES		NULL
address_street	varchar(150)	YES		NULL
address_city	varchar(100)	YES		NULL
address_state	varchar(100)	YES		NULL
address_country	varchar(25)	YES		NULL
address_ postalcode	varchar(9)	YES		NULL
user_preferences	text	YES		NULL
deleted	tinyint(1)			0
portal_only	tinyint(1)	YES		0
employee_status	varchar(25)	YES		NULL
messenger_id	varchar(25)	YES		NULL
messenger_type	varchar(25)	YES		NULL
is_group	tinyint(1)	YES		0

This table is used in the database schematics:

- Access Control List
- Accounts
- Bugs
- Calls
- Campaigns
- Cases
- Contacts

- Documents
- Email Management
- Emails
- Leads
- Meetings
- Opportunities
- Projects
- Project Tasks
- Prospects
- Schedulers
- Users

users_feeds

Field Name	Field Type	Null Allowed	Index	Default Value
user_id	char(36)	YES	MUL	NULL
feed_id	char(36)	YES		NULL
rank	int(11)	YES		NULL
date_modified	datetime	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematic:

- Users

users_last_import

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
assigned_user_id	char(36)	YES	MUL	NULL
bean_type	char(36)	YES		NULL
bean_id	char(36)	YES		NULL
deleted	tinyint(1)			0

This table is used in the database schematic:

- Users

users_signatures

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
deleted	tinyint(1)			0
user_id	char(36)	YES	MUL	NULL
name	varchar(255)	YES		NULL
signature	text	YES		NULL
signature_html	text	YES		NULL

This table is used in the database schematic:

- Users

vcals

Field Name	Field Type	Null Allowed	Index	Default Value
id	char(36)		PRI	
deleted	tinyint(1)			0
date_entered	datetime	YES		NULL
date_modified	datetime	YES		NULL
user_id	char(36)			
type	varchar(25)	YES	MUL	NULL
source	varchar(25)	YES		NULL
content	text	YES		NULL

This table is used in the database schematic:

- Users

Versions

Field Name	**Field Type**	**Null Allowed**	**Index**	**Default Value**
id	char(36)		PRI	
deleted	tinyint(1)			0
date_entered	datetime			0000-00-00 00:00:00
date_modified	datetime			0000-00-00 00:00:00
modified_user_id	char(36)			
created_by	char(36)	YES		NULL
name	varchar(255)		MUL	
file_version	varchar(255)			
db_version	varchar(255)			

Summary

With the information that is contained in Chapters 5 and 6 you have all of the information that you need for your SQL statements for any modules or reports that you build.

We have see in these chapters schematic diagrams to understand how the tables fit together and the table definitions to find out which fields are available in each table.

That's enough reference material for now—in Chapter 7 we'll make sure that we're developing in a sensible, and upgrade-safe way as we a look at development and testing strategies.

7
Development and Testing Strategies for SugarCRM

By now you must be confident about customizing SugarCRM, after all over the last six chapters we've looked at:

- Changing the look and feel of SugarCRM
- Adding simple modules
- Custom fields and logic hooks
- The architecture of the application and its supporting database

So, you'll be champing at the bit to move on and start building your own complex modules and processes, and that's what the remaining chapters of this book will help you do. However, before we jump in with both feet, it will be worth spending some time looking at development and testing strategies. By the end of Chapter 7 you will:

- Understand why development and testing strategies are important
- Understand how to set up development and testing servers
- Understand how to migrate code from development to testing to live

And *then* you'll be ready to start building your own modules.

Why Use Development and Testing Strategies?

Let's imagine a scenario—you've carried out all of your customizations, and everything is working beautifully. Pygoscelis, Korora, and all the other Penguin P.I. staff are really happy with your work, and they're now able to use SugarCRM to carry out their day-to-day tasks. In fact, they've even started to rely on the application completely. So, you decide to upgrade to the newest version of SugarCRM—and overnight all of your modifications disappear. Suddenly you become the most unpopular person in the organization.

Or let us think about a second scenario—you've carried out all your customizations, and everything is working beautifully. Pygoscelis is really happy with your work, and tells Korora, and all of the other Penguin P.I. staff, to start using it immediately. However, they turn round and say 'Pygoscelis really doesn't know how we do our work—this is all useless!'. Again, you become the most unpopular person in the organization.

And one final scenario—after the customizations have been completed, and everyone is using the application Korora comes to you and says that she just needs a minor modification carrying out. So, you carry out the changes, only to find that it's affected some of the other modules. And for the third time you become the most unpopular person in the organization.

You don't really want to be that unpopular, so we'll spend some time looking at how you can carry out SugarCRM customizations in a professional and safe manner—so that you will *always* be popular.

The Unbreakable Rule:Thou Shalt Not Do Any Development on a Live Server

The best way to ensure the failure of your project is to start messing about with the live application that people are using. So, don't do it. This may seem common sense, but there is always a temptation just to do a *quick change*, but as Pippin says in *The Lord of the Rings* 'Short cuts make long delays'.

So, if you're not going to do any development to the live SugarCRM application, then where are you going to do it? The answer is to set up two servers:

- Server 1—The live server that people use in their normal daily activities
- Server 2—The development server

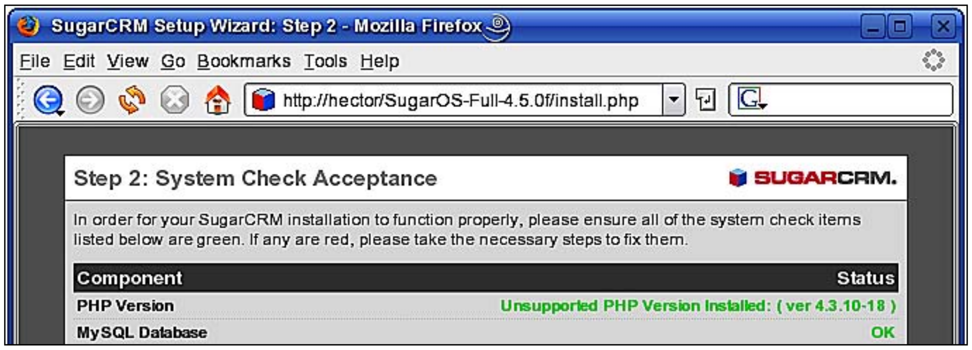
By doing this you can safely develop new modules and new functionality without affecting the users of your application. If something does go wrong then no one will be affected, and, in fact, no one but you need to know about it.

Setting up a Development Server

Having decided that we're *not* going to do any development directly on the live server, then obviously we need to set up a second server on which we can carry out the customizations. Now, at this stage you may be tempted to set up a second version of SugarCRM on your existing web server; after all it is possible to do that. However, there is a very good reason for not doing that.

Let's imagine that you've got an existing SugarCRM implementation based on SugarOS-Full-4.5.0f. Looking on the SugarCRM website you will find that the current version (at the time of writing) is SugarOS-Full-4.5.1, and so, you may decide that an upgrade is needed.

Now, remember from your initial installation that the second stage checks for SugarCRM dependencies. If you've currently got a working instance of SugarCRM then you would have seen something like:

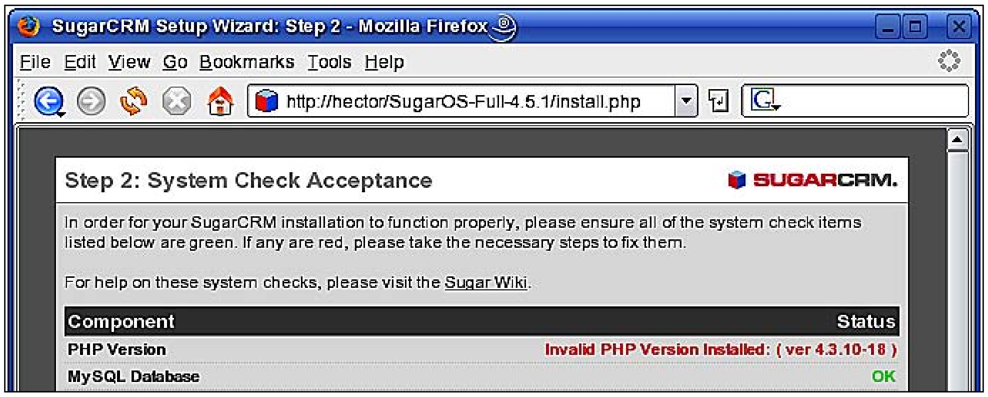

SugarCRM Setup Wizard: Step 2 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://hector/SugarOS-Full-4.5.0f/install.php
SUGARCRM.

Step 2: System Check Acceptance

In order for your SugarCRM installation to function properly, please ensure all of the system check items listed below are green. If any are red, please take the necessary steps to fix them.

Component	Status
PHP Version	Unsupported PHP Version Installed: (ver 4.3.10-18)
MySQL Database	OK

However, if you download the newest version and try to install it on your existing web server then you may come across an immediate show stopper (have a look at the line **PHP version**):


SugarCRM Setup Wizard: Step 2 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://hector/SugarOS-Full-4.5.1/install.php
SUGARCRM.

Step 2: System Check Acceptance

In order for your SugarCRM installation to function properly, please ensure all of the system check items listed below are green. If any are red, please take the necessary steps to fix them.

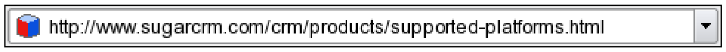
For help on these system checks, please visit the Sugar Wiki.

Component	Status
PHP Version	Invalid PHP Version Installed: (ver 4.3.10-18)
MySQL Database	OK

So, in this case the server has a PHP version suitable for SugarOS-Full-4.5.0f, but not SugarOS-Full-4.5.1. We could install a newer version of PHP, but then that goes against rule 1—thou shalt not do any development on a live server. If you do upgrade PHP, then can you *really* guarantee that the existing (live) implementation of SugarCRM will continue working with the newer version of PHP?

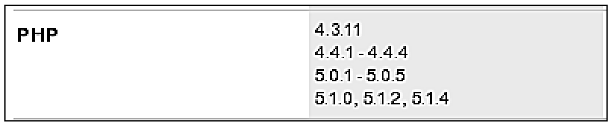
The answer is, therefore, to set up a new, clean server—one that you can play with, and which won't affect your live users.

In the Penguin P.I. example the servers have had Debian Linux as the operating system, and by default it comes with PHP version 4.3.10. Therefore, the question is—which version of PHP *do* you need on the server? Fortunately, you can find this out directly from the SugarCRM website:



http://www.sugarcrm.com/crm/products/supported-platforms.html

And once there we can look at the PHP section:



PHP	4.3.11 4.4.1 - 4.4.4 5.0.1 - 5.0.5 5.1.0, 5.1.2, 5.1.4

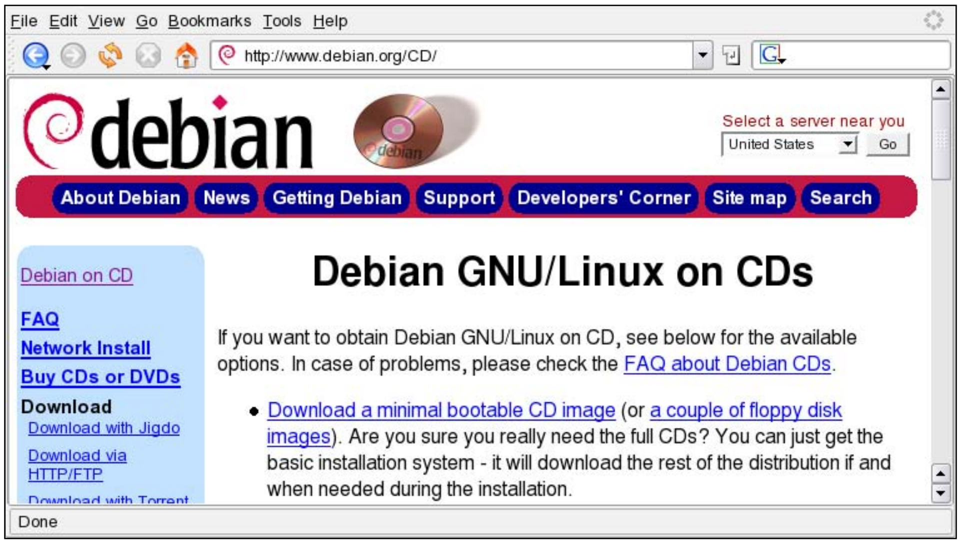
As you can see, the current version of SugarCRM does not support PHP 4.3.10. However, because we're going to be using two separate servers, we can upgrade one of them to another version of PHP and see what the effect is.

Creating a Server

If you're already confident in setting up your own server then just move on to the next section—where we'll look at migrating files from one server to another. If not then let's see just how easy it is to create a server using Debian (Debian GNU/Linux if you want to be exact).

The first thing that you need is a computer (obviously). Fortunately you won't need a new state of the art (and expensive) box—Debian, like many versions of Linux, will work on most machines—regardless of age, although a reasonable disk speed and plenty of memory won't do any harm. However, your computer *will* need a network card.

Next, you will need the installation disk, and this can be downloaded directly from Debian at `http://www.debian.org/CD`:



Debian GNU/Linux on CDs

If you want to obtain Debian GNU/Linux on CD, see below for the available options. In case of problems, please check the FAQ about Debian CDs.

- Download a minimal bootable CD image (or a couple of floppy disk images). Are you sure you really need the full CDs? You can just get the basic installation system - it will download the rest of the distribution if and when needed during the installation.

Once you've created your installation disk then it's just a matter of connecting your machine to the network, inserting your disk and rebooting. After that, just follow the on-screen instructions. The process will:

- Format the machine for you
- Install the core Linux files
- Allow you to select one of the online application sources (choose one of the HTTP sources near to your location)
- Create a root user account (so that you can log on to do further work)

If you're wondering why the computer needs a network connection at the moment— it allows you to download most of the required files directly from the Internet, rather than having to install them from a number of disks.

When the process has finished you'll have a minimal setup—just enough to log on and start turning the base install into a working server. In order to do this you need to log on using your new root account, and then use the `apt-get` command to download all of the applications that you're going to need.

Installing Software

Using `apt-get` is simple—all you need to know is the name of the application that you are going to install and then type:

```
apt-get install <package name>
```

For example:

```
apt-get install apache2
```

For your server you're going to need:

- SSH
- nfs-common
- nfs-kernel-server
- Sudo
- Apache2
- MySQL-Server
- PHP4
- php4-mysql
- libapache2-mod-php4
- Unzip

These are just a tiny proportion of all of the Debian packages that are available, but they're all that you'll need in order to manage your SugarCRM server. You will have to modify Apache's `config` files so that it recognizes PHP, but apart from that all you need to do now is to set up the server's IP address.

Setting the Server's IP Address

By default your new server would have been given a dynamic IP address—meaning that every time that there's a reboot then it will (potentially) be given a different IP address. And I'm sure that you will agree that this is not really of any use to the people trying to access SugarCRM. You'll be pleased to know that there is an easy remedy. All that you have to do is edit a file called `/etc/network/interfaces`. You need to find a line that says:

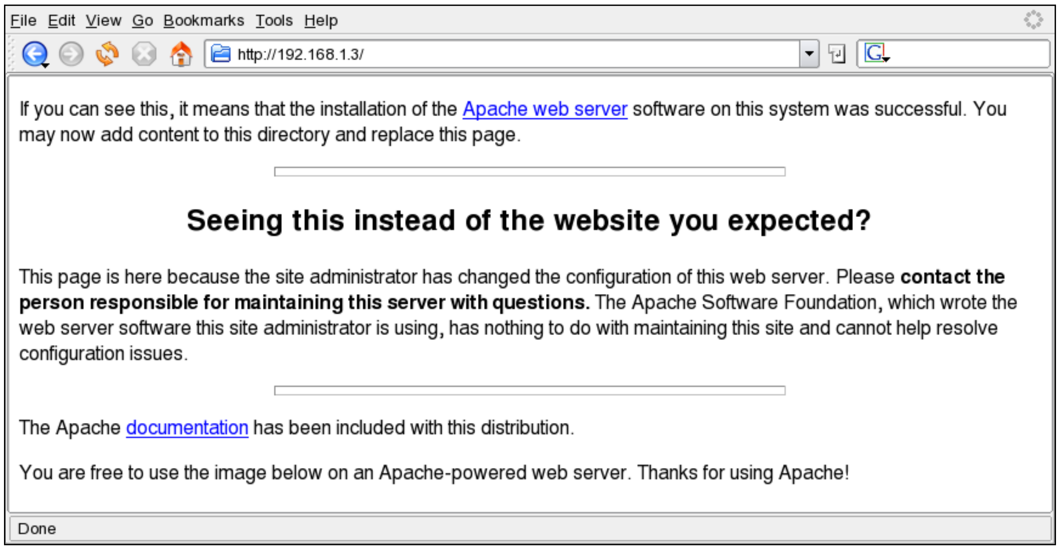
```
iface eth0 inet dhcp
```

Either comment out or delete the line and then add:

```
iface eth0 inet static
        address 192.168.1.3
        netmask 255.255.255.0
        gateway 192.168.1.1
```

The address and gateway will, of course, depend on your network—`address` is the IP address that you want your machine to have, and `gateway` is your network's gateway to the Internet.

Finally, reboot the machine, and you've got a server ready for SugarCRM—in fact you'll already be able to access it from other computers on your network:



If you can see this, it means that the installation of the Apache web server software on this system was successful. You may now add content to this directory and replace this page.

Seeing this instead of the website you expected?

This page is here because the site administrator has changed the configuration of this web server. Please **contact the person responsible for maintaining this server with questions.** The Apache Software Foundation, which wrote the web server software this site administrator is using, has nothing to do with maintaining this site and cannot help resolve configuration issues.

The Apache documentation has been included with this distribution.

You are free to use the image below on an Apache-powered web server. Thanks for using Apache!

With the server in place, and accessible from anywhere on your network, we can now install SugarCRM. However, we're not going to do that from scratch—we're going to use the files from our existing SugarCRM setup.

Migrating SugarCRM Files and Databases Between Servers

As you may well have worked out, we're working with two Debian Linux servers in the Penguin P.I scenario:

- Server 1—hector—which is to be used as the live server, and already has some minor customizations
- Server 2—acamas—the development server

Both servers have Apache, MySQL, and PHP installed—the only difference being that hector has a working version of SugarCRM. Our aim now is to:

- Set the servers up so that server 2 can see all of the files on server 1
- Copy all of the SugarCRM files from server 1 onto server 2
- Ensure that SugarCRM is running correctly on server 2

This can be achieved quite easily in Linux by setting up the appropriate exports and mount points.

Setting Up the Export on Server 1

The plan is to export data from server 1 (hector) to server 2 (acamas), and the first step is add the IP address of acamas into hector's `/etc/hosts` file:

```
192.168.1.3  acamas
```

We do this so that we just have to refer to acamas in other files, rather than having to repeat the IP address all over the place. For example we next have to add an entry to hector's `/etc/exports` file:

```
/       acamas(ro)
```

The entry tell the network that acamas has authorized read access to hector's top directory.

Finally we need to export the information:

```
sudo /usr/sbin/exportfs -a
```

With that done you can access information on hector from acamas—once we've set up acamas, of course.

Setting Up a Mount Point on Server 2

Turning our attention to acamas, we need to update its `/etc/hosts` file:

```
192.168.1.4            hector
```

Next we need to create a directory. This will be the mount point through which we will access all of the files on hector:

```
sudo mkdir /hector
```

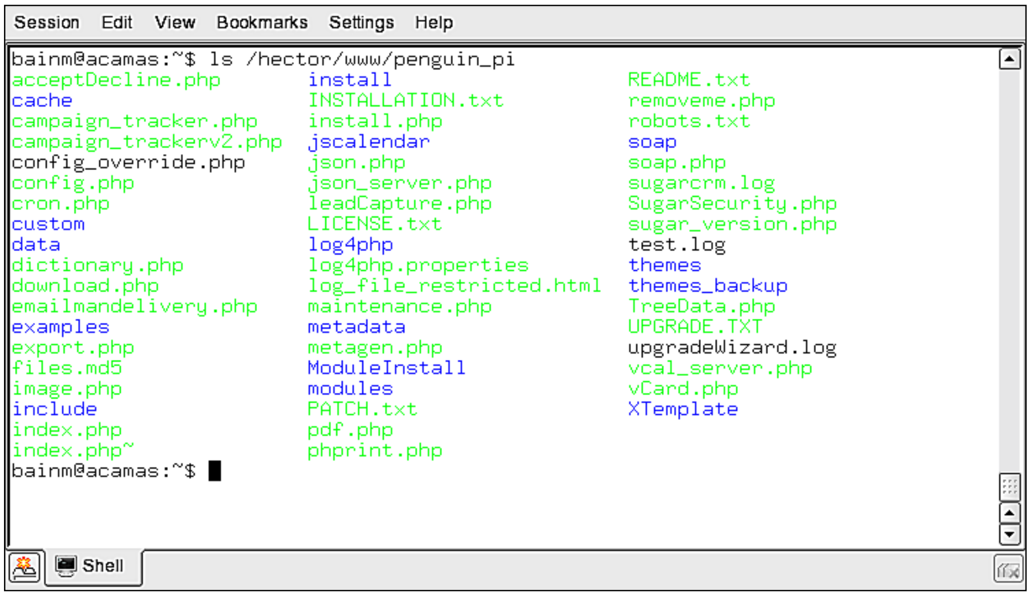
And then we need to tell acamas about this new mount point by updating its `/etc/fstab` file:

```
hector:/        /hector         nfs     ro 0 0
```

Finally we need to mount the mount point:

```
sudo mount /hector
```

Now we can access information on hector from acamas just as if it were in one of acamas' directories:



```
bainm@acamas:~$ ls /hector/www/penguin_pi
acceptDecline.php         install                  README.txt
cache                     INSTALLATION.txt         removeme.php
campaign_tracker.php      install.php              robots.txt
campaign_trackerv2.php    jscalendar               soap
config_override.php       json.php                 soap.php
config.php                json_server.php          sugarcrm.log
cron.php                  leadCapture.php          SugarSecurity.php
custom                    LICENSE.txt              sugar_version.php
data                      log4php                  test.log
dictionary.php            log4php.properties       themes
download.php              log_file_restricted.html themes_backup
emailmandelivery.php      maintenance.php          TreeData.php
examples                  metadata                 UPGRADE.TXT
export.php                metagen.php              upgradeWizard.log
files.md5                 ModuleInstall            vcal_server.php
image.php                 modules                  vCard.php
include                   PATCH.txt                XTemplate
index.php                 pdf.php
index.php~                phprint.php
bainm@acamas:~$
```

Migrating Files from Server 1 to Server 2

We've actually done the hardest part of the migration process, and that wasn't exactly difficult. All that's left to do is to transfer the files that we need from server 1 to server 2. So, to migrate the SugarCRM files we need to log onto server 2 (in this case acamas), and type:

```
sudo cp --preserve -r /hector/www/penguin_pi /www
```

Remembering, of course, to use your own directory location. Then we need to transfer the MySQL files:

```
sudo cp --preserve -r /hector/var/lib/mysql/penguinpi /var/lib/mysql
```

You'll need to restart MySQL:

```
sudo /etc/init.d/mysql restart
```

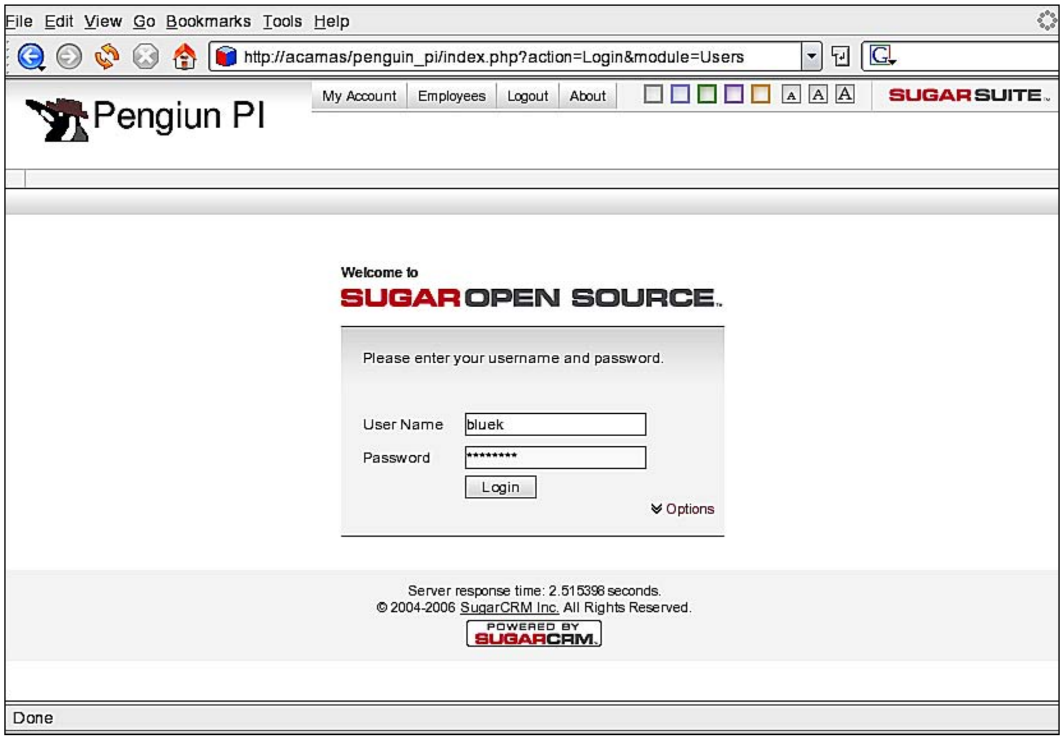
But once you've done that you'll be able to log on to the database and view the SugarCRM tables and their contents. However, you won't be able to access them from your web browser yet. Before you're able to do that you'll need to create a user on the database—the SugarCRM user account that's created during the normal installation process. Chances are you won't be able to remember the details that you originally entered. If you can't then don't worry—they are stored in the SugarCRM `config` file `/var/www/penguin_pi/config.php`. If you look through the file you'll find something like:

```
  array
  (
    'db_host_name' => 'localhost',
    'db_host_instance' => '',
    'db_user_name' => 'penguinpi_user',
    'db_password' => 'penguinpi_go',
    'db_name' => 'penguinpi',
    'db_type' => 'mysql',
  ),
```

You can now log onto the database and create the SugarCRM account from these details:

```
GRANT SELECT,UPDATE,INSERT,DELETE
ON penguinpi.* TO 'penguinpi_user'@'localhost'
IDENTIFIED BY 'penguinpi_go';
GRANT SELECT,UPDATE,INSERT,DELETE
ON penguinpi.* TO 'penguinpi_user'@'acamas'
IDENTIFIED BY 'penguinpi_go';
flush privileges;
```

And so, with all the files in place, and having given SugarCRM access to the database, you can now use a web browser to view your new SugarCRM implementation (which, at the moment, will be identical to your old one):



Please enter your username and password.

User Name bluek

Password ********

Server response time: 2.515398 seconds.

© 2004-2006 SugarCRM Inc. All Rights Reserved.

You can now customize and upgrade to your heart's content, knowing that any changes you make will not affect the live users at all—well, not until you're ready that is.

An Example Upgrade

You'll remember that earlier in the chapter we saw that we couldn't upgrade to SugarOS-Full-4.5.0h because:

- SugarOS-Full-4.5.0h needs a newer version of PHP than Debian supplies.
- We can't be sure of the effects that upgrading PHP will have on SugarOS-Full-4.5.0f.
- We don't want to do anything on the live server that might affect our users. If fact, we don't even want to do anything on the live server that *shouldn't* affect our users.

However, now that we've got a development server we can safely carry out the upgrade and see what effect it does have.

Upgrading PHP

Obviously, you need to check how to carry out the upgrade for your own operating system, but on Debian it's just of matter of carrying out two steps:

- Select an appropriate download source
- Install the software

So, the first step is to find a download source. One such source is `dotdeb.org`—this is a repository for many current applications, and so it's just a matter of updating `/etc/apt/sources.list` with the source details:

```
deb http://packages.dotdeb.org stable all
deb-src http://packages.dotdeb.org stable all
```

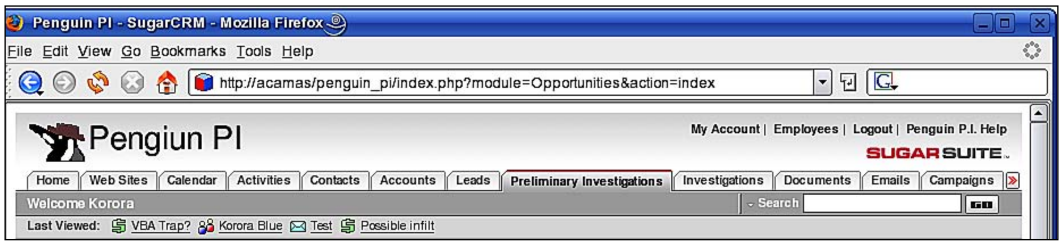
Next install the new version of PHP:

```
sudo apt-get update
sudo apt-get install php4
sudo apt-get install php4-mysql
```

And then you can check what version is now installed:

```
php4 -v
PHP 4.4.4-0.dotdeb.3 with Suhosin-Patch 0.9.6 (cli) (built: Nov 16
2006 11:21:12)
Copyright (c) 1997-2004 The PHP Group
Zend Engine v1.3.0, Copyright (c) 1998-2004 Zend Technologies
```

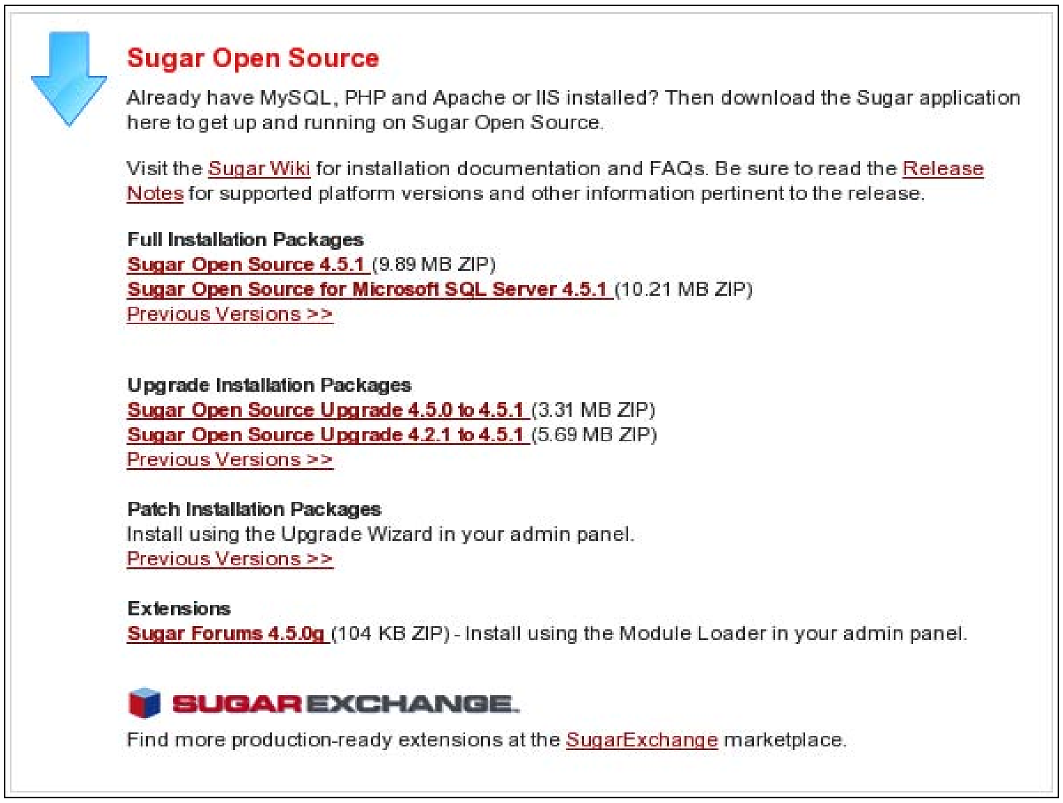
With the correct version of PHP installed, you can login to ensure that everything is working correctly:


Penguin PI - SugarCRM - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://acamas/penguin_pi/index.php?module=Opportunities&action=index
Pengiun PI
My Account | Employees | Logout | Penguin P.I. Help
SUGARSUITE
Home
Web Sites
Calendar
Activities
Contacts
Accounts
Leads
Preliminary Investigations
Investigations
Documents
Emails
Campaigns
Welcome Korora
Search
GO
Last Viewed: VBA Trap? Korora Blue Test Possible infilt

In theory everything should work perfectly, but if it doesn't then at least it won't affect your users—and you'll be able to correct any problems in isolation. And now you can think about upgrading to a newer version of SugarCRM.

Upgrading SugarCRM

As we've now got a suitable version of PHP on our development server we can think about looking at a more current version of SugarCRM. If you take a look at the SugarCRM website then you can see that you have a few options:



Sugar Open Source

Already have MySQL, PHP and Apache or IIS installed? Then download the Sugar application here to get up and running on Sugar Open Source.

Visit the Sugar Wiki for installation documentation and FAQs. Be sure to read the Release Notes for supported platform versions and other information pertinent to the release.

Full Installation Packages
Sugar Open Source 4.5.1 (9.89 MB ZIP)
Sugar Open Source for Microsoft SQL Server 4.5.1 (10.21 MB ZIP)
Previous Versions >>

Upgrade Installation Packages
Sugar Open Source Upgrade 4.5.0 to 4.5.1 (3.31 MB ZIP)
Sugar Open Source Upgrade 4.2.1 to 4.5.1 (5.69 MB ZIP)
Previous Versions >>

Patch Installation Packages
Install using the Upgrade Wizard in your admin panel.
Previous Versions >>

Extensions
Sugar Forums 4.5.0g (104 KB ZIP) - Install using the Module Loader in your admin panel.

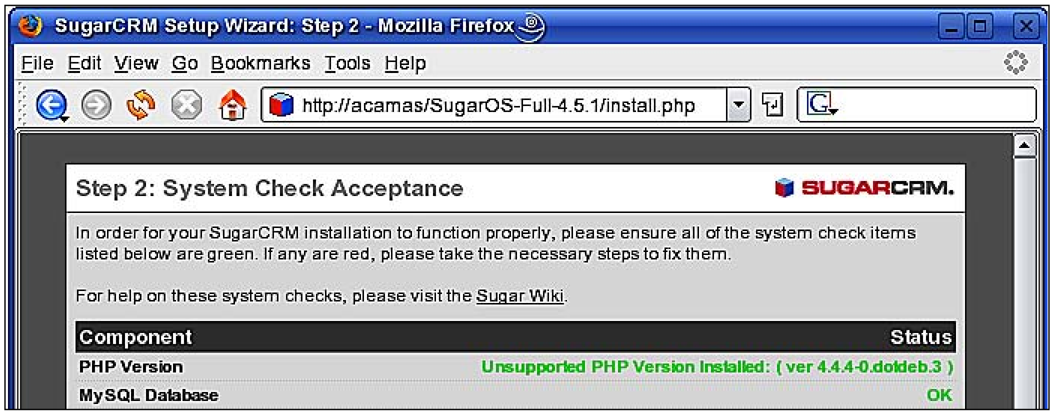
SUGAREXCHANGE.
Find more production-ready extensions at the SugarExchange marketplace.

If you're certain that the upgrade will have no effect on your current implementation then you can download the files that will enable you to upgrade from 4.5.0 to 4.5.1. However, since we have already carried out some customizations then it is much safer to create a new installation, and then carry out a comparison of the two.

So, to get the new version of SugarCRM either download via the browser, or use widget, for example:

```
wget http://www.sugarforge.org/frs/download.php/2535/SugarOS-
4.5.1.zip
```

Once you've unzipped the SugarCRM files then you'll be able to continue the installation process by opening up a web browser starting the installscript:


SugarCRM Setup Wizard: Step 2 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://acamas/SugarOS-Full-4.5.1/install.php
SUGARCRM.

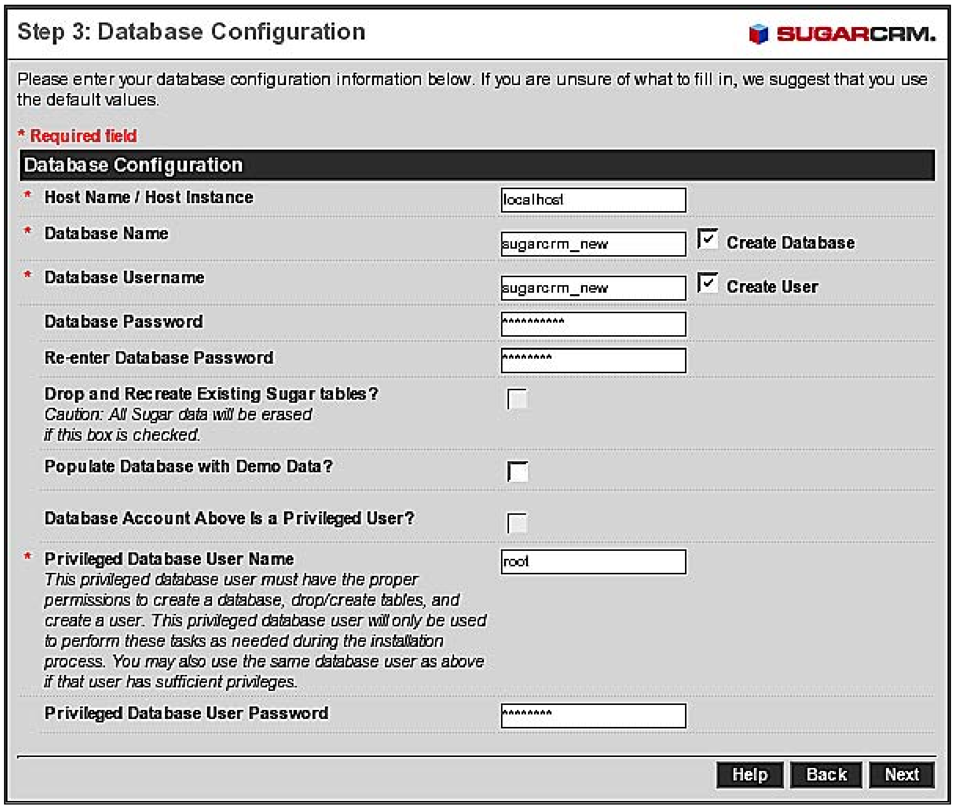
Step 2: System Check Acceptance

In order for your SugarCRM installation to function properly, please ensure all of the system check items listed below are green. If any are red, please take the necessary steps to fix them.

For help on these system checks, please visit the Sugar Wiki.

Component	Status
PHP Version	Unsupported PHP Version Installed: (ver 4.4.4-0.dotdeb.3)
MySQL Database	OK

As you go through the process make sure that you use a new name for your database—we don't want to overwrite the existing one:



Step 3: Database Configuration

Please enter your database configuration information below. If you are unsure of what to fill in, we suggest that you use the default values.

* Required field

Database Configuration

* Host Name / Host Instance	localhost	
* Database Name	sugarcrm_new	✓ Create Database
* Database Username	sugarcrm_new	✓ Create User
Database Password	***********	
Re-enter Database Password	********	
Drop and Recreate Existing Sugar tables? *Caution: All Sugar data will be erased if this box is checked.*	☐	
Populate Database with Demo Data?	☐	
Database Account Above Is a Privileged User?	☐	
* Privileged Database User Name *This privileged database user must have the proper permissions to create a database, drop/create tables, and create a user. This privileged database user will only be used to perform these tasks as needed during the installation process. You may also use the same database user as above if that user has sufficient privileges.*	root	
Privileged Database User Password	********	

Once you've followed all of the instructions, and completed the process then you're ready to start comparing installations.

Comparing Database Files

Before we look at the PHP application side of SugarCRM we'll look at the database. The question that we need to ask first is 'Are the tables in 4.5.0 the same as in 4.5.1?'. We can answer this with a quick bit of Linux scripting:

```
#Define the databases to used
DATABASES[0]="penguinpi"
DATABASES[1]="sugarcrm_new"

#Loop through the databases
for DATABASE in ${DATABASES[*]}
do
  #Count the tables
  TABLES=$(echo "show tables" |
  mysql -s -uroot -ppassword $DATABASE  |
  wc -l )

  #Output the result
  echo $DATABASE $TABLES
done
```

To which you'll get the output:

```
penguinpi 93
sugarcrm_new 92
```

So, at first glance it would appear that there are different tables involved—the new version loses a table. This means that we need to know the differences between the lists of tables. Again, a little Linux scripting will tell us:

```
#Define the databases to used
DATABASES[0]="penguinpi"
DATABASES[1]="sugarcrm_new"

#Loop through the databases
for DATABASE in ${DATABASES[*]}
do
  #output the stucture of the database to files
  echo "show tables" |
  mysql -s -uroot -ppassword $DATABASE > $DATABASE
done

#compare the contents of the files
diff ${DATABASES[*]}
```

This time the output is:

```
< opportunities_cstm
```

If you remember, this table was created automatically by SugarCRM when we introduced our own custom fields—meaning that the default list of tables is the same for both 4.5.0 and 4.5.1. Of course, that doesn't mean that the table structures are the same. So, let's look at that next:

```
#Define the databases
DATABASES[0]="penguinpi"
DATABASES[1]="sugarcrm_new"
#Loop through the databases
for DATABASE in ${DATABASES[*]}
do
  #Obtain the list of fileds in table in each database
  TABLES="$(cat $DATABASE)"
  for TABLE in $TABLES
  do
    echo desc $TABLE |
    mysql -s -uroot -ppassword $DATABASE |
    awk '{print $1}'> $DATABASE.$TABLE
  done
done

Get the list of table files for one database
TABLES="$(cat ${DATABASES[1]})"

#Compare the field list for each table
for TABLE in $TABLES
do
  DIFF="$(diff ${DATABASES[0]}.$TABLE ${DATABASES[1]}.$TABLE | wc -l)"
  if [ $DIFF  -gt 0 ]
  then
    diff ${DATABASES[0]}.$TABLE ${DATABASES[1]}.$TABLE|
    grep ">" | awk '{print $2}' > $TABLE.new
  fi
done

#Output the results
FILES=*.new
for FILE in $FILES
do
   basename $FILE .new
   echo "___________"
   cat $FILE
   echo
done
```

The output of this shows us the additional fields that are required in 4.5.1 (and you may remember these from Chapter 6):

Table	**Additional Fields Required**
accounts	campaign_id
campaign_log	marketing_id
campaigns	impressions frequency
contacts	campaign_id
email_templates	text_only
emails_contacts	campaign_data
emails_leads	campaign_data
emails_prospects	campaign_data
emails_users	campaign_data
notes	embed_flag
opportunities	campaign_id
prospects	campaign_id
upgrade_history	name description id_name manifest

In fact, if you log onto the database you will find that there are also some fundamental changes to the tables themselves:

```
mysql -uroot -ppassword mysql
mysql> desc penguinpi.cases;
+------------------+-------------+------+-----+---------------------+----------------+
| Field            | Type        | Null | Key | Default             | Extra          |
+------------------+-------------+------+-----+---------------------+----------------+
| id               | varchar(36) | NO   | PRI |                     |                |
| case_number      | int(11)     | NO   | MUL | NULL                | auto_increment |
| date_entered     | datetime    | NO   |     | 0000-00-00 00:00:00 |                |
| date_modified    | datetime    | NO   |     | 0000-00-00 00:00:00 |                |
| modified_user_id | varchar(36) | NO   |     |                     |                |
| assigned_user_id | varchar(36) | YES  |     | NULL                |                |
| created_by       | varchar(36) | YES  |     | NULL                |                |
| deleted          | tinyint(1)  | NO   |     | 0                   |                |
| name             | varchar(255)| YES  | MUL | NULL                |                |
| account_id       | varchar(36) | YES  |     | NULL                |                |
| status           | varchar(25) | YES  |     | NULL                |                |
| priority         | varchar(25) | YES  |     | NULL                |                |
| description      | text        | YES  |     | NULL                |                |
| resolution       | text        | YES  |     | NULL                |                |
+------------------+-------------+------+-----+---------------------+----------------+
14 rows in set (0.01 sec)

mysql> desc sugarcrm_new.cases;
+------------------+--------------+------+-----+---------+----------------+
| Field            | Type         | Null | Key | Default | Extra          |
+------------------+--------------+------+-----+---------+----------------+
| id               | char(36)     | NO   | PRI |         |                |
| case_number      | int(11)      | NO   | MUL | NULL    | auto_increment |
| date_entered     | datetime     | NO   |     |         |                |
| date_modified    | datetime     | NO   |     |         |                |
| modified_user_id | char(36)     | NO   |     |         |                |
| assigned_user_id | char(36)     | YES  |     | NULL    |                |
| created_by       | char(36)     | YES  |     | NULL    |                |
| deleted          | tinyint(1)   | NO   |     | 0       |                |
| name             | varchar(255) | YES  | MUL | NULL    |                |
| account_id       | char(36)     | YES  |     | NULL    |                |
| status           | varchar(25)  | YES  |     | NULL    |                |
| priority         | varchar(25)  | YES  |     | NULL    |                |
| description      | text         | YES  |     | NULL    |                |
| resolution       | text         | YES  |     | NULL    |                |
+------------------+--------------+------+-----+---------+----------------+
14 rows in set (0.01 sec)

mysql>
```

You'll notice that the ID, `modified_user_id`, `assigned_user_id`, and `created_by` fields have been changed from varchar to char, and the default values for the `date_entered` and `date_modified` fields have been removed. This means, therefore, that we can't just add the missing fields to our old tables—we must load our existing data into the new tables.

Migrating Database Files

If all of the tables contained the same fields then the migration from one database to another would be very simple, for example for cases the SQL would be:

```
insert into sugarcrm_new.cases select * from penguinpi.cases;
```

However, as we've already seen, some of the new database tables contains additional fields, and you mustn't forget the tables and fields that you may have added to the old database during your customizations. So, let's look at the custom tables first.

Earlier in the chapter we identified the one table that we were using, and that is missing from the new one—that's `opportunities_cstm`. You'll remember that this was created in Chapter 2 when we introduced custom fields into the SugarCRM application. Now, if you manually created the table then you can just apply the SQL to the new database. If not (i.e. SugarCRM created the table for you), then you won't have the SQL. Obviously you could write your own SQL to do the job, but there is an easier way—just copy the MySQL database files from the old database to the new one:

```
sudo cp /var/lib/mysql/penguinpi/opportunities_cstm.* /var/lib/
mysql/sugarcrm_new
```

That's fine for one table, but what if you have introduced a number of tables? If that's the case then the same Linux scripting that told us `opportunities_cstm` was missing can also migrate any other missing tables:

```
#Define the databases
DATABASES[0]="penguinpi"
DATABASES[1]="sugarcrm_new"

#Loop through the databases
for DATABASE in ${DATABASES[*]}
do
  #Write the data structure to a file
  echo "show tables" |
  mysql -s -uroot -ppassword $DATABASE > $DATABASE
done

#Obtain a list of missing tables
diff ${DATABASES[*]} |
awk '{print $2}' |
grep -v ^$ |
while read TABLE
do
 #Copy the missing tables from one database to the other
 sudo cp /var/lib/mysql/penguinpi/${TABLE}.* /var/lib/mysql/sugarcrm_
new/
done
```

I'm sure that you will agree that table migration is very straightforward; however, if you've created custom fields on your tables then things could be a little more involved—and this time you're going to have to do some manual checking, unless, of course you've been careful in the way that you've added the fields.

Let's say, for example, that you added a field called `campaign_id` to the accounts table. If that's the case then you've got a problem, because that's exactly what the new version of SugarCRM has done. However, if you name your fields sensibly then you can avoid this problem—for example if you'd named your field `ppi_campaign_id` (for example) then you can minimize the chances of this type of conflict happening.

Assuming that you've not been tripped up by any field naming problems then the next step is to add your fields to the new database. Now, the process of migrating fields should actually start with the initial creation of the custom field. When you create the custom field don't just log onto the database and type the SQL directly. Instead write your SQL into a file, and then apply the file to the database. Now all you have to do is apply this file to your new database. For example, for our correctly named `ppi_campaign_id` field we would store a file called `ppi_campaign_id.sql`. which would contain:

```
alter table accouts add  ppi_campaign_id char(36);
```

Now we just need to do:

```
mysql -uroot -ppassword sugarcrm_new  <  ppi_campaign_id.sql
```

Or if this is stored in a directory with any other custom field files then we could move to that directory and type:

```
FILES=*.sql
for F in $FILES
do
  mysql -uroot -ppassword sugarcrm_new  <  $F
done
```

Once we've applied the structure to the new database, we need to think about transferring the existing data. However, as we've already identified, some of the tables contain different numbers of fields, and in these cases we have to say exactly which fields are to be used. For example, to transfer the data in accounts the SQL would be:

```
insert into sugarcrm_new.accounts
(id, date_entered, date_modified, modified_user_id, assigned_user_id,
created_by, name, parent_id, account_type, industry, annual_revenue,
phone_fax, billing_address_street, billing_address_city, billing_
address_state, billing_address_postalcode, billing_address_country,
description, rating, phone_office, phone_alternate, email1, email2,
website, ownership, employees, sic_code, ticker_symbol, shipping_
address_street, shipping_address_city, shipping_address_state,
shipping_address_postalcode, shipping_address_country, deleted)
select
id, date_entered, date_modified, modified_user_id, assigned_user_id,
created_by, name, parent_id, account_type, industry, annual_revenue,
phone_fax, billing_address_street, billing_address_city, billing_
address_state, billing_address_postalcode, billing_address_country,
description, rating, phone_office, phone_alternate, email1,
email2, website, ownership, employees, sic_code, ticker_symbol,
shipping_address_street, shipping_address_city, shipping_address_
state, shipping_address_postalcode, shipping_address_country, deleted
from penguinpi.accounts ;
```

To do this by hand would be very time consuming, but again we can turn to a Linux script to do the jobs for us. First we'll create a set of SQL files that will load the data:

```
#Define the databases
DATABASES[0]="penguinpi"
DATABASES[1]="sugarcrm_new"

#Obtain the list of tables
TABLES="$(cat ${DATABASES[1]})"
#Loop through the tables
for TABLE in $TABLES
do
  #Define the SQL to empty the table
  SQL="delete from ${DATABASES[1]}.$TABLE;"

  #Define the SQL to load the new data
  SQL="$SQL insert into ${DATABASES[1]}.$TABLE"
  FIELDS="$(cat ${DATABASES[0]}.$TABLE)"
  FC=$(cat ${DATABASES[0]}.$TABLE|wc -l)
  FIELD_LIST=""
  FN=0
  for FIELD in $FIELDS
  do
    let FN=$FN+1
    if [ $FN -lt $FC ]
    then
       FIELD="$FIELD,"
    fi
    FIELD_LIST="${FIELD_LIST}${FIELD}"
  done
  SQL="$SQL ($FIELD_LIST)"
  SQL="$SQL select"
  SQL="$SQL $FIELD_LIST"
  SQL="$SQL from ${DATABASES[0]}.${TABLE};"

  #Output the complete SQL to a file
  echo $SQL > ${TABLE}.sql
done
```

Next we'll need another piece of script to run each of the SQL files:

```
#Obtain the list of SQL files
SQLS=*.sql

#Loop through the SQL files
```

```
for SQL in $SQLS
do
  #Display the table name
  basename $SQL .sql
  #Run the SQL
  mysql -uroot -ppassword mysql < $SQL
done
```

You may be wondering why we're using individual SQL files. That's just in case there is any problem with an individual table—the remainder of the data load will still continue, leaving you to look at the table's SQL file, and work out what the problem is.

One thing that you will have to do before you continue is to run a little more SQL on the new database:

```
update config set value='4.5.1' where name = 'sugar_version';
```

If you don't do this then SugarCRM will refuse to log on because it will believe that you're trying to use a 4.5.0 database.

At the end of the process you will have all of your data loaded into your brand new SugarCRM database, and it's time to turn your attention to the application files.

Comparing and Migrating the SugarCRM Application Files

Having set up the new SugarCRM database we need to consider the PHP files that make up the application itself. We, of course, can't just copy our modified files over the top of the new ones—we have no idea what kind of affect this will have since we don't know where new functionality has been added to SugarCRM 4.5.1. The first step, therefore, is to find any changes that affect our customizations.

In order to find files that have changed, and what those changes are, we can turn back to a Linux command that we've already been using—`diff`. If you do want to find *every* file that has been changed then go to your web server's document root and type:

```
diff -q -r penguin_pi SugarOS-Full-4.5.1
```

After you've seen data scrolling up the screen for a minute or two then you'll realize that there are an awful lot of files that have been changed. In fact if you type:

```
diff -q -r penguin_pi SugarOS-Full-4.5.1 | wc -l
```

you'll find that there are *1251* differences between the two versions. Obviously we want to narrow it down a little bit. And this is where it becomes essential that you keep track of your customizations as you carry them out—if you do that then this next bit becomes really easy.

In our case, we haven't customized that much yet, so it's not a major problem, the only module that we've worked with is Opportunities—if you remember in Chapter 3 we added some custom fields, and one of the files that we edited was `EditView.html`. So let's compare that to the version in SugarOS-Full-4.5.1 by making use of the Linux `diff` command:

```
diff penguin_pi/modules/Opportunities/EditView.html \
SugarOS-Full-4.5.1/modules/Opportunities/EditView.html
```

The output tells us that the only differences between the files are the ones that we made:

```
87,88c96,97
<       <td  class="dataLabel"><span sugar='slot11'>{MOD.Surveillance_
Required_c_10}</span sugar='slot'></td>
<       <td class="dataField"><span sugar='slot11b'><select
title='{SURVEILLANCE_REQUIRED_C_HELP}' name="surveillance_required_
c">{OPTIONS_SURVEILLANCE_REQUIRED_C}</select></span sugar='slot'></td>
```

You can see that there are differences between line 87 of the first file and 88 of the second, as well as 96 in the first and 97 of the second. So then it's just a matter of migrating our modified file:

```
sudo cp penguin_pi/modules/Opportunities/EditView.html \
SugarOS-Full-4.5.1/modules/Opportunities/
```

Of course we mustn't forget to transfer our custom files (including the logic hooks):

```
sudo cp -r penguin_pi/custom SugarOS-Full-4.5.1
```

And, our nice new theme:

```
sudo cp -r penguin_pi/themes/PenguinPI SugarOS-Full-4.5.1/themes/
```

As well as the module that we created:

```
sudo cp -r penguin_pi/modules/TestApp SugarOS-Full-4.5.1/modules
```

Finally we need to check and then migrate any supporting files that we've had to change:

```
diff penguin_pi/include/modules.php \
SugarOS-Full-4.5.1/include/modules.php
< $moduleList[] = 'TestApp';
< $beanList['NewTab'] = 'TestApp';
< $beanFiles['NewTab'] = 'modules/TestApp/TestApp.php';
```

In this case we can copy our modified `modules.php` over the top of the new one; however, that's not so in the next situation:

```
diff penguin_pi/modules/Emails/vardefs.php \
SugarOS-Full-4.5.1/modules/Emails/vardefs.php
<                        'massupdate'=>true,
>                        'massupdate'=>false,
<                        'type' => 'text',
>                        'type' => 'longtext',
<                        'type' => 'text',
>                        'type' => 'longtext',
```

Here, we can see that there are more changes than the ones that we made, and so you'll need to edit the new file rather than just copying the modified one.

So, as you can see the process consists of:

1. Compare all of the customized files with the new ones.
2. If suitable copy the customized files over the new ones, if not then make the changes directly to the new files.

Once you've done all that then you'll have two identical versions of SugarCRM, except that one is version 4.5.0 and the other is 4.5.1. So, next we need to think about testing our application.

Testing SugarCRM

You're now in a rather nice situation:

- Your users can work in the safe knowledge that their using a stable system that's not subject to random changes.
- You can work on customizations to the system knowing that you won't affect your users.

However, at some point you're going to want to release all of your changes to your users. At this time you'll need to think about testing. By this I don't mean the testing that you should be doing anyway; by testing I mean someone sitting down and replicating the normal day to day task that the users carry out. At this point there are two questions that you need to ask:

- Who is going to do the testing?
- Where is the testing to be carried out?

The answer to the first question should *not* be 'Myself' or 'one of the developers'. Why? Think about emails—when do you notice spelling mistakes? Invariably it's once you've pressed the send button. Bugs are just the same—you'll only find them after the application's been released. Plus, any user will only be confident if the software is tested by someone who understands their job—and that's not you, it's one of them. So, which one of *them* should do the testing? Let's look at the Penguin P.I. organization for some inspiration:

- Sphen—being the managing director he's the last person to ask. Not because he's too important—it's because he only *thinks* he knows how everything works.
- Robby Eudyptes—the newbie—he's not doing anything essential at the moment, as so seems the obvious choice. However, he's also the most inexperienced person in the organization.
- Korora—the most experienced (and busiest) person around—now, that's who you want on board. If you can get Sphen to get Korora to pass some of her work onto Robby, then you've got someone who completely understands the daily workings, and will pick up any problems very quickly.

Next, having a willing volunteer, you're going to have to give them something to test. Obviously you could let them loose on the development server, but by definition that's an unstable environment—plus you may have some elements that you're not ready to release yet. The solution is to set up another server—a test server.

So, it's back to the start of the chapter for you and just follow the instructions for setting up a server. However, this time you won't be migrating from the live environment, you'll be migrating from the development environment. And, of course, in the scenario that we've been looking at then you need two migration and testing periods:

- Migrate the original (4.5.0) set up to the test server with the newer version of PHP. Once that's been tested then you'll be able to upgrade the version of PHP on the live server.
- Migrate your customized version of SugarCRM 4.5.1 to the test server.

Of course, there's one other thing that you may want to consider at this point—documentation. It's all very well Korora telling you that everything is OK, but imagine the embarrassment when you release the software and then a bug is found, and then Korora turns round and says 'Well, I didn't test *that* bit'. So, make sure either:

1. You document the areas to be tested, and Korora signs to say that they've been tested.
2. Korora records her testing.

Obviously, the best solution is a combination of the two.

So, assuming that there are no issues, then you're ready to pass the new version on to all of the other users.

Releasing Your Customizations

You've now done all of the hard work, and you're on the final step of the process—releasing the application. All you have to do now is:

- Carry out any required upgrades to the live sever (such as PHP)
- Transfer the new SugarCRM directory to the live server
- Transfer the new SugarCRM database to the live server
- Migrate the live data into the new database
- Make the new SugarCRM application the default one

We've already covered the first three of these activities in detail, and so we'll just concentrate on the fourth and fifth ones—migrating the live data into the new database, and making the new SugarCRM application the default one.

In fact, we've already seen how to migrate the database data—you'll remember that we created a set of SQL files to copy the data from our live snapshot into the development database—we can use those same SQL files to transfer the data from the live database into the new database. Obviously you'll need to transfer the migration files to the live server, and you'll need to run the files at a time when there aren't any changes being made i.e. there are no users using the application.

And, of course, don't forget to set the value of `sugar_version` to '4.5.1' in the `config` table. The SQL (in case you can't remember) is:

```
update config set value='4.5.1' where name = 'sugar_version';
```

Next you can make the new application the default one—after all the users will want this change over to be as seamless as possible. If you looking in your web server's document root you should see two directories, and in our example these would be:

1. penguin_pi
2. SugarOS-Full-4.5.1

All you have to do is:

```
sudo mv penguin_pi penguin_pi_old
sudo mv SugarOS-Full-4.5.1 penguin_pi
```

Now, when your users next log on they'll be using your new version of SugarCRM. However, there's one little bit of tidying up that you want to do before they do that—the database name.

The database name has no affect on the operation at all, and so the users will be unaware of the fact that their database was named `penguinpi` when they last logged on, but is now named `sugarcrm_new`. However, it is easier for *you* to manage if you maintain some consistency. Therefore the final thing to do is to rename the database, and then tell your new version of SugarCRM about the change.

Renaming the database is easy:

```
cd /var/lib/mysql/
sudo mv penguinpi  penguinpi_old
sudo mv sugarcrm_new  penguinpi
```

Now you'll need to edit the SugarCRM `config.php` file where you'll find something like:

```
    'db_name' => 'sugarcrm_new',
```

Just change it to:

```
   'db_name' => 'penguinpi',
```

Finally you'll need to restart MySQL and Apache:

```
sudo /etc/init.d/mysql restart
sudo /etc/init.d/Apache2 restart
```

And now your users will be free to carry on with their day-to-day tasks, and you can get back to the next round of customizations.

Summary

In this chapter we've looked at how to develop, test, and use SugarCRM in a safe environment. To do this we've seen that we need: a development server, a test server, and a live server.

By setting up a development server we can ensure that SugarCRM customizations can be carried out in isolation—without any danger of affecting users of the live application. The server should have: a copy of the live application, a snapshot of the live data, and all your development work.

You should set up a test server so that you can ensure that: users have a safe environment in which they can test your customizations—they don't have to worry about affecting live data; and testing doesn't have to affect ongoing customizations—you can carry on working towards the next release whilst users test the current one.

Remember the cardinal rule—Thou shalt not do any development on a live server. The only thing that should be placed on the live server is a thoroughly tested release from the test server.

Don't take shortcuts, and always ensure that you follow a well defined process: replicate your live environment on the development server; carry out any customizations on the development server. When you're ready migrate your changes to the test server and get a well respected user to do all of the testing on the test server; if testing is successful then agree a time to migrate your changes to the live server.

If you load your data into a new database ensure that the SugarCRM version is set to the correct value when you finish.

Now that you know that you can develop and test your application safely, it's time to move on to Chapter 8 and look a developing a completely new module.

8
Developing Your Own Modules

Through the course of this book you've learned that the SugarCRM application consists of a number of modules, each of which governs a key element of the sales and service process. You've also learned how to customize those modules so that you can add in your own elements—such as additional drop-down boxes. We've also had a brief look at how to add your own modules. In Chapter 8, we're going to develop these modules further, so that you can introduce all of your required functionality into SugarCRM.

So what sort of functionality do you want to add? Let's imagine two things that Korora at Penguin P.I. might need adding to SugarCRM:

- The ability to create invoices
- A set of reports—again *everyone* needs to produce reports

In Chapter 8 we'll look at some ways in which you could introduce this functionality into your SugarCRM installation (and not necessarily by doing all of the work yourself).

By the end of this chapter you'll be able to:

- Incorporate third-party modules—make the most of work that people have already done
- Build your own fully functional modules

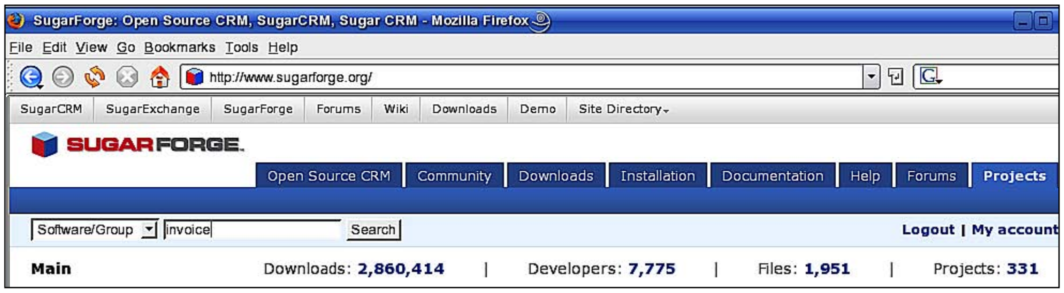
We'll start by looking at third-party modules.

Adding Third-party Modules

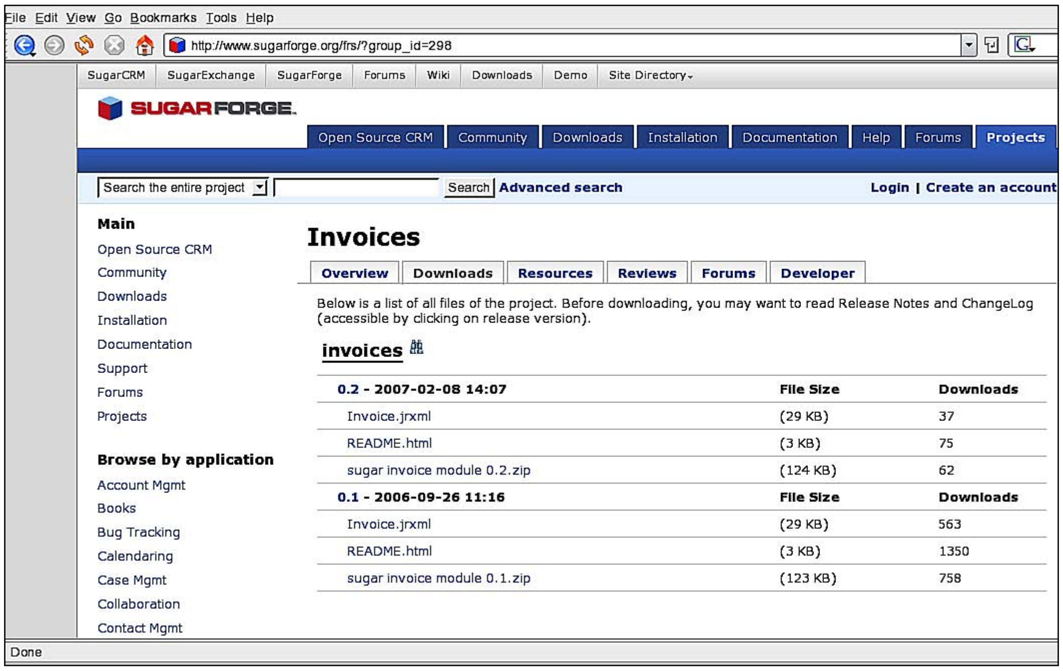
Before you actually move on to developing a new module you really must ask yourself an important question—has it already been done? If someone has already built a module that does the job for you then wouldn't you be better off installing that module, and then spend your time more productively—building modules containing functionality that *doesn't* exist? Therefore, let's start by looking at modules that already exist, and can be used.

You'll be pleased to know that there are already quite a number of modules that are available to you—the number is increasing all the time, and you can download them from the Internet (of course).

The website that you need is `http://www.sugarforge.org` where you'll find all the available modules listed by application type, although if you don't want to hunt through all of the categories then you can make use of the search facility:


SugarForge: Open Source CRM, SugarCRM, Sugar CRM - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.sugarforge.org/
SugarCRM SugarExchange SugarForge Forums Wiki Downloads Demo Site Directory
SUGARFORGE
Open Source CRM Community Downloads Installation Documentation Help Forums Projects
Software/Group invoice Search
Logout | My account
Main Downloads: 2,860,414 | Developers: 7,775 | Files: 1,951 | Projects: 331

You'll find that an invoice module has already been created (by Ray Gauss II), and that you can download it:


File Edit View Go Bookmarks Tools Help
http://www.sugarforge.org/frs/?group_id=298
SugarCRM SugarExchange SugarForge Forums Wiki Downloads Demo Site Directory
SUGARFORGE
Open Source CRM Community Downloads Installation Documentation Help Forums Projects
Search the entire project Search Advanced search
Login | Create an account
Main
Open Source CRM
Community
Downloads
Installation
Documentation
Support
Forums
Projects
Browse by application
Account Mgmt
Books
Bug Tracking
Calendaring
Case Mgmt
Collaboration
Contact Mgmt
Invoices
Overview Downloads Resources Reviews Forums Developer
Done

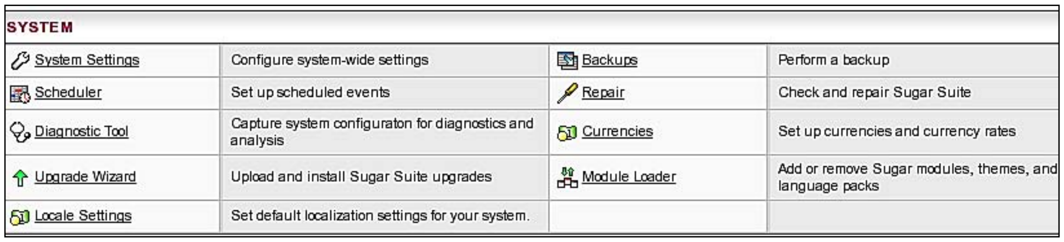
Below is a list of all files of the project. Before downloading, you may want to read Release Notes and ChangeLog (accessible by clicking on release version).

invoices

0.2 - 2007-02-08 14:07	File Size	Downloads
Invoice.jrxml	(29 KB)	37
README.html	(3 KB)	75
sugar invoice module 0.2.zip	(124 KB)	62
0.1 - 2006-09-26 11:16	**File Size**	**Downloads**
Invoice.jrxml	(29 KB)	563
README.html	(3 KB)	1350
sugar invoice module 0.1.zip	(123 KB)	758

Now, you don't have to download the ZIP file to the web server—just your normal desktop will suffice. And, don't unzip it either—SugarCRM will do all of the work for you when you load the module. So, next you'll need to know how to load the module.

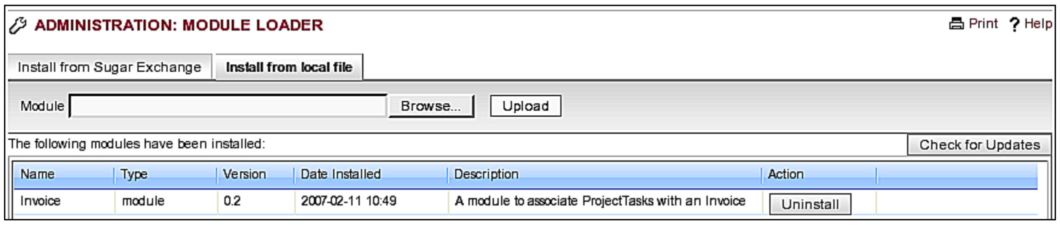
You will need to log on as the SugarCRM administrator, and then go to the **Administration** screen where you'll find the **Module Loader** in the **System** section:



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 configuraton 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.		

With the module loader you can browse for the `invoice.zip` file, upload it onto the server, and install the new module:



The following modules have been installed:

Name	Type	Version	Date Installed	Description	Action
Invoice	module	0.2	2007-02-11 10:49	A module to associate ProjectTasks with an Invoice	Uninstall

Your new invoices module is now up and running and you'll find that:

- There is a new **Invoices** module tab in which you can create your invoices.
- When you edit a project task you'll see that you're able to associate an invoice with it.

If you log onto your database you'll find that you've now got a new table as well:

```
mysql> desc invoice;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| id               | char(36)    | NO   | PRI |         |       |
| date_entered     | datetime    | NO   |     |         |       |
| date_modified    | datetime    | NO   |     |         |       |
| assigned_user_id | char(36)    | YES  |     | NULL    |       |
| contact_id       | char(36)    | NO   |     |         |       |
| modified_user_id | char(36)    | YES  |     | NULL    |       |
| created_by       | char(36)    | YES  |     | NULL    |       |
| name             | varchar(50) | NO   |     |         |       |
| description      | text        | YES  |     | NULL    |       |
| date_sent        | date        | YES  |     | NULL    |       |
| date_paid        | date        | YES  |     | NULL    |       |
| deleted          | tinyint(1)  | NO   |     | 0       |       |
+------------------+-------------+------+-----+---------+-------+
12 rows in set (0.01 sec)
```

In addition to the table some relationships will also have been created:

```
mysql> select relationship_name, lhs_module, lhs_table, lhs_key,
    -> rhs_module, rhs_table, rhs_key
    -> from relationships
    -> where lhs_module = 'Invoice' or rhs_module = 'Invoice';
```

```
+------------------------+------------+-----------+---------+------------+--------------+--------------+
| relationship_name      | lhs_module | lhs_table | lhs_key | rhs_module | rhs_table    | rhs_key      |
+------------------------+------------+-----------+---------+------------+--------------+--------------+
| invoice_notes          | Invoice    | invoice   | id      | Notes      | notes        | parent_id    |
| invoice_project_tasks  | Invoice    | invoice   | id      | ProjectTask | project_task | invoice_id  |
| invoice_assigned_user  | Users      | users     | id      | Invoice    | invoice      | assigned_user_
                                                                                                  id |
| invoice_contact        | Contacts   | contacts  | id      | Invoice    | invoice      | contact_id   |
| invoices_modified_user | Users      | users     | id      | Invoice    | invoice      | modified_user_
                                                                                                  id |
| invoices_created_by    | Users      | users     | id      | Invoice    | invoice      | created_by   |
+------------------------+------------+-----------+---------+------------+--------------+--------------+
6 rows in set (0.01 sec)
```

There is nothing here that you can't do yourself (or, at least, once you've finished this chapter), but using a third-party module will save you a lot of time and effort— provided that the module does the job that you want, of course. However, you may well find that the modules available don't *exactly* do what you want, or maybe there isn't a module that meets your requirements. If that's the case then you're going to have to do everything from scratch.

Creating Custom Modules

This chapter is all about creating custom modules, but you'll remember, no doubt, that we've already learned how to do this in Chapter 2. However, we're going to build a simple module and create something much more complex (and useful). So, to start with let's just recap on the basic requirements for a module.

A (Very) Basic Module

At its simplest level a module is a directory that contains at least three PHP files, and these are:

- `index.php`
- `Forms.php`
- `language/en_us.lang.php`

And that's all there is to a module. So, if we imagine Korora's second requirement (a set of reports), then we might want to do the following (and remember to do it on your development server *not* your live server):

```
mkdir modules/ppi_reports
touch modules/ppi_reports/index.php
touch modules/ppi_reports/Forms.php
mkdir modules/ppi_reports/language
touch modules/ppi_reports/language/en_us.lang.php
```

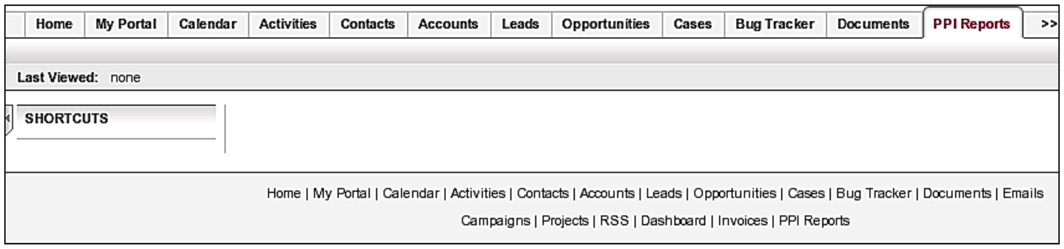
Next, we need to tell SugarCRM about the new module by editing `include/modules.php` and adding:

```
$moduleList[] = 'ppi_reports';
```

Finally we need to edit `custom/include/language/en_us.lang.php` to define the title for the module:

```
$app_list_strings['moduleList']['ppi_reports'] = 'PPI Reports';
```

Then it's just a matter of refreshing your web browser to see the new module:


Home
My Portal
Calendar
Activities
Contacts
Accounts
Leads
Opportunities
Cases
Bug Tracker
Documents
PPI Reports
>>
Last Viewed: none
SHORTCUTS
Home | My Portal | Calendar | Activities | Contacts | Accounts | Leads | Opportunities | Cases | Bug Tracker | Documents | Emails
Campaigns | Projects | RSS | Dashboard | Invoices | PPI Reports

Next we really want to be thinking about the data that we're going to be using.

Data for the New Module

Any report that you make will, naturally, use the tables in the SugarCRM database. This means that you can use the information from Chapters 6 and 7 to create the SQL that's going to extract the correct data for you. So, for example, if Korora wants a report that returns the name of every new Preliminary Investigation created in the current month then you could use the SQL:

```
SELECT name
FROM opportunities
WHERE MONTH(date_entered) = MONTH(NOW());
```

If she wants the assigned user name as well then you could use:

```
SELECT o.name, CONCAT(u.first_name, CONCAT(' ', u.last_name))
FROM opportunities o, users u
WHERE MONTH(o.date_entered) = MONTH(NOW())
AND o.assigned_user_id = u.id;
```

Now that we've got some SQL let's use it in our module.

Processing Data in the Module

We're going to keep things very simple to start with, and so, in this example, we'll:

- Connect to the database
- Run the SQL and obtain a set of records
- Display the contents of our set of records on the screen

You'll remember that we've already created the required files for the module (`index.php`, `Forms.php` and `language/en_us.lang.php`), and in this case we're going to edit `index.php`:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
                                                              Point');
$sql = "SELECT o.name, CONCAT(u.first_name, CONCAT(' ', u.last_name))
       FROM opportunities o, users u
       WHERE MONTH(o.date_entered) = MONTH(NOW())
       AND o.assigned_user_id = u.id";

$report_title = "Monthly New Preliminary Investigations Report";

$result = mysql_query($sql);

echo "<h2>$report_title</h2>";
echo "<table width=100% cellspacing=0 cellpading=0>";
$r=0;
while ($r < mysql_numrows($result))
{
  echo "<tr>";
  $c=0;
  while ($c < mysql_num_fields($result))
  {
    $field = mysql_result($result,$r,$c);
    echo "<td>$field</td>";
    $c++;
  }
  echo "<tr>";
```

```
    $r++;
  }
  echo "</table>";
  ?>
```

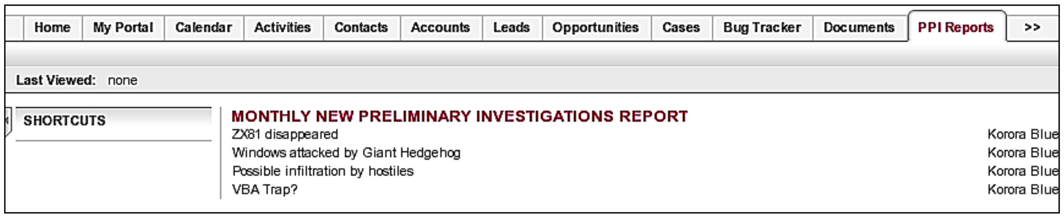
If you look through the code you'll see that we haven't hard coded in any of the database connection details (i.e. the host name, database name, user name, and password), and that's because SugarCRM does that for us. This means, of course, that we can write code without having to worry about such details—for example the password can be changed and it won't affect the operation of our module.

You'll also see that a strange looking line is at the start of the file:

```
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
```

In fact you'll find this line at the start of *every* SugarCRM PHP file. Its purpose? It ensures that any access can only be done through the SugarCRM application, and not by someone randomly accessing one of the files.

The next thing to take note of is the use of two functions—`mysql_numrows` and `mysql_num_fields`. Making use of these means that we don't have to be concerned with the number of rows or fields returned by our SQL—the code will always display them correctly. The end result is something like:



MONTHLY NEW PRELIMINARY INVESTIGATIONS REPORT

ZX81 disappeared	Korora Blue
Windows attacked by Giant Hedgehog	Korora Blue
Possible infiltration by hostiles	Korora Blue
VBA Trap?	Korora Blue

That's fine for a single report, but it's rather unlikely Korora will only need a single report, so the next stage is to add more reports to the module.

Adding More Data

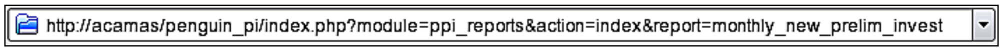
We know that the code will handle any number of fields returned by our SQL, and so we have to do two things:

1. Define the new SQL statement
2. Tell the module which SQL statement to use

We'll decide which SQL to be used by looking at the value of `report`—a variable that we'll pass to the module:

```
$report = $_REQUEST['report'];
if ($report == "monthly_new_prelim_invest")
{
  $sql = "SELECT o.name, CONCAT(u.first_name, CONCAT(' ', u.last_name))
         FROM opportunities o, users u
         WHERE MONTH(o.date_entered) = MONTH(NOW())
         AND o.assigned_user_id = u.id";

  $report_title = "Monthly New Preliminary Investigations Report";

}
  else if ($report == "monthly_open_invest")
 {
  $sql = "SELECT c.name, CONCAT(u.first_name, CONCAT(' ', u.last_name))
         FROM cases c, users u
         WHERE MONTH(c.date_entered) = MONTH(NOW())
         AND c.assigned_user_id = u.id
         AND c.status <> 'Closed'";

  $report_title = "Monthly Open Investigations Report";

 }
  else
{
  $sql = "select 'Choose report'";
}
```

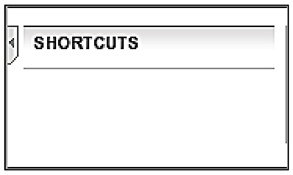
Now you can call the first report by using the URL:


http://acamas/penguin_pi/index.php?module=ppi_reports&action=index&report=monthly_new_prelim_invest

Or to call the second report you can change the URL to end—`monthly_open_invest`. However, at this point, you may be thinking that this is a rather inefficient way of calling the reports, and that we can't expect each user to remember the URLs—and you'd be right. That's why we'll look at shortcuts next.

Adding Shortcuts

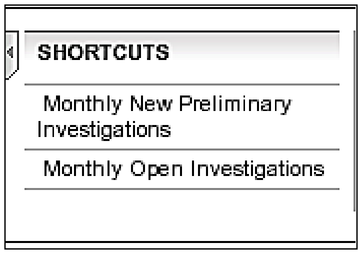
If you'll look at the left of the screen then you'll see the list of shortcuts associated with the module:



Not very impressive at the moment, but we can change this by adding a file to our module's directory. This file is `Menu.php`:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
$url = "index.php?module=ppi_reports&action=index&report=";

$module_menu[] = Array($url . "monthly_new_prelim_invest",
                'Monthly New Preliminary Investigations');
$module_menu[] = Array($url . "monthly_open_invest",
                'Monthly Open Investigations');
?>
```

As you can see the module menu consists of arrays each of which contains the URL for the shortcut and the text to display. Once you've saved the file and refreshed your web browser then you'll see:


SHORTCUTS
Monthly New Preliminary Investigations
Monthly Open Investigations

It may occur to you that we're being a bit inefficient again—we've stored the title for each report in two files: `index.php` and `Menu.php`. It's time to start using a central file for such details, and we've actually already created it—`language/en_us.lang.php`.

Using language/en_us.lang.php

When we created the module we also had to create the file `language/en_us.lang.php`, and this is why. It's used for the central location for text to be used specifically for the module. In this case we can edit it and add:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
                                                              Point');
$mod_strings['lbl_monthly_new_prelim_invest'] =
                            "Monthly New Preliminary Investigations";
$mod_strings['lbl_monthly_open_invest'] = "Monthly Open
Investigations";
?>
```

Then we can change some of the code in `index.php` to use your new `$mod_strings` array:

```
if ($report == "monthly_new_prelim_invest")
{
  $sql = ...

$report_title = $mod_strings['lbl_monthly_new_prelim_invest']."
Report";

}
else if ($report == "monthly_open_invest")
{
  $sql = ...

  $report_title = "$mod_strings['lbl_monthly_open_invest'] . " Report";

}
else
{
  $sql = "select 'Choose report'";
}
```

And then we can do the same in `Menu.php`:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
global $mod_strings;
$url = "index.php?module=ppi_reports&action=index&report=";

$module_menu[] = Array($url . "monthly_new_prelim_invest",
               $mod_strings['lbl_monthly_new_prelim_invest']);
$module_menu[] = Array($url . "monthly_open_invest",
               $mod_strings['lbl_monthly_open_invest']);
?>
```

It's worth noting that `$mod_strings` must be declared as a global in `Menu.php`, but you won't have to do that in `index.php`.

You'll realize the benefits of using a single location immediately—it won't affect your users at all, but it will make life a lot easier for you—you won't have to remember where you've used any particular title or label—you just have one file—`en_us.lang.php`.

Of course, if you want to be *really* efficient then you might want to create a table (or tables) for the module.

Tables for the Module

We've been using `index.php` to create our reports (all two of them), but I'm sure that you can see a major disadvantage here—every time you create a new report then you're going to have to go through the whole testing process before you can allow it to be used on the live server. In fact, we can even start thinking about moving the creation of reports from a developer to a user—after all you don't need a developer to create cases, accounts, or opportunities.

The first thing that we need, therefore, is a table:

```
create table ppi_reports (
   id char(36),
   date_entered datetime,
   assigned_user_id char(36),
   modified_user_id char(36),
   created_by char(36),
   name varchar(50),
   description text,
   report_sql longtext,
   date_modified datetime,
   deleted tinyint(1),
   primary key (id)
);
```

Remember to place this in a file rather than creating the table directly on the database. By doing it that way you've got a record of what you've done, and you can replicate it again when you migrate to testing and then to live.

Before we leave the structure of the table it's worth noting that there are three mandatory fields:

- `id`—for the unique SugarCRM identification string
- `date_modified`—certain of the SugarCRM processes automatically update this
- `deleted`—no data is actually deleted from the SugarCRM database; however, records with `deleted` set to 1 will be ignored

And, of course, a primary key will be required—this is always the `id` field.

Next, you'll need to load some data into the table. In this case we're using the name, SQL and title for the reports that we've already used:

```
insert into ppi_reports
(id,  name,  description, report_sql)
values
( 'monthly_new_prelim_invest' , 'monthly_new_prelim_invest',
'Monthly New Preliminary Investigations',
'SELECT o.name, CONCAT(u.first_name, CONCAT('' '', u.last_name)) FROM
opportunities o, users u WHERE MONTH(o.date_entered) = MONTH(NOW())
AND o.assigned_user_id = u.id');

insert into ppi_reports
(id,  name,  description, report_sql)
values
( 'monthly_open_invest' , 'monthly_open_invest',
'Monthly Open Investigations',
'SELECT c.name, CONCAT(u.first_name, CONCAT('' '', u.last_name))
FROM cases c, users u WHERE MONTH(c.date_entered) = MONTH(NOW()) AND
c.assigned_user_id = u.id AND c.status <> ''Closed''');
```

There are a couple of things to take note of in the SQL:

- We've used the report name as the `id`. Normally SugarCRM would assign its own unique ID, but in this case we need to provide our own ID because we're entering the data directly onto the database rather than using the SugarCRM application—we'll see how to do that shortly.
- You'll notice that there are some double quotes used in the SQL—this allows us to enter a single quote into the database similar to **''Closed''** ending up as **'Closed'** on the database.

Once the data is loaded into the table we can make use of it in the module, so we don't have to edit `index.php` every time we need a new report:

```
if ($report == "monthly_new_prelim_invest")
{
...
}
else if ($report == "monthly_open_invest")
{
...
if ($report == "monthly_new_prelim_invest")
{
...
}
else if ($report == ...)
{
...
}
else
{
...
}
```

Now we can extract the information required for the report (i.e. the title and the SQL for the report) directly from the database:

```
if ($report)
{
  $sql = "select description, report_sql
         from ppi_reports where name='$report'";
  $result = mysql_query($sql);
  $report_title = mysql_result($result,0,0);
  $sql = mysql_result($result,0,1);
}
else
{
  $sql = "select 'Choose report'";
}
```

And, we can do similarly for `Menu.php`:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
$url = "index.php?module=ppi_reports&action=index&report=";
$sql = "select name, description from ppi_reports";
$result = mysql_query($sql);
$r=0;
while ($r < mysql_numrows($result))
{
  $name = mysql_result($result,$r,'Name');
  $description = mysql_result($result,$r,'Description');
  $module_menu[] = Array($url . $name, $description);
  $r++;
}
?>
```

At first glance this looks more complicated than the original file; however, it does mean that you won't have to edit it every time that you add a new report—the information will just be picked up automatically from the database.

Advanced Modules

We've now created a couple of very simple modules, and we've seen how easy the built-in SugarCRM functionality makes this. For example, we can query the database without having any knowledge of the connection details—we just have to tell SugarCRM to send the query, and then we're free to deal with the results in our module. We'll now move on to create a module that uses more of the functionality that's available to us, one that:

- Allows us to view and edit all existing reports
- Allows us to create new reports

The Initial Setup

By now you should be quite happy with the basic setup required for new modules, but it's worth running through the process once more. First you'll need to create a directory for your module, and then populate it with the mandatory files:

```
mkdir modules/ppi_report_manager
touch modules/ppi_report_manager/index.php
touch modules/ppi_report_manager/Forms.php
mkdir modules/ppi_report_manager/language
touch modules/ppi_report_manager/language/en_us.lang.php
```

With the directory structure in place you'll need to tell SugarCRM about the module by editing `include/modules.php` and adding:

```
$moduleList[] = 'ppi_report_manager';
```

Finally you'll need to modify `custom/include/language/en_us.lang.php` to add a title for the module:

```
$app_list_strings['moduleList']['ppi_report_manager']='Reports Manager';
```

So, that's the module in place. Next we need to think about the data that we're going to be using.

The Module's Data Schema—vardefs.php

We've already created the table (`ppi_reports`), and we've seen how easy it is to use the data stored in it. However, SugarCRM doesn't normally access the database directly—instead it has its own data schema for each of the modules, and this data schema, or dictionary, is defined in the `vardefs.php` file (there's one in each of the module directories). Each vardefs file contains a `$dictionary` array, and this contains the table name, as well as a set of sub-arrays—one for each of the fields to be used:

```
<?php
#Ensure that the file can only be accessed via SugarCRM
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
#Define the dictionary
$dictionary['ppi_report_manager'] = array(
   #Define the table to be used, their data types, labels, etc
    'table' => 'ppi_reports',
    'unified_search' => true,
    'comment' => 'Reports',
   #Define the fields to be used
    'fields' => array(
           'id' => array(
                  'name' => 'id',
                  'vname' => 'LBL_ID',
                  'required' => true,
                  'type' => 'id',
                  'reportable'=>false,
                  'comment' => 'Unique identifier' ),
           'description' => array(
                  'name' => 'description',
                  'vname' => 'LBL_DESCRIPTION',
                  'required' => false,
                  'type' => 'text',
                  'comment' => 'Report description' ),
           'report_sql' => array(
                  'name' => 'report_sql',
                  'vname' => 'LBL_REPORT_SQL',
                  'required' => false,
                  'type' => 'text',
                  'comment' => 'Report SQL' ),
           'name' => array(
                  'name' => 'name',
                  'vname' => 'LBL_NAME',
                  'required' => true,
                  'dbType' => 'varchar',
                  'type' => 'name',
                  'len' => 50,
                  'unified_search' => true,
                  'comment' => 'Report name' ),
           'assigned_user_id' => array(
                  'name' => 'assigned_user_id',
                  'rname' => 'user_name',
                  'id_name' => 'assigned_user_id',
                  'type' => 'assigned_user_name',
                  'vname' => 'LBL_ASSIGNED_USER_ID',
                  'required' => false,
                  'len' => 36,
                  'dbType' => 'id',
                  'table' => 'users',
                  'isnull' => false,
                  'reportable'=>true,
                  'comment' => 'User assigned to this report' ),
           'date_modified' =>  array (
                   'name' => 'date_modified',
                   'vname' => 'LBL_DATE_MODIFIED',
                   'type' => 'datetime',
                   'required' => false,
                   'comment' => 'Date record last modified' ),
           'deleted' => array (
                  'name' => 'deleted',
                  'vname' => 'LBL_DELETED',
                  'type' => 'bool',
                  'required' => true,
                  'reportable'=>false,
                  'comment' => 'Record deletion indicator',
           ),
    ),
);
?>
```

The dictionary is only half of the data model. The other half is the module's business object.

The Module's Business Object

We have the module's data dictionary in place, but we won't normally be accessing it directly—instead we make use of the module's business object. The business object does two things:

- Define any variables to be used
- Set up any required functionality

We do this by creating a class in a PHP file—in this case `ppi_report_manager.php` in the module's directory :

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
require_once('data/SugarBean.php');
require_once('include/utils.php');

class ppi_report_manager extends SugarBean
{
   var $id;
   var $description;
   var $report_sql;
   var $name;
   var $assigned_user_id;
   var $date_modified;
   var $deleted;

   var $table_name = "ppi_reports";
   var $module_dir = "ppi_report_manager";

   var $track_on_save=true;

   var $object_name = "ppi_report_manager";

   function ppi_report_manager()
        {
          parent::SugarBean();
        }
}
?>
```

If you read through the code you'll see that it:

- Makes use of the SugarCRM SugarBean file—this incorporates your vardefs file, and sets up the business object itself
- Loads all of the utilities that you'll need for working with your business object

Now, it's worth noting that you can call this file anything you like, but the normally accepted naming convention is to use either the module name, or the singular of the module name—for example the business object for Opportunities is `opportunity.php`.

Finally, you'll need to give SugarCRM the details of your new file.

Registering the Business Object

You'll remember that we needed to tell SugarCRM about the module by editing `include/modules.php` and adding:

```
$moduleList[] = 'ppi_report_manager';
```

Well, we use the same file to register the business object:

```
$beanList['ppi_report_manager'] = 'ppi_report_manager';
$beanFiles['ppi_report_manager'] =
  'modules/ppi_report_manager/ppi_report_manager.php';
```

The business object will now be incorporated into SugarCRM, and you can start making use of it; however, there is just a little tidying up that needs to be done—the setting up of the language file.

The Module's Language File

We now need to turn back to one of the module's required files: `language/en_us.lang.php`. It's here that we define the default terminology to be used by the module.

If you look at the `vardefs.php` file you'll see that each field contains a variable `vname`, for example `assigned_user_id` has the `vname` `LBL_ASSIGNED_USER_ID`. You must assign some text to this `vname` in `language/en_us.lang.php`, and it's this text that SugarCRM will then display on the screen:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
$mod_strings = array (
    'LBL_MODULE_NAME' => 'ppi_report_manager',
    'LBL_MODULE_TITLE' => 'Report Manager',
    'LBL_ID' => 'ID',
    'LBL_DESCRIPTION' => 'Title',
    'LBL_REPORT_SQL' => 'SQL',
    'LBL_NAME' => 'Name',
    'LBL_ASSIGNED_USER_ID' => 'Owner',
);
?>
```

So, in the above example SugarCRM will display the text 'Owner' on the screen where ever `assigned_user_id` is used.

That's all the background setting up that the module needs—now we can turn our attention to something that we can actually see via the web browser.

The Module's List View

When you click on any module tab in SugarCRM then the first thing that you'll see is the List View—so, for example, if you go to Opportunities then you'll see the list of all of the opportunities currently in the system. We'll now look at doing exactly the same for our new module.

Selecting the Fields to be Displayed

The first thing that you must do is to decide which fields are to be displayed in the List View. Once, you know which fields you want, then you'll need to tell SugarCRM about them in the module's `metadata/listviewdefs.php` file:

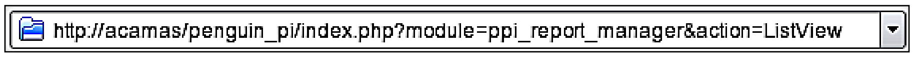
```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
$listViewDefs['Report'] = array(
    'NAME' => array(
            'width' => '50',
            'label' => 'LBL_NAME',
            'default' => true),
    'DESCRIPTION' => array(
            'width' => '50',
            'label' => 'LBL_DESCRIPTION',
            'default' => true),
    'REPORT_SQL' => array(
            'width' => '50',
            'label' => 'LBL_REPORT_SQL',
            'default' => true), );
?>
```

Creating the List View

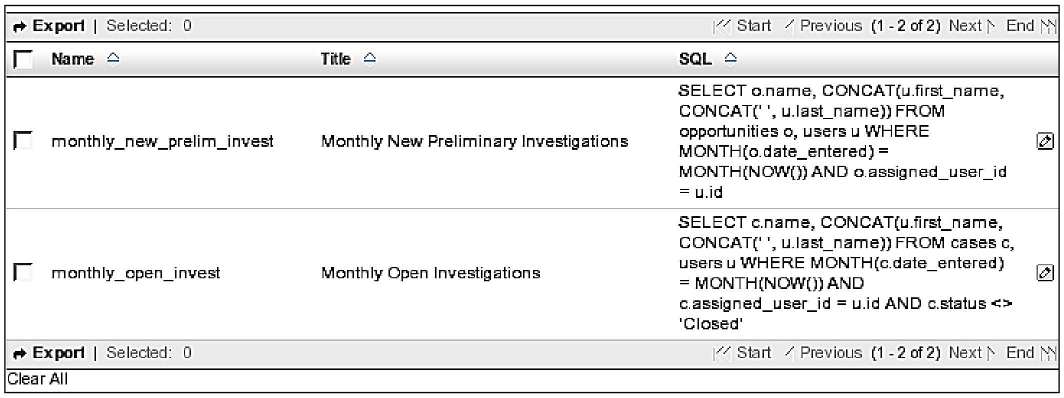
You've actually done all of the hard work—all you have to do now is to create a PHP file (normally named `ListView.php`), which will make use of your business object and some SugarCRM functionality:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
require_once ('modules/ppi_report_manager/ppi_report_manager.php');
require_once('include/ListView/ListViewSmarty.php');
require_once('modules/ppi_report_manager/metadata/listviewdefs.php');

$seedReport = new ppi_report_manager();
$lv = new ListViewSmarty();
$lv->displayColumns = $listViewDefs['Report'];
$lv->setup($seedReport,
    'include/ListView/ListViewGeneric.tpl', $where,
    $listViewDefs['Report']);
echo $lv->display();
?>
```

Now you can view the result:



http://acamas/penguin_pi/index.php?module=ppi_report_manager&action=ListView



Name	Title	SQL
monthly_new_prelim_invest	Monthly New Preliminary Investigations	SELECT o.name, CONCAT(u.first_name, CONCAT(' ', u.last_name)) FROM opportunities o, users u WHERE MONTH(o.date_entered) = MONTH(NOW()) AND o.assigned_user_id = u.id
monthly_open_invest	Monthly Open Investigations	SELECT c.name, CONCAT(u.first_name, CONCAT(' ', u.last_name)) FROM cases c, users u WHERE MONTH(c.date_entered) = MONTH(NOW()) AND c.assigned_user_id = u.id AND c.status <> 'Closed'

Making the List View the Default View

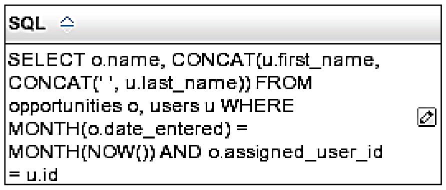
Having created the List View we need to make it the default screen. To do this you'll need to edit the module's `index.php` file:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

echo get_module_title( $mod_strings['LBL_MODULE_NAME'],
                                    $mod_strings['LBL_MODULE_TITLE'],
true);

include ("modules/$currentModule/ListView.php");
?>
```

Now that we can see the list of reports the next logical thing to do is to edit existing ones and add new ones. To do that we need to add an Edit View.

The Modules Edit View

If you look on the right of the List View then you'll see the edit button:



SQL

SELECT o.name, CONCAT(u.first_name, CONCAT(' ', u.last_name)) FROM opportunities o, users u WHERE MONTH(o.date_entered) = MONTH(NOW()) AND o.assigned_user_id = u.id

If you click on this button then SugarCRM will take you to the Edit View—once you've created it, of course.

The EditView.php File

This time you have no choice as to the name for the PHP file that you create. It must be named `EditView.php`. However, as before you make use of a lot of built-in functionality to minimize the amount of coding that you need to do:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

require_once('XTemplate/xtpl.php');
require_once('modules/ppi_report_manager/ppi_report_manager.php');
```

```
$focus = new ppi_report_manager();

//Load the data for the fields
if(isset($_REQUEST['record']))
{
    $focus->retrieve($_REQUEST['record']);
    $focus->format_all_fields();
}

echo get_module_title($mod_strings['LBL_MODULE_NAME'],
             $mod_strings['LBL_MODULE_NAME'].": ".$focus->name, true);

//Load the edit form
$xtpl=new XTemplate ('modules/ppi_report_manager/EditView.html');

//Define the Save and Cancel buttons
$xtpl->assign("MOD", $mod_strings);
$xtpl->assign("APP", $app_strings);

//Create a popup for the Assigned user
$json = getJSONobj();
$popup_request_data = array(
   'call_back_function' => 'set_return',
   'form_name' => 'EditView',
   'field_to_name_array' => array(
          'id' => 'assigned_user_id',
          'user_name' => 'assigned_user_name',
          ),
   );
$xtpl->assign('encoded_users_popup_request_data',
   $json->encode($popup_request_data));

$xtpl->assign("ID", $focus->id);
$xtpl->assign("NAME", $focus->name);
$xtpl->assign("DESCRIPTION", $focus->description);
$xtpl->assign("REPORT_SQL", $focus->report_sql);
$xtpl->assign("ASSIGNED_USER_ID",$focus->assigned_user_id);
$xtpl->assign("ASSIGNED_USER_NAME",
   get_assigned_user_name ($focus->assigned_user_id));

//Output to the screen
$xtpl->parse("main");
$xtpl->out("main");
?>
```

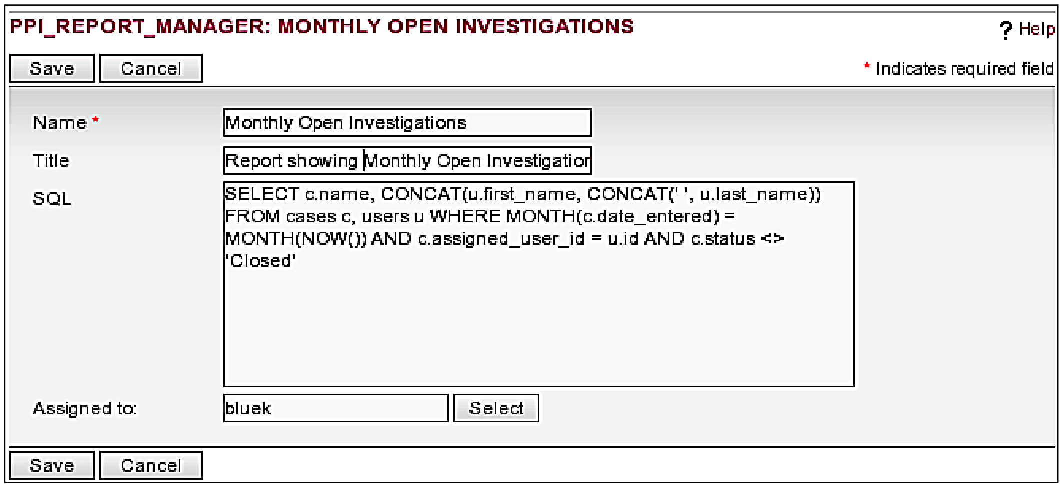
If you look through the code then you'll see that it references a file that doesn't exist yet—`modules/ppi_report_manager/EditView.html`.

The EditView.html File

Your module's `EditView.html` file is used for designing the layout of your edit form:

```
<table width="100%" border="0" cellspacing="0" cellpadding="0">
<tr>
<td width="15%" class="dataLabel"><span sugar='slot1'>{MOD.LBL_NAME}
<span  class="required">{APP.LBL_REQUIRED_SYMBOL}</span></span
sugar='slot'></td>
<td width="35%" class="dataField"><span sugar='slot1b'><input
name='name' type="text" tabindex='1' size='35' maxlength='50'
value="{NAME}"></span sugar='slot'></td>
</tr>
<tr>
<td  valign="top" class="dataLabel"><span sugar='slot2'>{MOD.LBL_
DESCRIPTION}</span sugar='slot'></td>
<td colspan="4" class="dataField"><span sugar='slot2b'><input
name='description' type="text" tabindex='1' size='35' maxlength='50'
value="{DESCRIPTION}"></span sugar='slot'></td>
</tr>
<tr>
<td  valign="top" class="dataLabel"><span sugar='slot3'>{MOD.LBL_
REPORT_SQL}</span sugar='slot'></td>
<td colspan="4" class="dataField"><span sugar='slot3b'><textarea
name='report_sql' tabindex='3' cols="60" rows="8">{REPORT_SQL}</
textarea></span sugar='slot'></td>
</tr>
<tr>
<td  class="dataLabel"><span sugar='slot13'>{APP.LBL_ASSIGNED_TO}</
span sugar='slot'></td>
<td class="dataField"><span sugar='slot13b'><input class="sqsEnabled"
tabindex="1" autocomplete="off" id="assigned_user_name"
name='assigned_user_name' type="text" value="{ASSIGNED_USER_
NAME}"><input id='assigned_user_id' name='assigned_user_id'
type="hidden" value="{ASSIGNED_USER_ID}" />
<input title="{APP.LBL_SELECT_BUTTON_TITLE}" accessKey="{APP.
LBL_SELECT_BUTTON_KEY}" type="button" tabindex='1' class="button"
value='{APP.LBL_SELECT_BUTTON_LABEL}' name=btn1 onclick='open_
popup("Users", 600, 400, "", true, false, {encoded_users_popup_
request_data});' /></span sugar='slot'>
</td>
</tr>
</table>
```

The end result is a form in which you can edit the details for any existing report:



PPI_REPORT_MANAGER: MONTHLY OPEN INVESTIGATIONS

Name *	Monthly Open Investigations
Title	Report showing Monthly Open Investigatior
SQL	SELECT c.name, CONCAT(u.first_name, CONCAT(' ', u.last_name)) FROM cases c, users u WHERE MONTH(c.date_entered) = MONTH(NOW()) AND c.assigned_user_id = u.id AND c.status <> 'Closed'
Assigned to:	bluek

Of course, now that you've edited the report you need to be able to save it.

The Module's Save File

Your module's save file must be called `Save.php`, and should contain any preprocessing that your data may need before sending to the database:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

require_once('include/utils.php');
require_once('modules/ppi_report_manager/vardefs.php');

//Function to prepare data for the database
function format_mysql_text ($ip_text) {
  return mysql_real_escape_string(
    html_entity_decode(str_replace("&#039;","'",$ip_text)));
}

//Identify the fields to be loaded
$table = $dictionary['ppi_report_manager']['table'];
while (list($key, $value) = each($dictionary['ppi_report_
manager']['fields'])) {
  if ($key != "id") {
    if ($_REQUEST[$key] != "") {
      $field_list[] = $key;
```

```
        }
      }
    }
    /*Create a SQL statement according to whether this is an insert or an
    update*/
    if ($_REQUEST['record'] == "")
    {
        $field_names .= "id";
        $field_values .= "'" . create_guid() . "'";
        foreach ($field_list as $key)
        {
         $field_names .= "," . $key;
         $field_values .= ",'" . format_mysql_text($_REQUEST[$key]) . "'";
        }
          $sql = "insert into $table";
          $sql .= "(" . $field_names . ")";
          $sql .= " values ";
          $sql .= " (". $field_values . ")";
    }
    else
    {
      foreach ($field_list as $key)
    {
        if ($sql_body != "")
        {
          $sql_body .=  ",";
        }
        $sql_body .= $key . " = '" . format_mysql_text(
                                                    $_REQUEST[$key]) ."'";
    }
      $sql = "update $table set ";
      $sql .= $sql_body;
      $sql .= " where id ='" . $_REQUEST['record'] ."'";
    }
    //Send the SQL to the database
    $db->query($sql);
    //Return to the index page
    header
      ("Location: index.php?module=".$_REQUEST['module']."&action=index")
    ;
    ?>
```

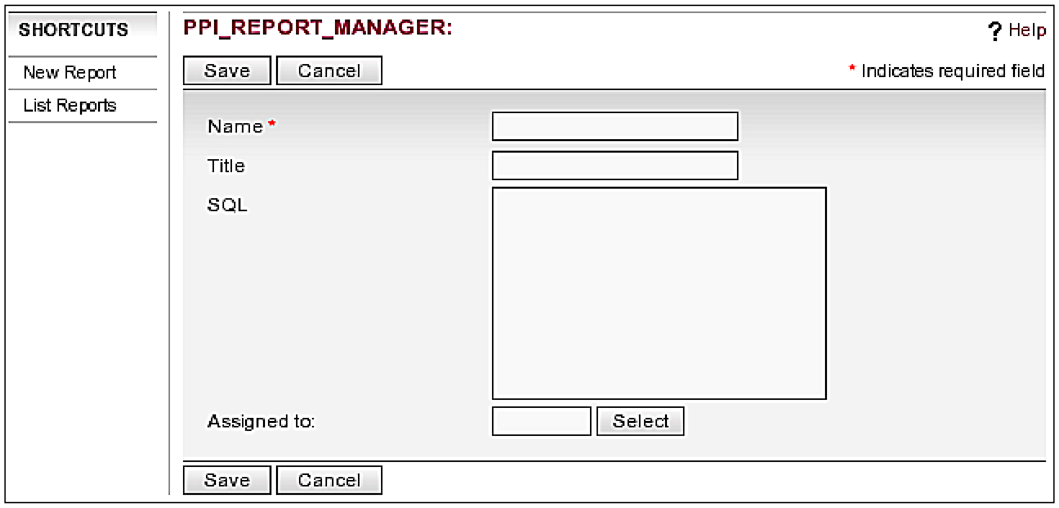
You'll see from the code that the save file handles both update and insert statements, and that the script returns you to the module index at the end of the process.

Creating New Reports

Having seen how to edit the existing reports you'll be wondering how to create new ones. You'll be pleased to know that we've done all of the hard work. All you have to do now is call the Edit View without any input details, and we can do this just by editing the `Menu.php` file for the module:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
$module_menu[] =
  Array("index.php?module=ppi_report_manager&action=EditView",
    "New Report");
$module_menu[] =
  Array("index.php?module=ppi_report_manager&action=index",
    "List Reports");
?>
```

Now you can either edit existing records or create a new one:



You can now go back to Korora and tell her that you don't have to create any new reports for her—she can do it all for herself.

Summary

In this chapter we've seen that you have two options when it comes to new modules.

You can incorporate a third-party module (if it does the job that you want carried out) and you also know the procedure for developing your own modules from scratch.

We'll be looking at other aspects of module development in Chapter 10, but before that we'll look at a contentious issue for any organization—the workflow.

9 Developing a Custom Workflow within SugarCRM

We've now come to a very contentious area—Workflow. It's not that anyone disagrees about what workflow is:

Workflow is about getting the right work to the right people at the right time, repeatedly – and knowing you have done so. Workflow is human-centric. First and foremost, workflow is a human activity that is made by and for those who use it: workflow is something that can easily be handled and understood by human beings.

UK Enterprise Workflow National e-Government Project – Workflow from a Business Perspective

Well, that sounds good, but the problems start to occur when you ask people to consider workflow in their organization, and there are usually a few main issues to deal with:

- You'll find that people are normally experts in their own fields—there are often very few people who have an overview of the whole process that you're trying to map.
- Sections of a large organization will often have different ways of carrying out the same overall process.
- People don't really like to be told how to do their jobs—they especially don't like to have any extra processes imposed on them for now obvious reason—well, would you?
- Talk of 'improved utilization of resources', 'improved performance monitoring', and such like can soon alienate the staff who are going to be using the system. They'll soon start using terms such as 'Big Brother'.

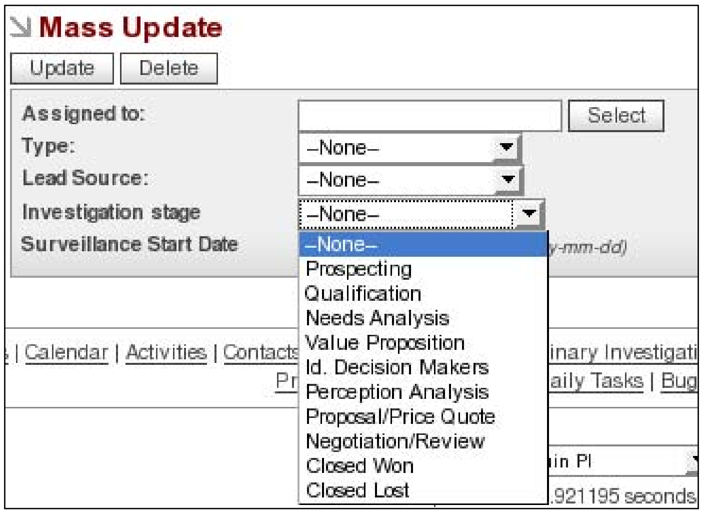
How you are able to deal with these will depend on your organization and the people that are available to you. At least once you've read this chapter you'll know that, once you've overcome those problems, the workflow itself will be easy.

A Very Simple Workflow

In our simple workflow we'll assume that each task is carried out by one person at a time, and that all tasks are done sequentially (i.e. none are done in parallel). So, we'll look at the PPI Preliminary Investigation which, as you remember, maps to the standard SugarCRM Opportunity. Also, in this example, we're going to have a different person carrying out each one of the Investigation stages.

Setting up the Process Stages

If you look at SugarCRM then you'll see that by default none of the stages are related to investigations—they're all named using standard CRM terms:



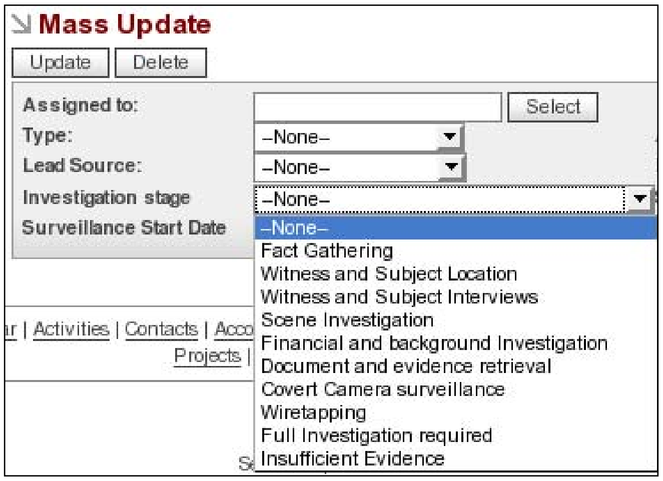
Assigned to:	
Type:	–None–
Lead Source:	–None–
Investigation stage	–None–
Surveillance Start Date	(yyyy-mm-dd)

–None–
Prospecting
Qualification
Needs Analysis
Value Proposition
Id. Decision Makers
Perception Analysis
Proposal/Price Quote
Negotiation/Review
Closed Won
Closed Lost

Obviously the first thing to do is to decide what the preliminary investigation stages actually are, and then map these to the SugarCRM stages. You'll realize that you'll need to edit the `custom/include/langauge/en_us.lang.php` file:

```
$app_list_strings['sales_stage_dom']=array (
    'Prospecting' => 'Fact Gathering',
    'Qualification' => 'Witness and Subject Location',
    'Needs Analysis' => 'Witness and Subject Interviews',
    'Value Proposition' => 'Scene Investigation',
    'Id. Decision Makers' => 'Financial and background Investigation',
    'Perception Analysis' => 'Document and evidence retrieval',
    'Proposal/Price Quote' => 'Covert Camera surveillance',
    'Negotiation/Review' => 'Wiretapping',
    'Closed Won' => 'Full Investigation required',
    'Closed Lost' => 'Insufficient Evidence',
);
```

Don't forget that you can also do this via Studio. However, once you've added your mapping into `custom/include/langauge/en_us.lang.php` file, and refresh your browser, then you'll see the new stages:



Assigned to:	
Type:	–None–
Lead Source:	–None–
Investigation stage	–None–
Surveillance Start Date	

–None–
Fact Gathering
Witness and Subject Location
Witness and Subject Interviews
Scene Investigation
Financial and background Investigation
Document and evidence retrieval
Covert Camera surveillance
Wiretapping
Full Investigation required
Insufficient Evidence

Now that our stages are set up we need to know who'll be carrying out each one.

Deciding Who Does What

In our simple workflow there may not be the need to do anything further. Each person just needs to know who does what next:

Preliminary Investigation Stage	Investigator	User Name
Fact Gathering	Fran Varady	varadyf
Witness and Subject Location	William Monk	monkw
Witness and Subject Interviews	Charlotte Pitt	pittc
Scene Investigation	David Brock	brockd
Financial and background Investigation	Guido Brunetti	brunettig
Document and evidence retrieval	Luke Thanet	thanetl

Preliminary Investigation Stage	Investigator	User Name
Covert Camera surveillance	Kurt Wallander	wallanderk
Wiretapping	Maisie Dobbs	dobbsm
Full Investigation required	Korora Blue	bluek
Insufficient Evidence	Korora Blue	bluek

For example, once Kurt finishes the 'Covert Camera surveillance' stage then he just needs to update the Preliminary Investigation so that the stage is set to 'Wiretapping' and the assigned user as 'dobbsm'.

However, things are rarely as simple as that. It's much more likely that:

- Investigations may be based on geographical locations, so that the above table may only apply to investigations based in London. Investigations based in New York follow the same process but with a different set of staff.
- On Mondays Fran does 'Witness and Subject Location' and William does 'Fact Gathering'.

This means, of course, that we need to be using some businesses rules.

Introducing Business Rules

We saw how to start implementing business rules in Chapter 4 when we made use of SugarCRM's logic hooks, and it's those that we are going to make use of again. Just to recap—you'll remember that there are six 'triggers' that will cause the logic hooks to fire:

- `after_retrieve`
- `before_save`
- `before_delete`
- `after_delete`
- `before_undelete`
- `after_undelete`

And the logic hooks are stored in `custom/modules/<module name>/logic_hook.php`, so for 'Preliminary Inquiries' this will be `custom/modules/Opportunities/logic_hook.php`. You'll also remember, of course, that the logic hook file needs to contain:

- The priority of the business rule
- The name of the businesses rule

- The file containing the business rule
- The business rule class
- The business rule function

So, `custom/modules/Opportunities/logic_hook.php` needs to contain something like:

```
<?php
#As always ensure that the file can only be accessed through SugarCRM
if(!defined('sugarEntry') || !sugarEntry) die(
                                            'Not A Valid Entry Point');
$hook_array = Array(); #Create an array

$hook_array['before_save'] = Array();
$hook_array['before_save'][] = Array(1, 'ppi_workflow',
  'custom/include/ppi_workflow.php',
  'ppi_workflow', 'ppi_workflow');
?>
```

Next we'll need the file that logic hook will be calling, but to start with this can be very basic—so, `custom/include/ppi_workflow.php` just needs to contain something like:

```
<?php
#Define the entry point
if(!defined('sugarEntry') || !sugarEntry) die(
                                            'Not A Valid Entry Point');
#Load any required files
require_once('data/SugarBean.php');
require_once('modules/Opportunities/Opportunity.php');

#Define the class
class ppi_workflow
{
  function ppi_workflow (&$bean, $event, $arguments)
  {

  }
}
?>
```

With those two files set up as above nothing obvious will change in the operation of SugarCRM—the logic hook will fire, but we haven't told it to do anything, and so that what we'll do now.

When the logic hook does run (i.e. when any Primary Investigation is saved) we would want it to:

- Check to see what stage we're now at
- Define the assigned user accordingly

All of the relevant information (i.e. the new stage) is passed to the logic hook by means of the `$bean` object, and we can obtain the stage from `$bean->sales_stage`. Now all we have to do is combine this with PHP's switch statement into the `ppi_workflow` function:

```
       switch ($bean->sales_stage)
   {
         case "Prospecting":
           $assigned_user = "varadyf";
           break;
         case "Qualification":
           $assigned_user = "monkw";
           break;
         case "Needs Analysis":
           $assigned_user = "pittc";
           break;
         case "Value Proposition":
           $assigned_user = "brockd";
           break;
         case "Id. Decision Makers":
           $assigned_user = "brunettig";
           break;
         case "Perception Analysis":
           $assigned_user = "thanetl";
           break;
         case "Proposal/Price Quote":
           $assigned_user = "wallanderk";
           break;
         case "Negotiation/Review":
           $assigned_user = "dobbsm";
           break;
         case "Closed Won":
           $assigned_user = "bluek";
           break;
         case "Closed Lost":
           $assigned_user = "bluek";
           break;
       }
```

You'll notice from the code that we must use the original SugarCRM sales stage terms and not our new mapping—that only appears on the screen.

Next we'll have to add the code to update `$bean->assigned_user_id` with the ID of our new user:

```
global $db;

$sql =
  "select id from users where user_name = '" . $assigned_user ."'";
$result = $db->query($sql);
$bean->assigned_user_id = mysql_result($result,0,0);
```

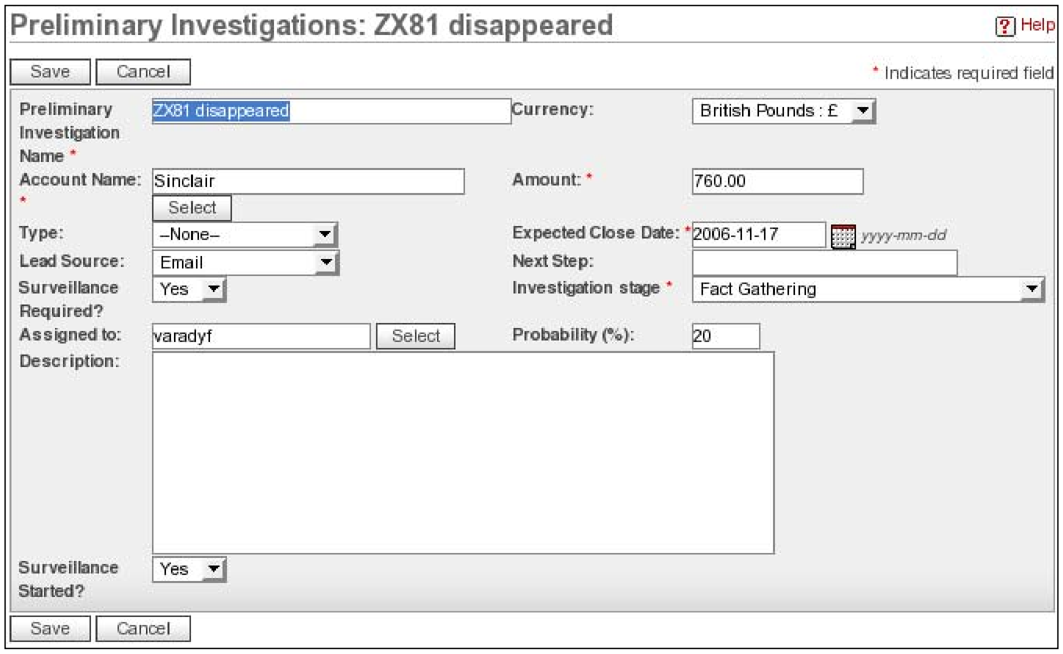
With the code in place, if you now change the Investigation (or Sales) stage, and then save the Preliminary Investigation (or Opportunity) then you'll see that the assigned user is automatically updated for you.

However, this is still only a semi-automatic process—the correct person for the stage is selected correctly, but only if the stage is selected manually. The process running correctly still depends on someone telling SugarCRM what that next stage is. Obviously the next step is to move from stage to stage automatically.

Completing the Automated Workflow

At the moment we're relying on a user telling the application which stage to move to next. However, it would be much better for the user to tell SugarCRM that the current stage has been completed, and then for the business rules to decide which stage should be carried out next. We want to keep it simple and therefore an 'Investigation Stage Complete' checkbox will do the job.

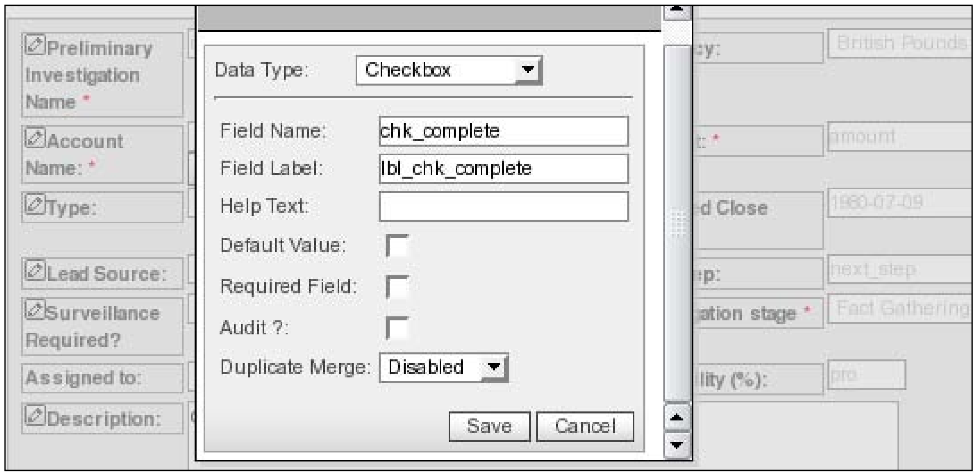
If you look at the edit view for any of the existing Opportunities then you'll see that there's nothing that can really be renamed to represent our 'Investigation Stage Complete':



Preliminary Investigations: ZX81 disappeared

Preliminary Investigation Name *	ZX81 disappeared	Currency:	British Pounds : £
Account Name: *	Sinclair	Amount: *	760.00
Type:	--None--	Expected Close Date: *	2006-11-17 yyyy-mm-dd
Lead Source:	Email	Next Step:	
Surveillance Required?	Yes	Investigation stage *	Fact Gathering
Assigned to:	varadyf	Probability (%):	20
Description:			
Surveillance Started?	Yes		

However, as we saw in Chapter 3, we can use the SugarCRM Studio to add the field that we're going to need:

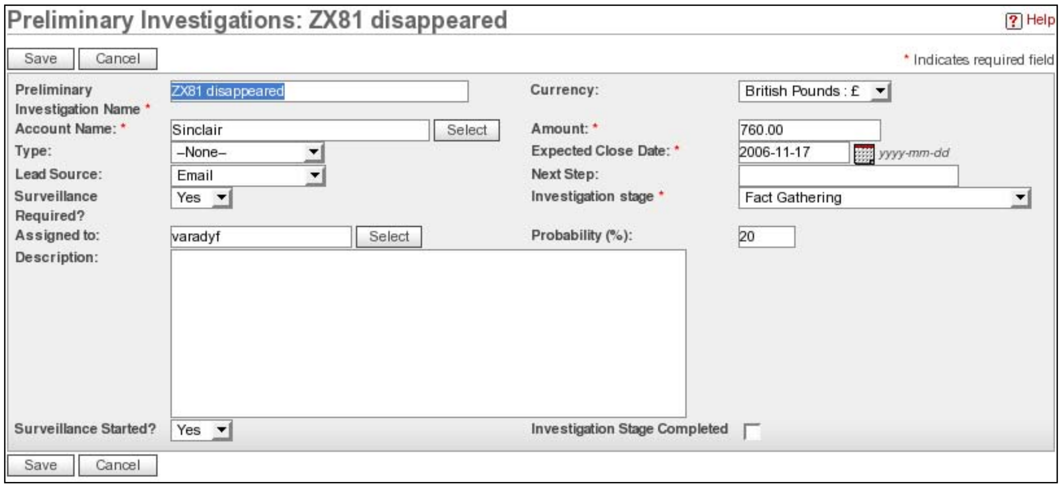


Data Type:	Checkbox
Field Name:	chk_complete
Field Label:	lbl_chk_complete
Help Text:	
Default Value:	☐
Required Field:	☐
Audit ?:	☐
Duplicate Merge:	Disabled

After adding the custom field itself we'll need to add text for the field label into `custom/modules/Opportunities/language/en_us.lang.php`:

```
$mod_strings['lbl_chk_complete_c_10'] = "Investigation Stage
Completed";
```

And then we're ready to see the new edit view:


Preliminary Investigations: ZX81 disappeared

Preliminary Investigation Name *	ZX81 disappeared	Currency:	British Pounds : £
Account Name: *	Sinclair	Amount: *	760.00
Type:	–None–	Expected Close Date: *	2006-11-17 yyyy-mm-dd
Lead Source:	Email	Next Step:	
Surveillance Required?	Yes	Investigation stage *	Fact Gathering
Assigned to:	varadyf	Probability (%):	20
Description:			
Surveillance Started?	Yes	Investigation Stage Completed	☐

We can now go back to our code (in `custom/include/ppi_workflow.php`), and we can place all of our functionality within an `if` statement:

```
if ( $bean->chk_complete_c == 1 )
{
  switch ($bean->sales_stage)
  {
    /* etc, etc, etc */
  }
}
```

Our logic hook will, of course, fire every time a save is made—but our business rule will only be implemented if the **Investigation Stage Completed** box is ticked. So now we need the code that will define the process itself, and you would need to place this before the code for deciding who the assigned user is:

```
      switch ($bean->sales_stage)
{
        case "Prospecting":
          $bean->sales_stage = "Qualification";
          break;
        case "Qualification":
          $bean->sales_stage = "Needs Analysis";
          break;
        case "Needs Analysis":
          $bean->sales_stage = "Value Proposition";
          break;
        case "Value Proposition":
          $bean->sales_stage = "Id. Decision Makers";
          break;
        case "Id. Decision Makers":
          $bean->sales_stage = "Perception Analysis";
          break;
        case "Perception Analysis":
          $bean->sales_stage = "Proposal/Price Quote";
          break;
        case "Proposal/Price Quote":
          $bean->sales_stage = "Negotiation/Review";
          break;
        case "Negotiation/Review":
          $bean->sales_stage = "Closed Won";
          break;
      }
      //Now decide who the assigned user is...
```

And finally we need to reset the completed status back to 0:

```
$bean->chk_complete_c = 0;
```

You'll notice that we've not taken the process all the way to the final stage—this is because the final two stages are 'Closed won' or 'Closed lost' (in PPI speak—'Full Investigation required' and 'Insufficient Evidence'). Korora will need to make that decision herself.

However, the important thing is that you can see just how easy it is to set up a simple process (not that any process is ever simple).

Moving the Rules into the Database

Now that we've proved how easy it is to set up a workflow within SugarCRM you're probably wondering how we can improve the process. Well, the first thing we can do is move the rules onto the database—meaning, of course, that we won't have to edit the files every time a change is made to the workflow itself.

Add a Custom Table

Your first step should be to decide what fields you're going to require for your workflow; however, we'll start by just keeping it to the basics:

```
create table ppi_workflow
(
    id char(36),
    user_id char(36),
    process_step varchar(50),
    predecessor varchar(50),
    primary key (id)
);
```

Don't forget that if you save this to a file then you have a record of what you've done, and you can use the file to create the table from the command line:

```
mysql -uroot -p<password> penguin_pi < create_ppi_workflow.sql
```

With the table in place we now need to think about entering the workflow details.

Create the Workflow Module

You can, of course load the data you want from the command line, but it's probably just as easy (certainly in the long run) to add a module to help you do the job. Your first step is to create the directory and obligatory files:

```
mkdir modules/ppi_workflow
touch modules/ppi_workflow/index.php
touch modules/ppi_workflow/Forms.php
mkdir modules/ppi_workflow/language
touch modules/ppi_workflow/language/en_us.lang.php
```

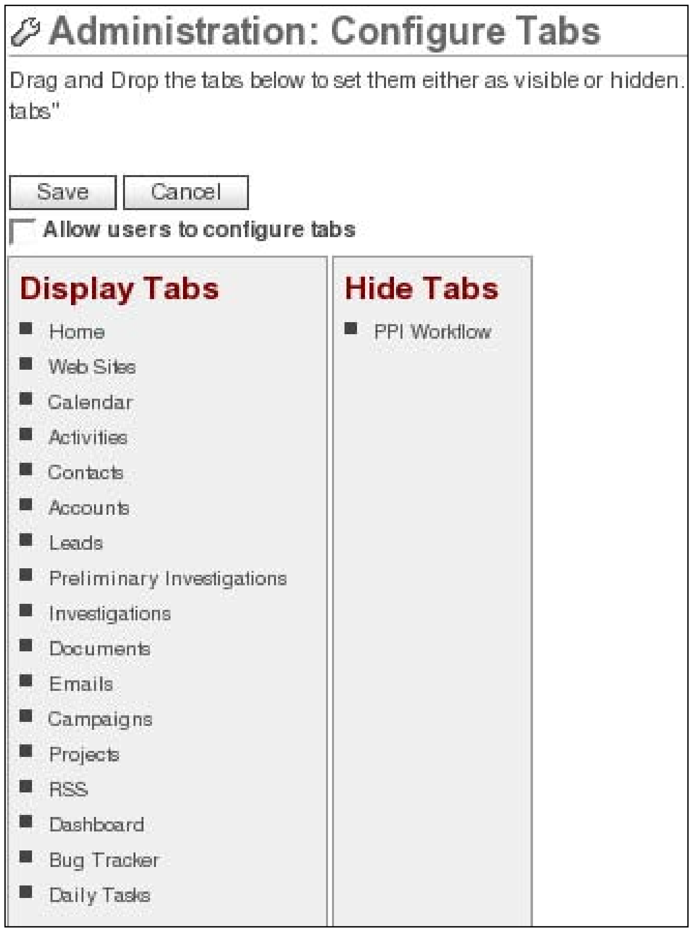
As always add the module to `include/modules.php`:

```
$moduleList[] = 'ppi_workflow';
```

And place it's title in `custom/include/language/en_us.lang.php`:

```
$app_list_strings['moduleList']['ppi_workflow'] = 'PPI Workflow';
```

Finally, don't forget—you may have to log on as the administrator so that you can move 'PPI Workflow' from **Hide Tabs** to **Display Tabs**:


Administration: Configure Tabs

Drag and Drop the tabs below to set them either as visible or hidden. tabs"

Allow users to configure tabs

Display Tabs

- Home
- Web Sites
- Calendar
- Activities
- Contacts
- Accounts
- Leads
- Preliminary Investigations
- Investigations
- Documents
- Emails
- Campaigns
- Projects
- RSS
- Dashboard
- Bug Tracker
- Daily Tasks

Hide Tabs

- PPI Workflow

That's the blank module in place—now it's time to think about doing something with it—in this case it will handle the data entry for us.

Building a Data Input Module

You're now ready to add the workings of the module, and this time we're going to do everything within one file—`index.php`:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
                                                          Point');
```

Having ensured that the module can only be access through the SugarCRM we can initialize the database:

```
$db=$GLOBALS['db']; #Not always essential, but is safer
```

and then deal with any requests that we send to the module:

```
if ($_REQUEST['assigned_user_id'])
{
  $sql = "insert into ppi_workflow (id, user_id,
                                        process_step,predecessor)";
  $sql .= " values ";
  $sql .= "('". create_guid() ."'";
  $sql .= ",'".$_REQUEST['assigned_user_id']."'";
  $sql .= ",'".$_REQUEST['prelim_stage']."'";
  $sql .= ",'".$_REQUEST['predecessor']."')";
  $db->query( $sql);
}
?>
```

The above code will save any new information to the database, and then we'll want to display the contents of the table to the screen:

```
<table width=100%>
<tr><td><b>Assigned User</b></td><td><b>Process step</b></
td><td><b>Predecessor</b></td></tr>
<?php
$sql = "select u.user_name, w.process_step,w.predecessor";
$sql .= " from users u,ppi_workflow w";
$sql .= " where u.id = w.user_id";
$result =  $db->query($sql);
$r=0;
while ($r < mysql_numrows($result))
{
  $user_name = mysql_result($result,$r,0);
  $ppi_stage =
    $app_list_strings['sales_stage_dom'][mysql_result($result,$r,1)];
  $predecessor =
    $app_list_strings['sales_stage_dom'][mysql_result($result,$r,2)];
  $op = "<tr>";
  $op .= "<td>" . $user_name . "</td>";
  $op .= "<td>" . $ppi_stage . "</td>";
  $op .= "<td>" . $predecessor . "</td>";
  $op .= "</tr>";
  echo $op;
  $r++;
}
?>
</table>
<hr>
```

Finally we can add a form for adding new details:

```
<form action="index.php?module=ppi_workflow&action=index" method=post>
<table>
<tr>
```

You'll notice that we call the module by assigning it to the form action, and then we need to go on to create a combo-box containing the list of users by querying the database:

```
<td>Assigned User</td>
<td>
<!-- Build the combo for the assigned user -->
<select name='assigned_user_id'>
<?php
$sql="select id,user_name from users where id <> '1'";
$sql .= " and deleted = 0 and status = 'Active'";
$result =  $db->query($sql);
$r=0;
```

And we create the combo by looping through the results:

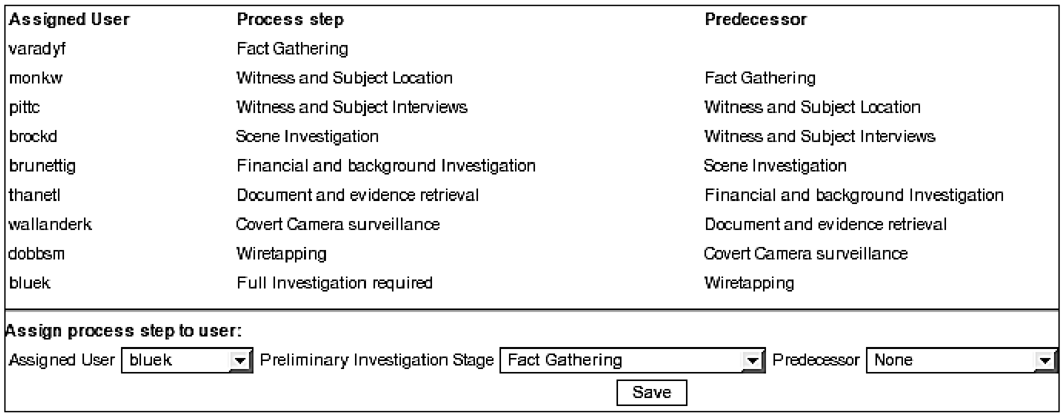
```
while ($r < mysql_numrows($result))
{
  $op = "<option value='";
  $op .= mysql_result($result,$r,0) . "'>";
  $op .= mysql_result($result,$r,1);
  echo $op;
  $r++;
}
?>
</select>
</td>
```

However, to create combo-boxes containing a list of the investigation stages we use the `$app_list_strings['sales_stage_dom']` array with a `for each` statement:

```
<td>Preliminary Investigation Stage</td>
<td>
<select name='prelim_stage'>
<?php
foreach ($app_list_strings['sales_stage_dom'] as $key => $value)
{
  echo "<option value='" . $key . "'>$value";
}
?>
</select>
</td>
```

```
<td>Predecessor</td>
<td>
<select name='predecessor'>
<option value=NONE>None
<?php
foreach ($app_list_strings['sales_stage_dom'] as $key => $value)
{
 echo "<option value='" . $key . "'>$value";
}
?>
</select>
</td>

</tr>
<tr><td colspan=4 align=center><input type=submit value="Save"></td></
tr>
</form>
```

Once you've saved `index.php` then you can access the module via your web browser where you'll see that we have a simple means of assigning process steps to users:



Assigned User	Process step	Predecessor
varadyf	Fact Gathering	
monkw	Witness and Subject Location	Fact Gathering
pittc	Witness and Subject Interviews	Witness and Subject Location
brockd	Scene Investigation	Witness and Subject Interviews
brunettig	Financial and background Investigation	Scene Investigation
thanetl	Document and evidence retrieval	Financial and background Investigation
wallanderk	Covert Camera surveillance	Document and evidence retrieval
dobbsm	Wiretapping	Covert Camera surveillance
bluek	Full Investigation required	Wiretapping

Assign process step to user:

Assigned User: bluek Preliminary Investigation Stage: Fact Gathering Predecessor: None

Once you've assigned each step to a user then you can start thinking about modifying the way that the business rule carries out its decision making process.

Making Use of the Rules in the Database

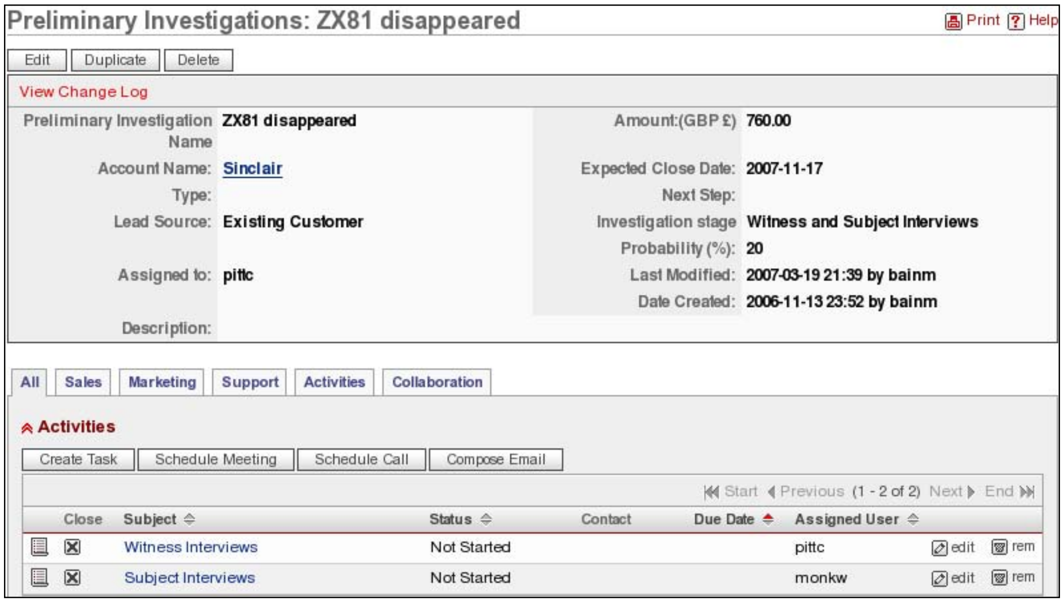
Now that we've got the set of rules on the database the decision making process becomes much easier. All we have to do is to query the database to find the process step for which the current step is the predecessor. Then it's just a matter of updating the assigned user and sales stage accordingly:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
require_once('data/SugarBean.php');
require_once('modules/Opportunities/Opportunity.php');

class ppi_workflow {
  function ppi_workflow (&$bean, $event, $arguments)
{
    global $db;

    if ( $bean->chk_complete_c == 1 )
    {
      $sql = "select user_id, process_step from ppi_workflow";
      $sql .= " where predecessor = '" . $bean->sales_stage . "'";
      $result = $db->query($sql);

      $bean->assigned_user_id = mysql_result($result,0,0);
      $bean->sales_stage = mysql_result($result,0,1);
      $bean->chk_complete_c = 0;
    }
  }
}
?>
```

So that's a simple workflow in place—simple because all tasks are done sequentially and by one person at a time. However, as we all know, that doesn't really represent the real world. In the real world you'll have a team of people involved, and they'll all be working on a project at the same time. Obviously what we need next is the ability to carry out parallel processing.

Parallel Tasks

In our scenario we've assigned the SugarCRM Opportunities to the Penguin Private Investigation organization's Preliminary Investigation, and we've set up a simple sequential process. However, in many cases some jobs will have to be carried out in parallel—for example at the 'Witness and Subject Interviews' stage Charlotte Pitt has overall control, but while she undertakes witness interviews William Monk does the subject interviews. It's only once they've *both* finished that the stage is complete. This can be done by going to a Preliminary Investigation (or Opportunity) and manually adding tasks by clicking **Create Task**:



Preliminary Investigations: ZX81 disappeared

Preliminary Investigation Name	ZX81 disappeared	Amount:(GBP £)	760.00
Account Name:	Sinclair	Expected Close Date:	2007-11-17
Type:		Next Step:	
Lead Source:	Existing Customer	Investigation stage	Witness and Subject Interviews
		Probability (%):	20
Assigned to:	pittc	Last Modified:	2007-03-19 21:39 by bainm
		Date Created:	2006-11-13 23:52 by bainm
Description:			

Activities

Start Previous (1 - 2 of 2) Next End

	Close	Subject	Status	Contact	Due Date	Assigned User		
		Witness Interviews	Not Started			pittc	edit	rem
		Subject Interviews	Not Started			monkw	edit	rem

Obviously we'll need to change the logic hook so that:

- The tasks are created automatically and assigned to the correct person.
- The stage cannot be completed until all tasks have been closed.

Adding Dependent Tasks to the Database

As you've already seen, we can build the business rule functionality into a PHP file, but it is much more efficient (both from a maintenance and development perspective) to store the rules in the database. With that in mind the first thing that we need to do is to create a table in which to store the workflow tasks (i.e. the tasks that need to be run in parallel):

```
create table ppi_workflow_tasks (
    id char(36),
    user_id char(36),
    process_step varchar(50),
    title varchar(50),
    primary key (id) );
```

As always you should store the SQL in a file and then run the file against the database.

Once you've created your table then you need to think about populating it with data. Now, as you know, we can:

- Load the data from the command line
- Create a specific module for data entry

However, this time we'll add an action to an existing module that will carry out the creation of the data for us:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

$step='Needs Analysis';
$task['title'][0] = 'Subject Interviews';
$task['title'][1] = 'Witness Interviews';
$task['username'][0] = 'monkw';
$task['username'][1] = 'pittc';

for ( $c = 0; $c <= 1; $c++ )
{
  $sql = "select id from users";
  $sql .= " where user_name = '" . $task['username'][$c] . "'";
  $result = $db->query($sql);
  $user_id = mysql_result($result,0,0);

  $sql = "insert into ppi_workflow_tasks";
  $sql .= "(id, user_id, process_step, title)";
  $sql .= " values ";
  $sql .= "('". create_guid() ."'";
  $sql .= ",'". $user_id ."'";
  $sql .= ",'". $step ."'";
  $sql .= ",'". $task['title'][$c] ."')";
  $db->query($sql);
}
?>
```

If you store this file as `ppi_workflow_tasks.php` in the `ppi_workflow` module directory then you can run it through the web browser—for example if your were in the PPI organization then you would type in the URL:

```
http://hector/penguin_pi/index.php?module=ppi_workflow&action=ppi_workflow_
tasks
```

However you decide to load the data for the dependent tasks, the next stage is to modify the workflow so that they can be taken into account.

Using Dependent Tasks in the Workflow

Our simple workflow assumes that everything is done sequentially; however, our new workflow must:

- Ensure that all dependent tasks have been closed before the next stage can be initiated
- Create any required dependent tasks for each stage

So, it's back to our business rules file to make the necessary changes. The start of the file is the same—we need to:

- Define the entry point
- Include any other required files
- Define the class and the function to be called
- Declare any global variables (in this case `$db`)

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die(
                                       'Not A Valid Entry Point');
require_once('data/SugarBean.php');
require_once('modules/Opportunities/Opportunity.php');

class ppi_workflow {
  function ppi_workflow (&$bean, $event, $arguments) {
    global $db;
```

And, as before, we only want the business rule to be run if the stage has been marked as 'Completed':

```
    if ( $bean->chk_complete_c == 1 ) {
```

However, the differences start at this point, and the first thing we must do is to check if the current step is one with dependent tasks:

```
      $sql = "select * from ppi_workflow_tasks";
      $sql .= " where process_step = '" . $bean->sales_stage . "'";
      $result = $db->query($sql);
      if (mysql_numrows($result) > 0) {
```

If we find that there are dependent tasks then we must now look to see if any of those are still being worked on:

```
        $sql = "select id from tasks";
        $sql .= " where parent_id = '" . $bean->id . "'";
        $sql .= " and status <> 'Completed'";
        $status_result = $db->query($sql);
```

And if we find that any tasks are open then we need to record that fact (we'll use that to decide how the remainder of the code is to be run:

```
            if (mysql_numrows($status_result) > 0)
            {
               $tasks_outstanding = 1;
            }
          }
```

The next portion of the code is unchanged from the original, apart from the fact that it's only run if all dependent tasks have been closed:

```
          if (! $tasks_outstanding)
          {
            $sql = "select user_id, process_step from ppi_workflow";
            $sql .= " where predecessor = '" . $bean->sales_stage . "'";
            $result = $db->query($sql);

            $bean->assigned_user_id = mysql_result($result,0,0);
            $bean->sales_stage = mysql_result($result,0,1);
```

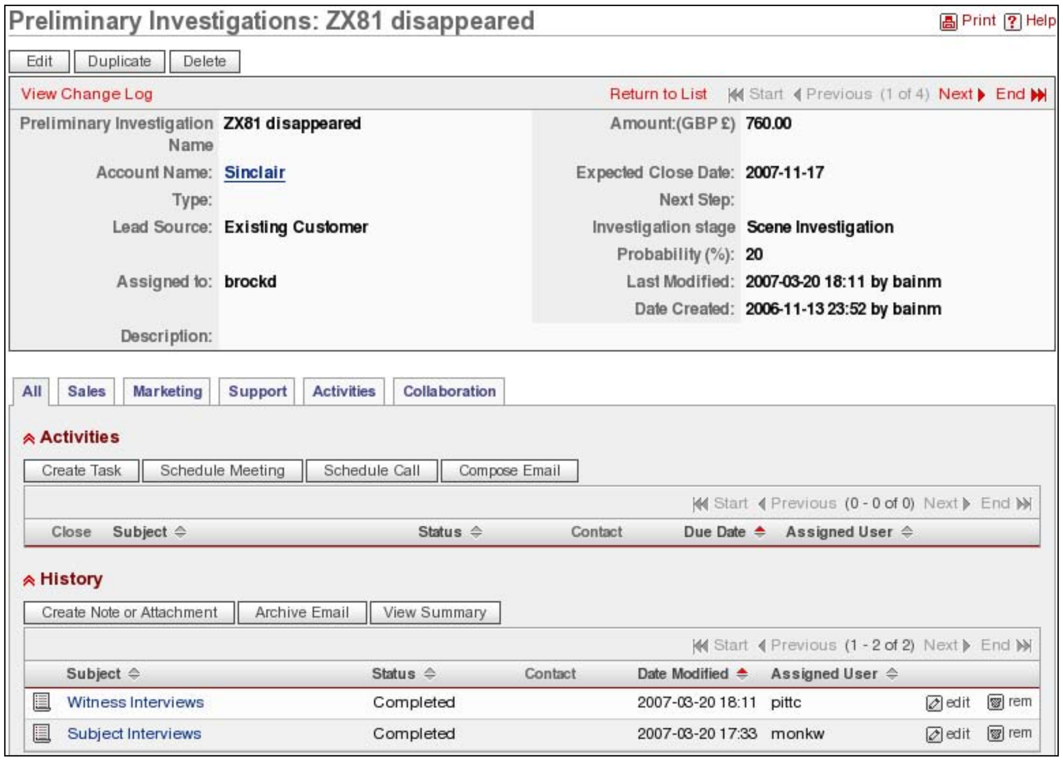
Once we've decided what the new stage is then we need to create any dependent tasks (if they're required):

```
            $sql = "select user_id,title from ppi_workflow_tasks";
            $sql .= " where process_step = '" . $bean->sales_stage . "'";
            $result = $db->query($sql);
            $r=0;
            while ($r < mysql_numrows($result))
            {
              $sql = "insert into tasks";
              $sql .= "(id,parent_id,date_entered,date_modified";
              $sql .= ",assigned_user_id,name,status,parent_type)";
              $sql .= " values ";
              $sql .= "(";
              $sql .= "'" . create_guid() . "'";
              $sql .= ",'" . $bean->id . "'";
              $sql .= ",now(),now()";
              $sql .= ",'" . mysql_result($result,$r,0) . "'";
              $sql .= ",'" . mysql_result($result,$r,1) . "'";
              $sql .= ",'Not Started'";
              $sql .= ",'Opportunities'";
              $sql .= ")";
              $db->query($sql);
              $r++;
            }
          }
```

All finally we set the new step to uncompleted (or change the original step back to uncompleted if there are outstanding tasks):

```
          $bean->chk_complete_c = 0;
        }
      }
    }
    ?>
```

With all of that done you'll find that any required tasks will be created as you move from stage to stage, and that you won't be able to move on to the next stage until all of the dependent tasks have been marked as completed. One advantage of using the tasks in this way is that you also end up with a record of who did what and when:


Preliminary Investigations: ZX81 disappeared

Preliminary Investigation Name	ZX81 disappeared	Amount:(GBP £)	760.00
Account Name:	Sinclair	Expected Close Date:	2007-11-17
Type:		Next Step:	
Lead Source:	Existing Customer	Investigation stage	Scene Investigation
		Probability (%):	20
Assigned to:	brockd	Last Modified:	2007-03-20 18:11 by bainm
		Date Created:	2006-11-13 23:52 by bainm
Description:			

History

Subject	Status	Contact	Date Modified	Assigned User
Witness Interviews	Completed		2007-03-20 18:11	pittc
Subject Interviews	Completed		2007-03-20 17:33	monkw

Obviously you'll want to take this technique further and modify it according to the particular situations that your organization deals with. For instance by adding additional fields you can have different workflows according to:

- Department
- Country
- Lead Source

Or you could just add your own custom fields. And, of course, you can start adding workflows to all of your modules.

The important thing, of course, is that you are now able to create the workflow itself, and that you can incorporate both serial and parallel tasks.

Summary

Before you start work on your workflow, ensure that: the people who understand the processes in the organization are available to you; both managers and staff agree that the process plan is correct; staff don't feel that the process is being enforced on them; and that this isn't another case of 'Big Brother'.

When you do start building the work flow ensure that you've correctly mapped your organization's stages onto the SugarCRM stages and that you have a complete listing of who does what and when.

Business rules are created by making use of SugarCRM's logic hooks. The logic hook file contains the priority of the business rule, the name of the businesses rule, the file containing the business rule, the business rule class, and the business rule function.

You can build all of your rules into a single file; however, it is better to build these into the database because you won't have to add code every time that you add a workflow or change an existing one; changes in assigned users can be handled easily.

A workflow can be serial (i.e. each task is handled one at a time and by only one person); however, it is much more likely that each stage will contain a number of tasks all carried out at once.

Your workflow can be maintained in a number of ways: you may wish to update everything directly on the database via the command line; you may wish to build a custom module to do the job; you can create individual PHP files to carry out maintenance.

If you do use parallel tasks then your code must ensure that all tasks have been completed before the next stage; the code must automatically create the tasks for each of the stages.

Remember that you can add your own fields to provide additional workflows that may depend on: region, department, time of year, and anything else that you can possibly think of.

With the workflow up and running we can now move on to the final chapter of the book. In Chapter 10 we'll be looking at various ways of improving SugarCRM even further.

10
Customizing and Optimizing SugarCRM—Tips and Tricks

Through the course of this book we've looked at some of the major issues involved in carrying out SugarCRM customizations:

- Changing the look and feel of SugarCRM
- Changing the modules that make up the application
- Understanding the SugarCRM user interface and SugarBeans
- Understanding the database structure and how SugarCRM interfaces with it
- How to carry out development in a safe and professional manner
- How to implement your own custom modules
- How to introduce your own workflow system

With the knowledge that you now have at your fingertips you'll be able to create your own set of customizations for your organization. We've seen how we could modify SugarCRM for PPI so that Korora and all of her workmates have an application that will be of direct benefit to their day-to-day tasks, and hopefully will make their lives that little bit easier.

In this final chapter we're not going to tackle any major new projects; instead we'll look at various simple ways of improving the application even more. We will, therefore, re-visit one or two of the subjects that we've already covered, and will have a look at a few additional techniques—things that may help turn this from a very useful piece of software into a truly killer application.

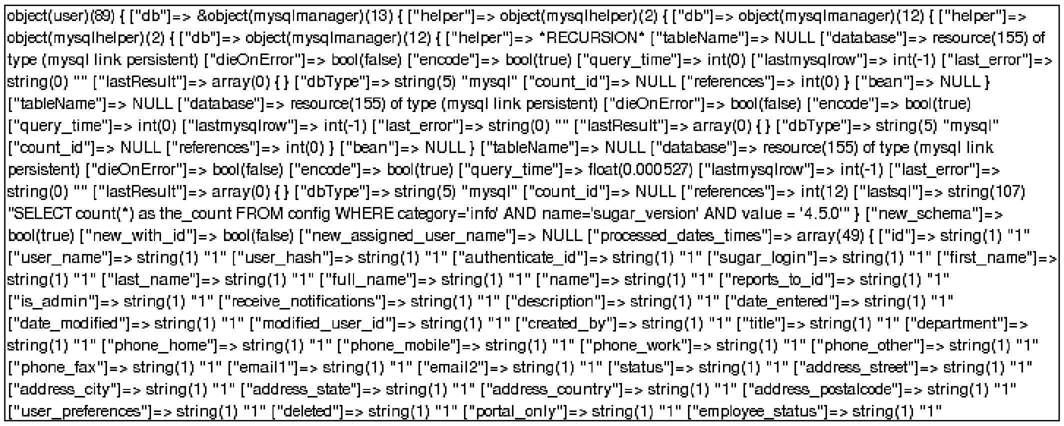
The first area that we'll look at is how to get more information out of SugarCRM.

Delving into SugarCRM Variables

While we've been customizing SugarCRM we've made use of a number of different variables—most of which are elements in SugarCRM arrays. Now, it may have occurred to you to ask the question—"How can I find out what other elements are available to me in SugarCRM arrays?" For instance let's consider one such array—`$current_user`. We can view the elements of any array that we're interested in by adding an action to one of our modules:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
var_dump($current_user);
?>
```

You'll notice that we've used the `var_dump` function here, and the end result (once you'll call it through the web browser) is:

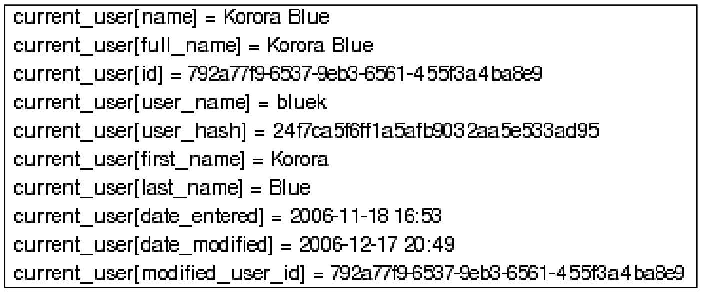


```
object(user)(89) { ["db"]=> &object(mysqlmanager)(13) { ["helper"]=> object(mysqlhelper)(2) { ["db"]=> object(mysqlmanager)(12) { ["helper"]=>
object(mysqlhelper)(2) { ["db"]=> object(mysqlmanager)(12) { ["helper"]=> *RECURSION* ["tableName"]=> NULL ["database"]=> resource(155) of
type (mysql link persistent) ["dieOnError"]=> bool(false) ["encode"]=> bool(true) ["query_time"]=> int(0) ["lastmysqlrow"]=> int(-1) ["last_error"]=>
string(0) "" ["lastResult"]=> array(0) { } ["dbType"]=> string(5) "mysql" ["count_id"]=> NULL ["references"]=> int(0) } ["bean"]=> NULL }
["tableName"]=> NULL ["database"]=> resource(155) of type (mysql link persistent) ["dieOnError"]=> bool(false) ["encode"]=> bool(true)
["query_time"]=> int(0) ["lastmysqlrow"]=> int(-1) ["last_error"]=> string(0) "" ["lastResult"]=> array(0) { } ["dbType"]=> string(5) "mysql"
["count_id"]=> NULL ["references"]=> int(0) } ["bean"]=> NULL } ["tableName"]=> NULL ["database"]=> resource(155) of type (mysql link
persistent) ["dieOnError"]=> bool(false) ["encode"]=> bool(true) ["query_time"]=> float(0.000527) ["lastmysqlrow"]=> int(-1) ["last_error"]=>
string(0) "" ["lastResult"]=> array(0) { } ["dbType"]=> string(5) "mysql" ["count_id"]=> NULL ["references"]=> int(12) ["lastsql"]=> string(107)
"SELECT count(*) as the_count FROM config WHERE category='info' AND name='sugar_version' AND value = '4.5.0'" } ["new_schema"]=>
bool(true) ["new_with_id"]=> bool(false) ["new_assigned_user_name"]=> NULL ["processed_dates_times"]=> array(49) { ["id"]=> string(1) "1"
["user_name"]=> string(1) "1" ["user_hash"]=> string(1) "1" ["authenticate_id"]=> string(1) "1" ["sugar_login"]=> string(1) "1" ["first_name"]=>
string(1) "1" ["last_name"]=> string(1) "1" ["full_name"]=> string(1) "1" ["name"]=> string(1) "1" ["reports_to_id"]=> string(1) "1"
["is_admin"]=> string(1) "1" ["receive_notifications"]=> string(1) "1" ["description"]=> string(1) "1" ["date_entered"]=> string(1) "1"
["date_modified"]=> string(1) "1" ["modified_user_id"]=> string(1) "1" ["created_by"]=> string(1) "1" ["title"]=> string(1) "1" ["department"]=>
string(1) "1" ["phone_home"]=> string(1) "1" ["phone_mobile"]=> string(1) "1" ["phone_work"]=> string(1) "1" ["phone_other"]=> string(1) "1"
["phone_fax"]=> string(1) "1" ["email1"]=> string(1) "1" ["email2"]=> string(1) "1" ["status"]=> string(1) "1" ["address_street"]=> string(1) "1"
["address_city"]=> string(1) "1" ["address_state"]=> string(1) "1" ["address_country"]=> string(1) "1" ["address_postalcode"]=> string(1) "1"
["user_preferences"]=> string(1) "1" ["deleted"]=> string(1) "1" ["portal_only"]=> string(1) "1" ["employee_status"]=> string(1) "1"
```

That's useful, but if it looks a bit too complicated (and it does) then the output can be improved by a change to the script:

```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
foreach ($current_user as $key => $value){
  echo "current_user[$key] = $value<br>";
}
?>
```

Once you refresh the browser you'll see something like:

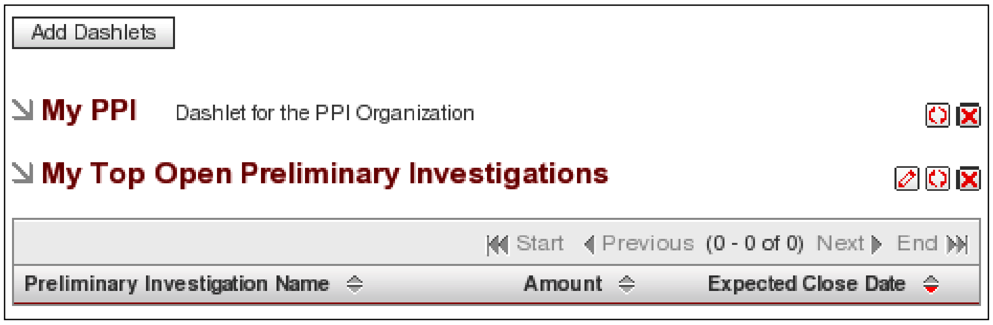


```
current_user[name] = Korora Blue
current_user[full_name] = Korora Blue
current_user[id] = 792a77f9-6537-9eb3-6561-455f3a4ba8e9
current_user[user_name] = bluek
current_user[user_hash] = 24f7ca5f6ff1a5afb9032aa5e533ad95
current_user[first_name] = Korora
current_user[last_name] = Blue
current_user[date_entered] = 2006-11-18 16:53
current_user[date_modified] = 2006-12-17 20:49
current_user[modified_user_id] = 792a77f9-6537-9eb3-6561-455f3a4ba8e9
```

Now we can start making use of all of that useful information.

Developing Dashlets Further

You'll remember that we first came across dashlets in Chapter 2—these allow us to build additional functionality into the Home page without having to create completely new modules. You'll also remember that we built a very simple dashlet:

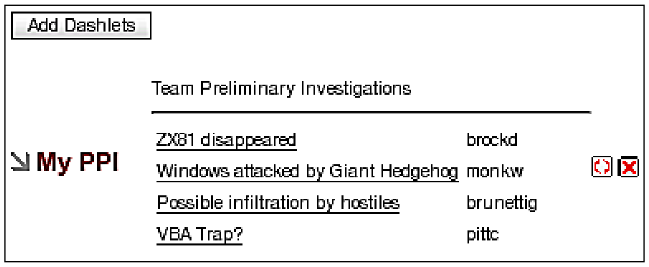

Add Dashlets
My PPI
Dashlet for the PPI Organization
My Top Open Preliminary Investigations
Start
Previous
(0 - 0 of 0)
Next
End
Preliminary Investigation Name
Amount
Expected Close Date

OK—not very exciting at the moment; however, we can quickly make it more interesting by making use of the `$current_user` array that we've just been examining. So, in this case we can use the `reports_to_id` element of `$current_user` to create a dashlet that displays all of the Preliminary Investigations for the team that the currently logged on user belongs to:

```
<?php
#As always ensure that the file can only run from SugarCRM
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
  //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 = ''){
         #Initialize the current user and the database
         global $current_user, $db;
         #Create the sql to be run
         $sql = "select o.id,o.name,u.user_name";
         $sql .= " from opportunities o, users u";
         $sql .= " where o.assigned_user_id";
         $sql .= " in (select id from users";
         $sql .=
           " where reports_to_id = '" .
                               $current_user->reports_to_id . "')";
         $sql .= " and o.assigned_user_id = u.id";
         #Obtain the results
         $result = $db->query($sql);

         Define the text to be returned
         $text = 'Team Preliminary Investigations<hr>';
         $text .= "<table>";
         $r=0;
         while ($r < mysql_numrows($result)) {
           $text .= "<tr>";
           $text .= "<td><a href=index.php?";
           $text .= "action=DetailView&module=Opportunities&record=";
           $text .= mysql_result($result,$r,0) . ">";
           $text .= mysql_result($result,$r,1);
           $text .= "</a></td>";
           $text .= "<td>" . mysql_result($result,$r,2) . "</td>";
           $text .= "</tr>";
           $r++;
         }
         $text .= "</table>";
         #Output the text
         return parent::display($text);
       }
     }
   ?>
```

Now, for example, if Maisie Dobbs logs on then she'll see:



Team Preliminary Investigations

ZX81 disappeared	brockd
Windows attacked by Giant Hedgehog	monkw
Possible infiltration by hostiles	brunettig
VBA Trap?	pittc

However, as you add extra functionality with more and more complex queries then you may find that SugarCRM starts to slow down—so next we'll look at various ways in which we can speed up the application.

Speeding up SugarCRM

Let's imagine you've finished all of your customizations, tested, and released the application to the organization. After a month or two Korora comes to your office.

'This application of yours,' she says 'it's getting really slow—what have you done to it?'

'Nothing.' you reply 'I've not touched it.'

'Well, we're finding it impossible to use—you have to sort it out'.

So, what's gone wrong? There are two areas that we can look at:

- What is there in the application that could slow it down.
- What is there in the database that could affect performance.

Since we've just added another query into our dashlet, let's look at the database first.

Optimizing Queries

When you write SQL queries for your application you may expect all queries to operate pretty well the same. However, you'll find that this isn't necessarily true. For instance let's look at two very similar queries. The first extracts data from the `users` table:

```
mysql> select user_name
    -> from users
    -> where id = '792a77f9-6537-9eb3-6561-455f3a4ba8e9';
```

```
+-----------+
| user_name |
+-----------+
| bluek     |
+-----------+
1 row in set (0.00 sec)
```

and so does the second:

```
mysql> select user_name
    -> from users
    -> where reports_to_id = '792a77f9-6537-9eb3-6561-455f3a4ba8e9';
+------------+
| user_name  |
+------------+
| varadyf    |
| monkw      |
| pittc      |
| brockd     |
| brunettig  |
| thanetl    |
| wallanderk |
| dobbsm     |
+------------+
8 rows in set (0.01 sec)
```

The only difference between the two queries is that the first uses the `id` field in its `where` clause, the second uses the `reports_to_id` field.

Having seen the two queries you might expect their performance to be identical—they both extract information from the same table in almost the same way. However, you'd be wrong. As you add more and more users into the database you'll find that the first query (using `id` in the `where` clause) carries on running quickly, while the second (using `reports_to_id` in the `where` clause) will start to run more slowly. So what's the difference?

Using the explain Command

To understand the difference between our two queries we need to make use of the SQL *explain* command:

```
mysql> explain select user_name
    -> from users
    -> where id = '792a77f9-6537-9eb3-6561-455f3a4ba8e9';
```

```
+----+-------------+-------+-------+---------------+---------+---------+-------+------+-------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref   | rows | Extra |
+----+-------------+-------+-------+---------------+---------+---------+-------+------+-------+
|  1 | SIMPLE      | users | const | PRIMARY       | PRIMARY | 108     | const |    1 |       |
+----+-------------+-------+-------+---------------+---------+---------+-------+------+-------+
1 row in set (0.00 sec)
```

If you look at the results you'll see the rows field—this contains the number of rows of data from the table that have had to be searched in order to find the required information. In this case only one row has had to be searched. However, if we use `explain` on the second query then we see a different picture:

```
mysql> explain select user_name
    -> from users
    -> where reports_to_id = '792a77f9-6537-9eb3-6561-455f3a4ba8e9';
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | users | ALL  | NULL          | NULL | NULL    | NULL |  512 | Using where |
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
1 row in set (0.00 sec)
```

The query itself only returns eight rows of data, but 512 rows have had to be searched to find them. Why 512 rows? Running another query may give us some more information:

```
mysql> select count(*)
    -> from users;
+----------+
| count(*) |
+----------+
|      512 |
+----------+
1 row in set (0.05 sec)
```

You'll realize from this that the query has had to search through the whole table in order to find what it's looking for. Obviously the more users we add then the more rows that the query must search through, and the longer that the query is going to take to run.

So why does one query search through only one of the 512 rows while the other searches through all of them? The answer is indexes.

Creating Indexes

Indexes do for a database exactly the same as they do for books. If you were looking in this book for the section on 'Creating Indexes' you wouldn't go through page by page, you'd just look for the page number in the index and then go there directly. Database indexes are just the same.

At this point it may occur to you to ask why *every* column isn't indexed—wouldn't that guarantee that every query always ran quickly? Well, that's true but indexes take up space, and so it's a trade-off between the amount of space used and the benefit of increased speed. And, of course, if you have too many indexes then this will actually slow the database down. For that reason only the most often used fields are normally indexed.

Next you would want to know how to find out if a field is indexed or not, and there are two ways to do this. The first way is to use the `explain` command that we've just been looking at. If there is an index you'll see it listed in the *key* field. If there's not one then the key field will contain 'NULL'.

You can also see if any fields are indexed by looking at the table structure on the database:

```
mysql> desc users;
+----------------------+-------------+------+-----+---------------------+-------+
| Field                | Type        | Null | Key | Default             | Extra |
+----------------------+-------------+------+-----+---------------------+-------+
| id                   | varchar(36) | NO   | PRI |                     |       |
| user_name            | varchar(60) | YES  | MUL | NULL                |       |
| user_hash            | varchar(32) | YES  |     | NULL                |       |
| authenticate_id      | varchar(100)| YES  |     | NULL                |       |
| sugar_login          | tinyint(1)  | YES  |     | 1                   |       |
| first_name           | varchar(30) | YES  |     | NULL                |       |
| last_name            | varchar(30) | YES  |     | NULL                |       |
| reports_to_id        | varchar(36) | YES  |     | NULL                |       |
| is_admin             | tinyint(1)  | YES  |     | 0                   |       |
| receive_notifications| tinyint(1)  | YES  |     | 1                   |       |
| description          | text        | YES  |     | NULL                |       |
| date_entered         | datetime    | NO   |     | 0000-00-00 00:00:00 |       |
| date_modified        | datetime    | NO   |     | 0000-00-00 00:00:00 |       |
| modified_user_id     | varchar(36) | YES  |     | NULL                |       |
| created_by           | varchar(36) | YES  |     | NULL                |       |
| title                | varchar(50) | YES  |     | NULL                |       |
| department           | varchar(50) | YES  |     | NULL                |       |
| phone_home           | varchar(50) | YES  |     | NULL                |       |
| phone_mobile         | varchar(50) | YES  |     | NULL                |       |
| phone_work           | varchar(50) | YES  |     | NULL                |       |
| phone_other          | varchar(50) | YES  |     | NULL                |       |
| phone_fax            | varchar(50) | YES  |     | NULL                |       |
| email1               | varchar(100)| YES  |     | NULL                |       |
| email2               | varchar(100)| YES  |     | NULL                |       |
| status               | varchar(25) | YES  |     | NULL                |       |
| address_street        | varchar(150) | YES  |     | NULL                |       |
| address_city          | varchar(100) | YES  |     | NULL                |       |
| address_state         | varchar(100) | YES  |     | NULL                |       |
| address_country       | varchar(25)  | YES  |     | NULL                |       |
| address_postalcode    | varchar(9)   | YES  |     | NULL                |       |
| user_preferences      | text         | YES  |     | NULL                |       |
| deleted               | tinyint(1)   | NO   |     | 0                   |       |
| portal_only           | tinyint(1)   | YES  |     | 0                   |       |
| employee_status       | varchar(25)  | YES  |     | NULL                |       |
| messenger_id          | varchar(25)  | YES  |     | NULL                |       |
| messenger_type        | varchar(25)  | YES  |     | NULL                |       |
| is_group              | tinyint(1)   | YES  |     | 0                   |       |
+-----------------------+--------------+------+-----+---------------------+-------+
37 rows in set (0.01 sec)
```

From the output you can see that two of the users table fields are indexed—`id` and `user_name`. You'll notice that `reports_to_id` is not indexed.

If you decide that the lack of an index is contributing to any degradation in performance that your users are experiencing then it's easy to create a new one:

```
mysql> create index idx_rti on users (reports_to_id);
Query OK, 512 rows affected (0.15 sec)
Records: 512  Duplicates: 0  Warnings: 0
```

And now you can use `explain` to see what affect this has had:

```
mysql> explain select user_name
    -> from users
    -> where reports_to_id = '792a77f9-6537-9eb3-6561-455f3a4ba8e9';
+----+-------------+-------+------+---------------+---------+---------+-------+------+-------------+
| id | select_type | table | type | possible_keys | key     | key_len | ref   | rows | Extra       |
+----+-------------+-------+------+---------------+---------+---------+-------+------+-------------+
|  1 | SIMPLE      | users | ref  | idx_rti       | idx_rti | 111     | const |   31 | Using where |
+----+-------------+-------+------+---------------+---------+---------+-------+------+-------------+
1 row in set (0.00 sec)
```

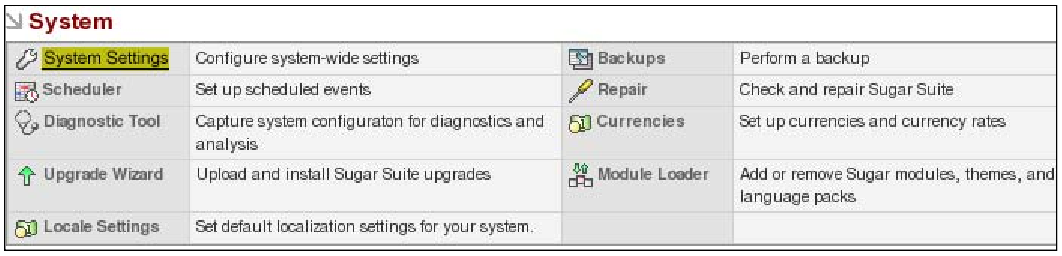
This time the query will still need to search through some of the rows of data (31 of them), but won't need to go through the full 512 records in the table.

Before we leave the optimization of queries all together we'll just look at how you can catch those queries that are causing you problems.

Logging Slow Queries

Since it's you that's done all of these SugarCRM customizations you'll already know which queries are likely to cause you problems, and hopefully you'll have done all of your optimization before releasing the application to your users. However, how do you know which queries to look at, if someone else has carried out the customizations, or if you've inherited the system? Luckily SugarCRM can come to your rescue by enabling you to log slow queries.

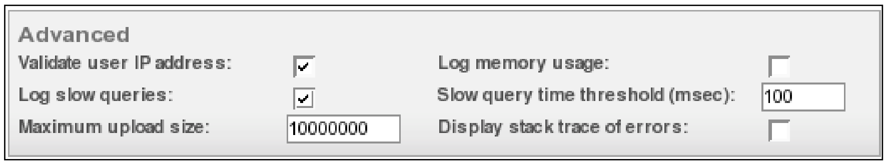
You'll need to log on as administrator and go to the **Administration** module. When there click on **System Settings**:



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 configuraton 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.		

Now scroll down to the **Advanced** section, where you'll be able to turn on slow query logging:



Advanced

Validate user IP address:	✓	Log memory usage:	
Log slow queries:	✓	Slow query time threshold (msec):	100
Maximum upload size:	10000000	Display stack trace of errors:	

You'll notice that you can also set the threshold for the query (in milliseconds)—any query that takes longer that that will be logged.

Of course, you may have a problem here—what if you want to see this in action, but haven't got any slow queries? If that's the case then set the threshold to -1—this will catch every query that you (or any of your users) run.

If you've done that then you can now see what queries are being run, and how they are performing, by viewing the SugarCRM log file (which should be located in the SugarCRM directory on your web server):

```
bluek@hector:/www/penguin_pi$ cat sugarcrm.log
Thu Mar 29 19:09:38 2007,685 [3852] FATAL SugarCRM - Slow Query
(time:0.000523
select o.id,o.name,u.user_name from opportunities o, users u where
o.assigned_user_id in (select id from users where reports_to_id =
'792a77f9-6537-9eb3-6561-455f3a4ba8e9') and o.assigned_user_id = u.id
```

So here we can see the result of the query that we added to our dashlet at the start of this chapter—it takes 0.000523 seconds to run.

And just before we finish we'll just have a quick look at the administration of the log file itself. If you find that messages are not being logged then examine the `log4php.properties` in the SugarCRM directory, and look for the setting for `log4php.appender.A2.File`. If this is set as:

```
log4php.appender.A2.File=/sugarcrm.log
```

Then change it so that it shows the full path to your file, for example:

```
log4php.appender.A2.File=/www/penguin_pi/sugarcrm.log
```

Of course, the file must exist, and it must be writable by your web service.

```
bluek@hector:/www/penguin_pi$ ls -l sugarcrm.log
-rwxrwxr-x 1 www-data www-data 147809 2007-03-30 22:37 sugarcrm.log
```

We've been looking at how to optimize individual queries, but there is a way to optimize *all* of your queries—by making use of the MySQL query cache.

Using the MySQL Query Cache

You'll no doubt realize that every time SugarCRM sends a query to the database then a new set of data will be compiled and then returned to the application. You'll also realize that the queries that SugarCRM sends are always the same—they are always the ones built into the application, either the default ones or ones that you've added through your customizations. You can make use of this knowledge by enabling the MySQL query cache.

If the MySQL query cache is enabled then instead of accessing the tables every time a query is carried out, a result is stored on the database. It's this result that's returned when another, identical, query is sent to the database.

Now—before you ask—this doesn't mean that you may retrieve out of date information from the database. To prevent this, the cache is automatically updated every time that the underlying data is changed.

So, how do we enable the MySQL query cache?

First you need to make sure that the cache exists (and it will by default):

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
1 row in set (0.02 sec)
```

Having confirmed that the cache is there, then we can turn it on by allocating a size to it (a size of 0 turns it off):

```
mysql> SET GLOBAL query_cache_size = 1000000;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+------------------+--------+
| Variable_name    | Value  |
+------------------+--------+
| query_cache_size | 999424 |
+------------------+--------+
1 row in set (0.01 sec)
```

How much of a benefit you reap will depend on the complexity of the query—the more complex the query the better the saving—but you can expect to more than halve the time it takes to run a query.

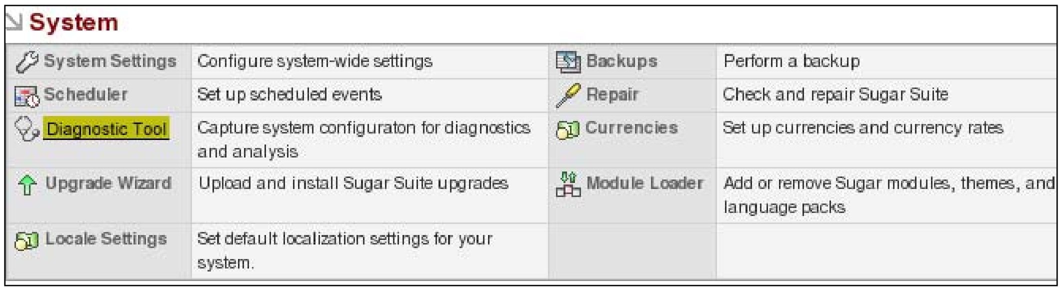
Now that you know how to optimize your queries we can turn our attention to optimizing the SugarCRM application itself

Optimizing the SugarCRM Application

You may find that simply optimizing your queries will provide all of the performance enhancements that you require. However, if not, then you'll need to start looking at the SugarCRM application itself—and a good starting point is the SugarCRM diagnostic tool.

The SugarCRM Diagnostic Tool

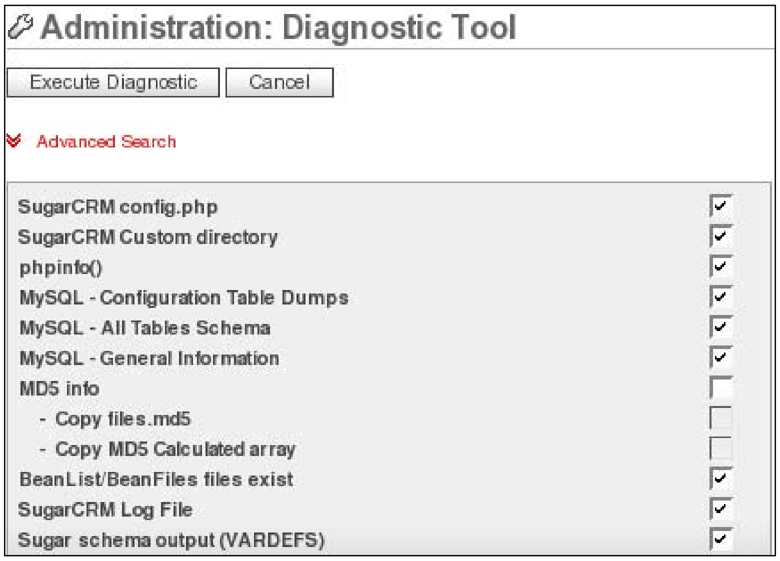
If you want to use the SugarCRM diagnostic tool, then you'll have to again log on as the administrator and go to the Admin module where you'll see the link to the tool:



System

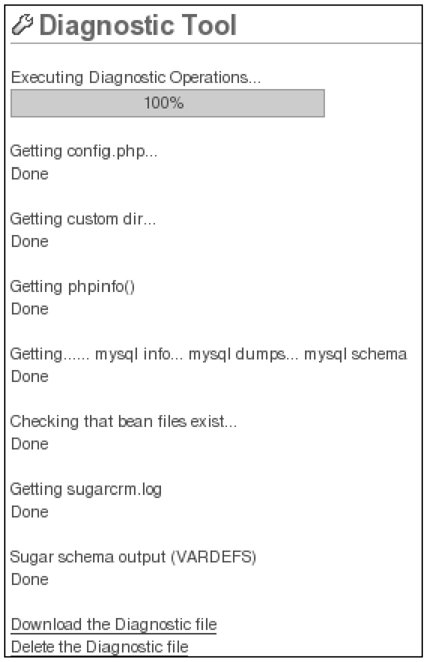
System Settings	Configure system-wide settings	Backups	Perform a backup
Scheduler	Set up scheduled events	Repair	Check and repair Sugar Suite
Diagnostic Tool	Capture system configuraton for diagnostics and analysis	Currencies	Set up currencies and currency rates
Upgrade Wizard	Upload and install Sugar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

You'll find that the diagnostic tool allows you to check various aspects of the application:



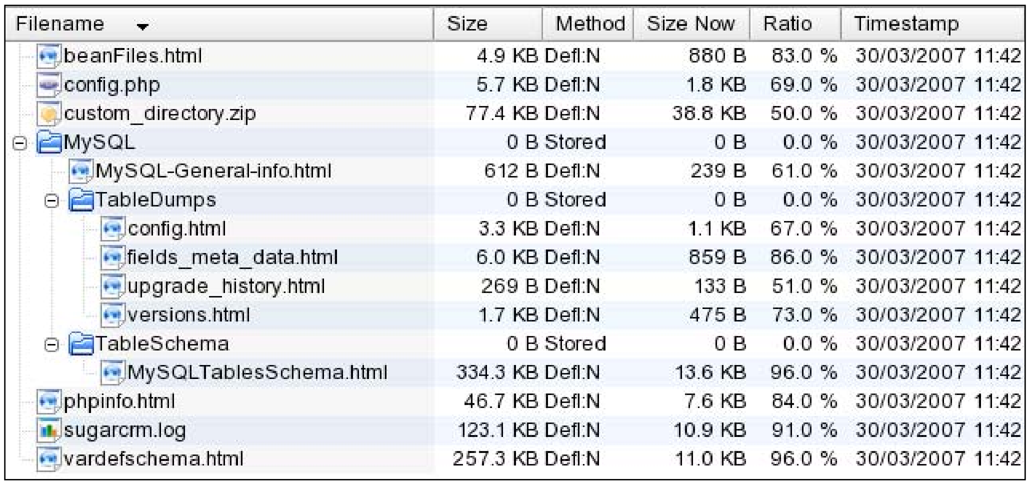
SugarCRM config.php	✓
SugarCRM Custom directory	✓
phpinfo()	✓
MySQL - Configuration Table Dumps	✓
MySQL - All Tables Schema	✓
MySQL - General Information	✓
MD5 info	
- Copy files.md5	
- Copy MD5 Calculated array	
BeanList/BeanFiles files exist	✓
SugarCRM Log File	✓
Sugar schema output (VARDEFS)	✓

Then, when you've clicked on **Execute Diagnostic Operations**, the SugarCRM will display its progress:



```
Executing Diagnostic Operations...

Getting config.php...
Done

Getting custom dir...
Done

Getting phpinfo()
Done

Getting...... mysql info... mysql dumps... mysql schema
Done

Checking that bean files exist...
Done

Getting sugarcrm.log
Done

Sugar schema output (VARDEFS)
Done
```

The diagnostic tools won't give you any results directly—you'll need to download and then open the diagnostic file (which will be a ZIP file):

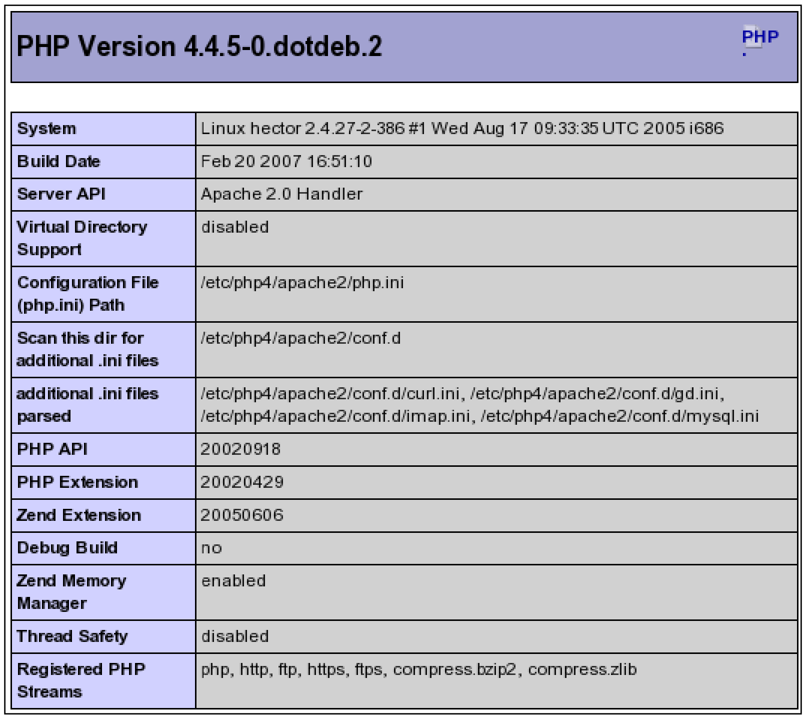


Filename	Size	Method	Size Now	Ratio	Timestamp
beanFiles.html	4.9 KB	Defl:N	880 B	83.0 %	30/03/2007 11:42
config.php	5.7 KB	Defl:N	1.8 KB	69.0 %	30/03/2007 11:42
custom_directory.zip	77.4 KB	Defl:N	38.8 KB	50.0 %	30/03/2007 11:42
MySQL	0 B	Stored	0 B	0.0 %	30/03/2007 11:42
MySQL-General-info.html	612 B	Defl:N	239 B	61.0 %	30/03/2007 11:42
TableDumps	0 B	Stored	0 B	0.0 %	30/03/2007 11:42
config.html	3.3 KB	Defl:N	1.1 KB	67.0 %	30/03/2007 11:42
fields_meta_data.html	6.0 KB	Defl:N	859 B	86.0 %	30/03/2007 11:42
upgrade_history.html	269 B	Defl:N	133 B	51.0 %	30/03/2007 11:42
versions.html	1.7 KB	Defl:N	475 B	73.0 %	30/03/2007 11:42
TableSchema	0 B	Stored	0 B	0.0 %	30/03/2007 11:42
MySQLTablesSchema.html	334.3 KB	Defl:N	13.6 KB	96.0 %	30/03/2007 11:42
phpinfo.html	46.7 KB	Defl:N	7.6 KB	84.0 %	30/03/2007 11:42
sugarcrm.log	123.1 KB	Defl:N	10.9 KB	91.0 %	30/03/2007 11:42
vardefschema.html	257.3 KB	Defl:N	11.0 KB	96.0 %	30/03/2007 11:42

Further (manual) analysis of these files may (or may not) give you further insight into what's happening in your SugarCRM application. We've already looked at one of these files—`sugarcrm.log`—but there's one other that you may find particularly useful—`phpinfo.html`.

Using phpinfo

If you open up `phpinfo.html` then you'll find that it contains something like:



PHP Version 4.4.5-0.dotdeb.2

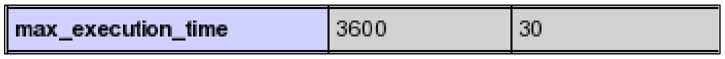
System	Linux hector 2.4.27-2-386 #1 Wed Aug 17 09:33:35 UTC 2005 i686
Build Date	Feb 20 2007 16:51:10
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php4/apache2/php.ini
Scan this dir for additional .ini files	/etc/php4/apache2/conf.d
additional .ini files parsed	/etc/php4/apache2/conf.d/curl.ini, /etc/php4/apache2/conf.d/gd.ini, /etc/php4/apache2/conf.d/imap.ini, /etc/php4/apache2/conf.d/mysql.ini
PHP API	20020918
PHP Extension	20020429
Zend Extension	20050606
Debug Build	no
Zend Memory Manager	enabled
Thread Safety	disabled
Registered PHP Streams	php, http, ftp, https, ftps, compress.bzip2, compress.zlib

However, if you scroll down the file then you'll see that it contain a *lot* of information. In particular:

- Configuration
- PHP core
- Apache Environment
- HTTP Headers
- MySQL
- PHP Variables

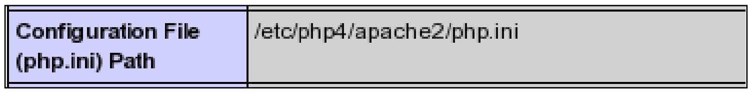
So, how can this help us? Let's imagine that you've just created a new module—it's very complex, and pulls a lot of information from the database. However, during testing Korora comes to you and tells you that sometimes it will run, sometimes it won't. This is where the techniques that we've looked at come in.

Hopefully it won't take you too long looking through the files to work out what's going on. The maximum execution time of a PHP script is set by default to 30 seconds. If your script takes longer than that then the application will time out. After a minor modification then the problem is solved and, if you're re-run the diagnostics, then you'll be able to see your new settings:



max_execution_time	3600	30

And where do you need to make these changes? The `phpinfo.html` file will tell you that as well:

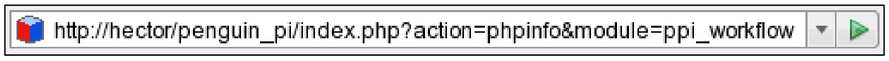


Configuration File (php.ini) Path	/etc/php4/apache2/php.ini

You'll probably find `phpinfo` very useful, and you'll be pleased to know that you don't have to run the diagnostic tool every time that you want to view the data. You can call it directly via a module action:

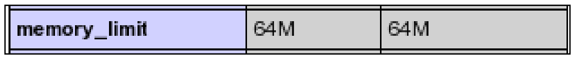
```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');
phpinfo();
?>
```

If you save this as `phpinfo.php` in the `ppi_workflow` directory then you can call it via your web browser:



http://hector/penguin_pi/index.php?action=phpinfo&module=ppi_workflow

There are many, many PHP variables that you can play with—far too many to cover in this chapter, and so we'll just finish this section by mentioning one important one. If your users start complaining about blank screens then check the `memory_limit` setting:



memory_limit	64M	64M

If you are having problems then try setting this to a higher value.

Install a PHP Optimizer

We've seen how we can make use of MySQL's query cache in order to speed up the application, and you can achieve a similar level of performance improvement by doing the same for PHP. However, there is no such cache with PHP—you must obtain a third-party tool to do the same job—a PHP optimizer. The optimizer will compile and cache the PHP code so that this doesn't have to be done every time a web page is accessed.

There are few available, but two of the most mature are:

- APC (Alternative PHP Cache)—`http://pecl.php.net/package/APC`
- eAccelerator—`http://eaccelerator.net/`

In both cases it's just a matter of downloading the software and following the installation instructions.

Right—now that we've got the database and the application running as efficiently and as quickly as possible, we'll turn our minds to another important aspect of working with SugarCRM.

Creating Reports

Everybody has to create reports—all the time—and we've already seen how to create our own reports within SugarCRM. However, as I'm sure you'll agree, they may not be the most pretty looking reports in the world. Would you really want to print one one off for your managing director? So, what's the solution? Rather than doing lots of work developing something ourselves, why don't we look at something designed to do the job? In this case we're going to look at OpenOffice.org.

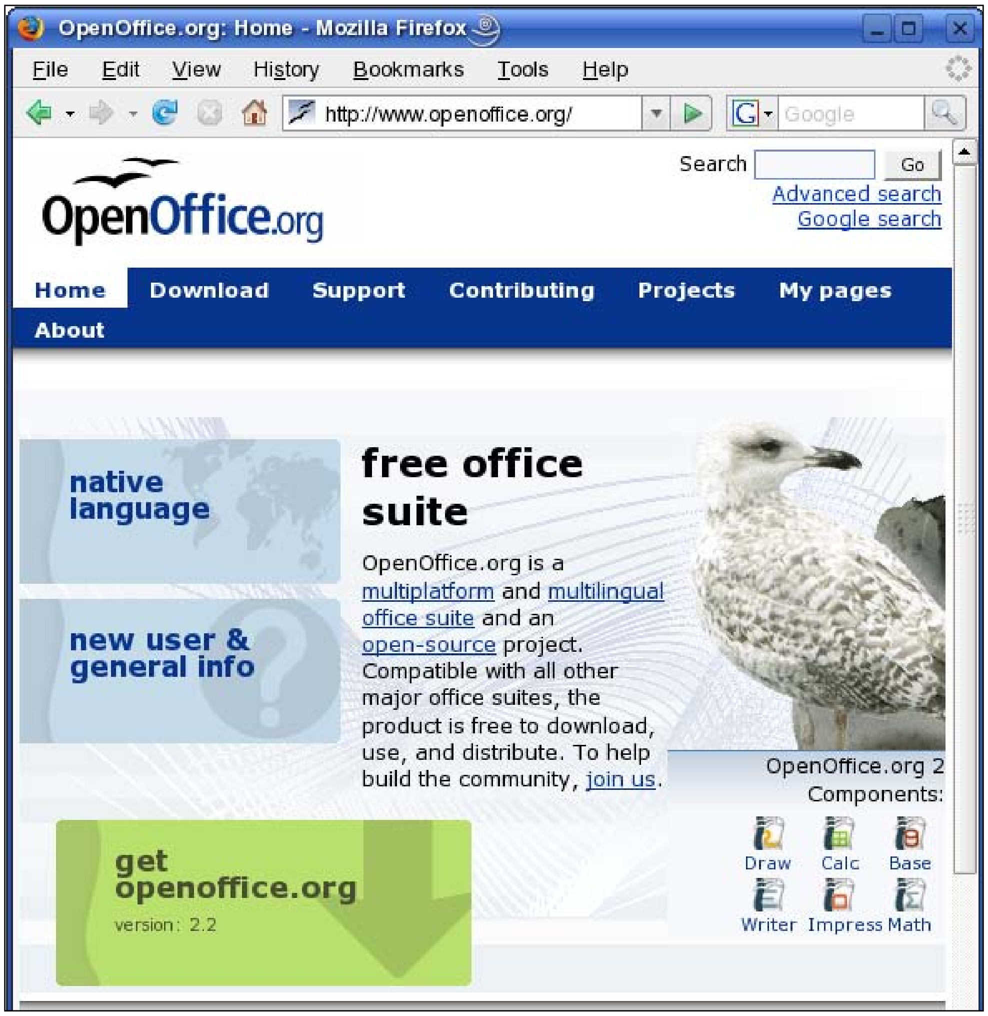
If I say that OpenOffice.org (often referred to as OOo) is like Microsoft Office then you'll immediately understand what it's all about—there's a word processor, spreadsheet, database, and many more things that are often required to make a good report. The key difference is that it's open source and, therefore, free.

We'll now take the time to have a look at how we can make use of OpenOffice.org to make reports from the information that we can obtain from the SugarCRM database.

Obtaining OpenOffice.org

Before you start installing OpenOffice.org it's just worth noting that it goes on your client PC, and *not* the server.

Installation is simple, regardless of the operating system that you're using. Just go to the OpenOffice.org website:



OpenOffice.org is a multiplatform and multilingual office suite and an open-source project. Compatible with all other major office suites, the product is free to download, use, and distribute. To help build the community, join us.

It's then just a matter of downloading the version for your particular OS and following the instructions (and remember—it goes on your own PC, not the server).

Accessing the Database Remotely

So far we've accessed the database via the SugarCRM application via the web browser. However, when we start to use OpenOffice.org we'll need to create a direct connection to the database, and to do that we have to:

- Change the settings on the database so that it can accept the connection
- Create a read-only account on the database—one specifically for our reports

Allowing Connections to the Database

You'll find that by default you can only connect to the database when you're logged on to the server—that's because MySQL has been told to only accept connections on the local IP address (127.0.0.1). All we have to do is to edit `/etc/mysql/my.cnf` and change the *bind-address* variable so it listens on the network address of the server rather than the local address:

```
bind-address            = 192.168.1.4 #The ip address of your server
#bind-address           = 127.0.0.1
#skip-networking
```

And to bring that into effect you'll need to restart the server:

```
sudo /etc/init.d/mysql restart
```

With the network connection set up you need to add a user account through which you can access the database.

Creating an Account for Remote Access

At the moment you will still not be able to connect to the database from anywhere else apart from the web server. This is because any standard user accounts only allow you to log on from the local machine. We'll need to create a new account specifically for our remote access. Therefore the first thing that you must do is to log onto the database as root:

```
bluek@hector: mysql -uroot -ppassword penguin_pi
```

Now you'll need to create an account with select only privileges:

```
mysql> grant select
    -> on *
    -> to reports@"%"
    -> identified by "reports";
Query OK, 0 rows affected (0.03 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.03 sec)
```

You'll notice that we've defined the user name as `reports@"%"`—the % means 'allow a connection from anywhere'.

Now you can log onto the database from any other PC on your network:

```
bluek@hector: mysql -hhector -ureports -preports penguin_pi
```

And you'll find that you can carry out any select queries as per normal:

```
mysql> select count(*)
    -> from opportunities
    -> where deleted = 0;
+----------+
| count(*) |
+----------+
|     1004 |
+----------+
1 row in set (0.07 sec)
```

However, if you try to update, insert, or delete then you'll just get an error message:

```
mysql> delete from opportunites
    -> where id = 'ref_999';
ERROR 1142 (42000): DELETE command denied to user 'reports'@'aeneas'
for table 'opportunites'
```

With that done you're *nearly* ready to start creating your reports.

Setting Up the ODBC Connection

One of the standard ways to connect to a database is by using an Open Database Connectivity (ODBC) connection. Basically this is a piece of third-party software that handles all the boring connecting stuff for you—if you're a Windows user then you've probably got MS ODBC, if you (sensibly) prefer Linux then you may have unixODBC. You'll also need *drivers* for MySQL—these tell ODBC how to talk to the database, and you'll be able to download the driver from the suppliers of your OS, or directly from MySQL at `http://dev.mysql.com/downloads/connector/odbc/3.51.html`.

Once you've installed your drivers then you'll need to update two files. First you'll need to find your `odbcinst.ini` file to set up the driver name and location:

```
bluek@aeneas:/etc/unixODBC> nano  odbcinst.ini
[myodbcdriver]
Description             = MySQL ODBC Driver
Driver          = /usr/lib/unixODBC/libmyodbc3.so
Setup           =
FileUsage               =
```

and then you'll need find your `odbc.ini` file and set up the data source itself:

```
bluek@aeneas:/etc/unixODBC> nano odbc.ini
[penguin_pi]
Driver       = myodbcdriver
Description  = MySQL ODBC
SERVER       = hector
PORT         =
USER         = reports
Password     = reports
Database     = penguin_pi
OPTION       = 3
SOCKET       =
```

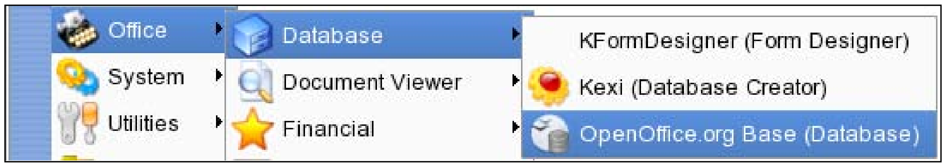
Finally, you can test the connection:

```
bluek@aeneas:~> isql penguin_pi
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
SQL>
```

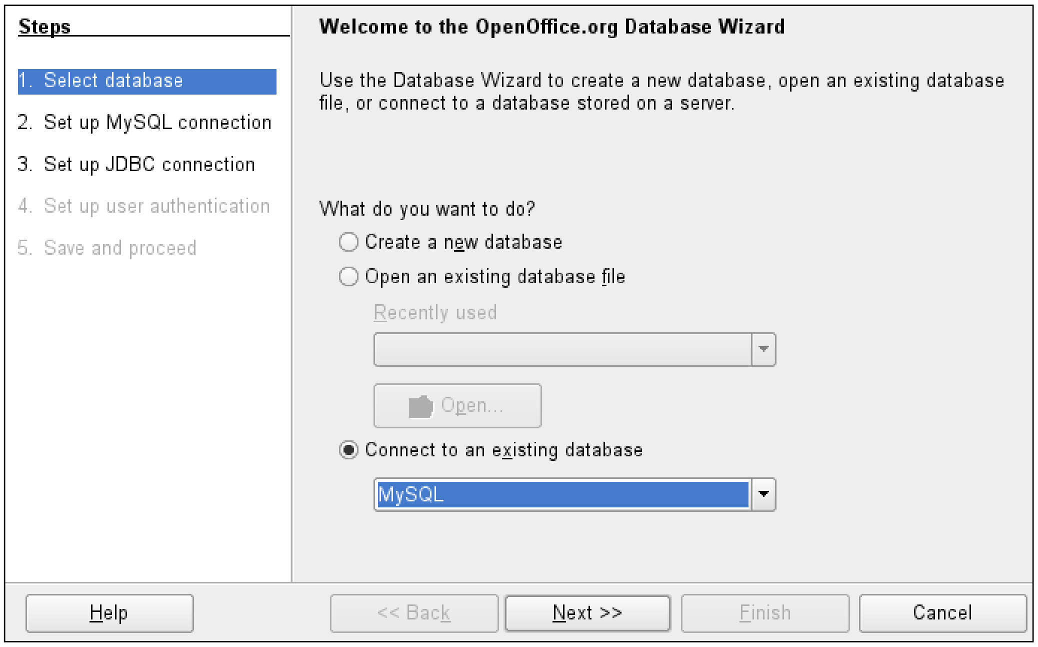
If all's well then you're ready to start using OpenOffice.org.

Accessing the Data Through OOo Base

If you've already installed OOo then it's just a matter of calling it from your desktop menu—in this case we're going to be running OOo Base, and we're going to make use of its built-in report generator:



This will start a wizard that will help you connect to the MySQL database:



Steps
1. Select database
2. Set up MySQL connection
3. Set up JDBC connection
4. Set up user authentication
5. Save and proceed

Welcome to the OpenOffice.org Database Wizard

Use the Database Wizard to create a new database, open an existing database file, or connect to a database stored on a server.

What do you want to do?

Create a new database

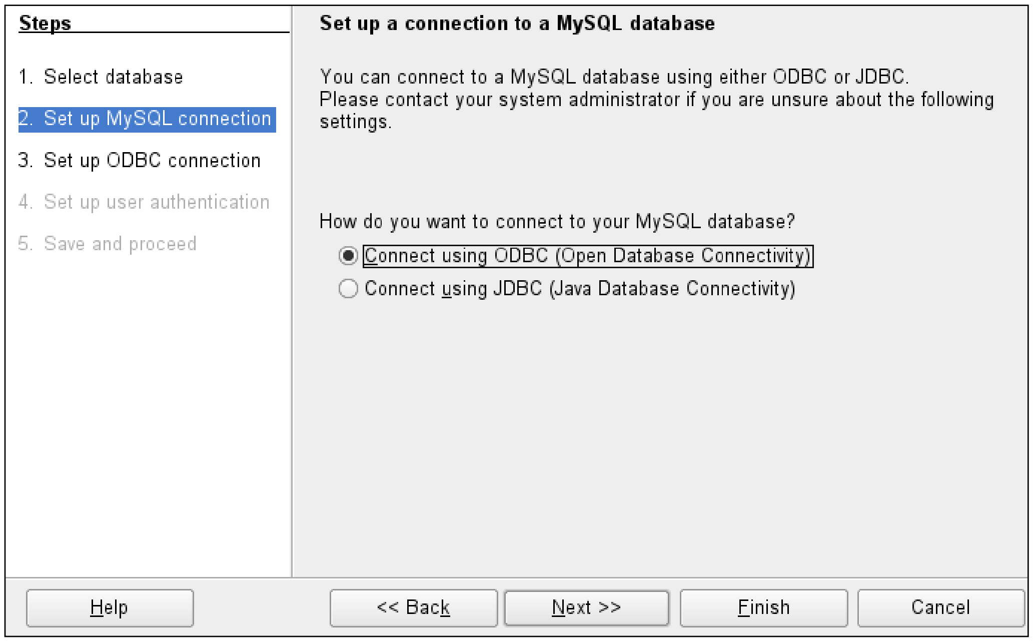
Open an existing database file

Recently used

Connect to an existing database [✓]

MySQL

Next you need to tell the wizard that we're going to be using the ODBC connection:



Steps
1. Select database
2. Set up MySQL connection
3. Set up ODBC connection
4. Set up user authentication
5. Save and proceed

Set up a connection to a MySQL database

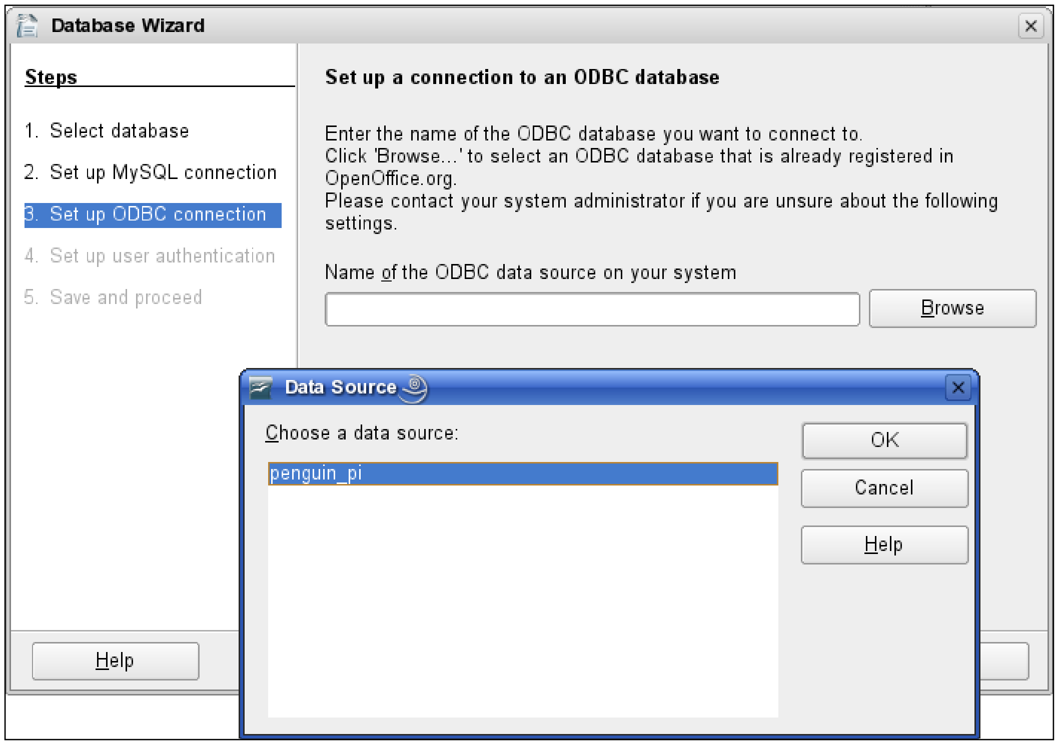
You can connect to a MySQL database using either ODBC or JDBC.
Please contact your system administrator if you are unsure about the following settings.

How do you want to connect to your MySQL database?

Connect using ODBC (Open Database Connectivity) [✓]

Connect using JDBC (Java Database Connectivity)

And then you'll supply it with the name of the data source that we've just created:



Database Wizard

Steps

1. Select database
2. Set up MySQL connection
3. Set up ODBC connection
4. Set up user authentication
5. Save and proceed

Set up a connection to an ODBC database

Enter the name of the ODBC database you want to connect to.
Click 'Browse...' to select an ODBC database that is already registered in OpenOffice.org.
Please contact your system administrator if you are unsure about the following settings.

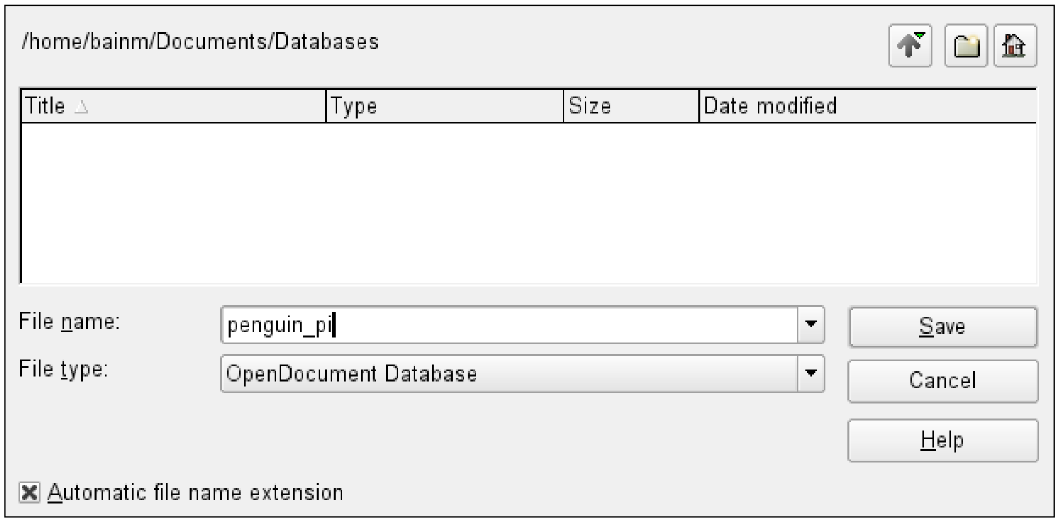
Name of the ODBC data source on your system

Data Source

Choose a data source:

penguin_pi

Finally you need to save this as an OOo Base file:



/home/bainm/Documents/Databases

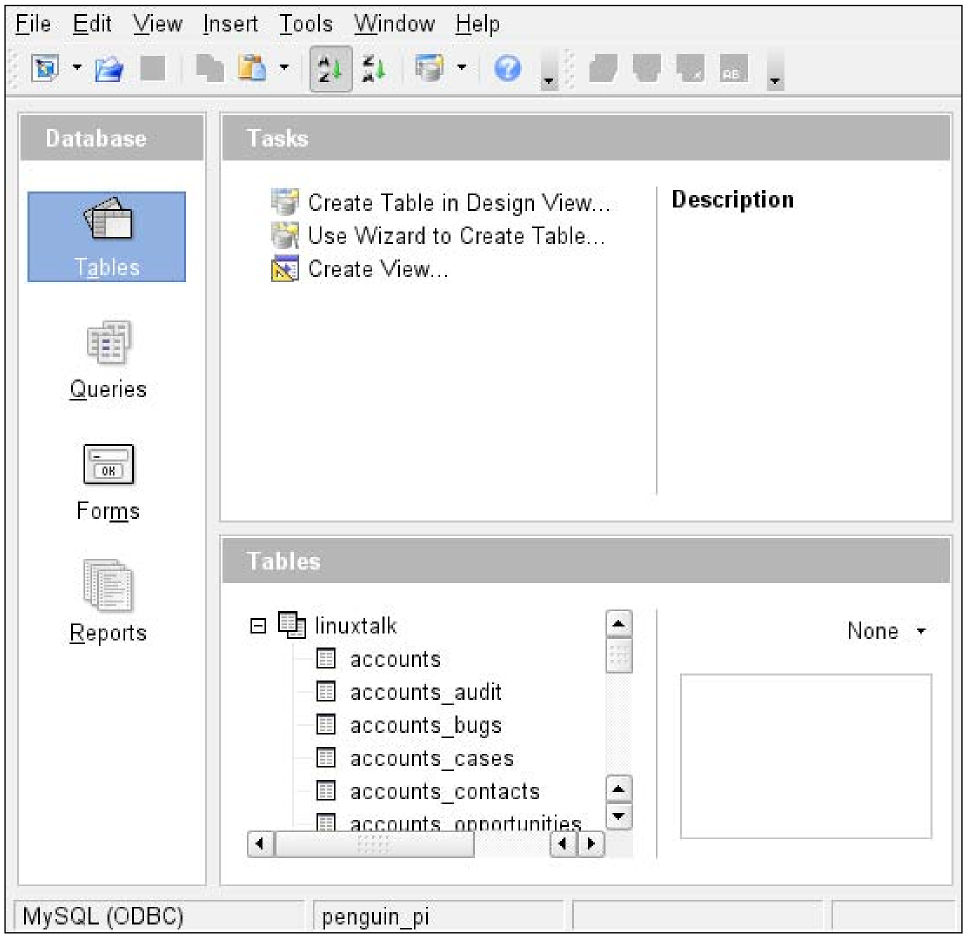
Title	Type	Size	Date modified

File name: penguin_pi

File type: OpenDocument Database

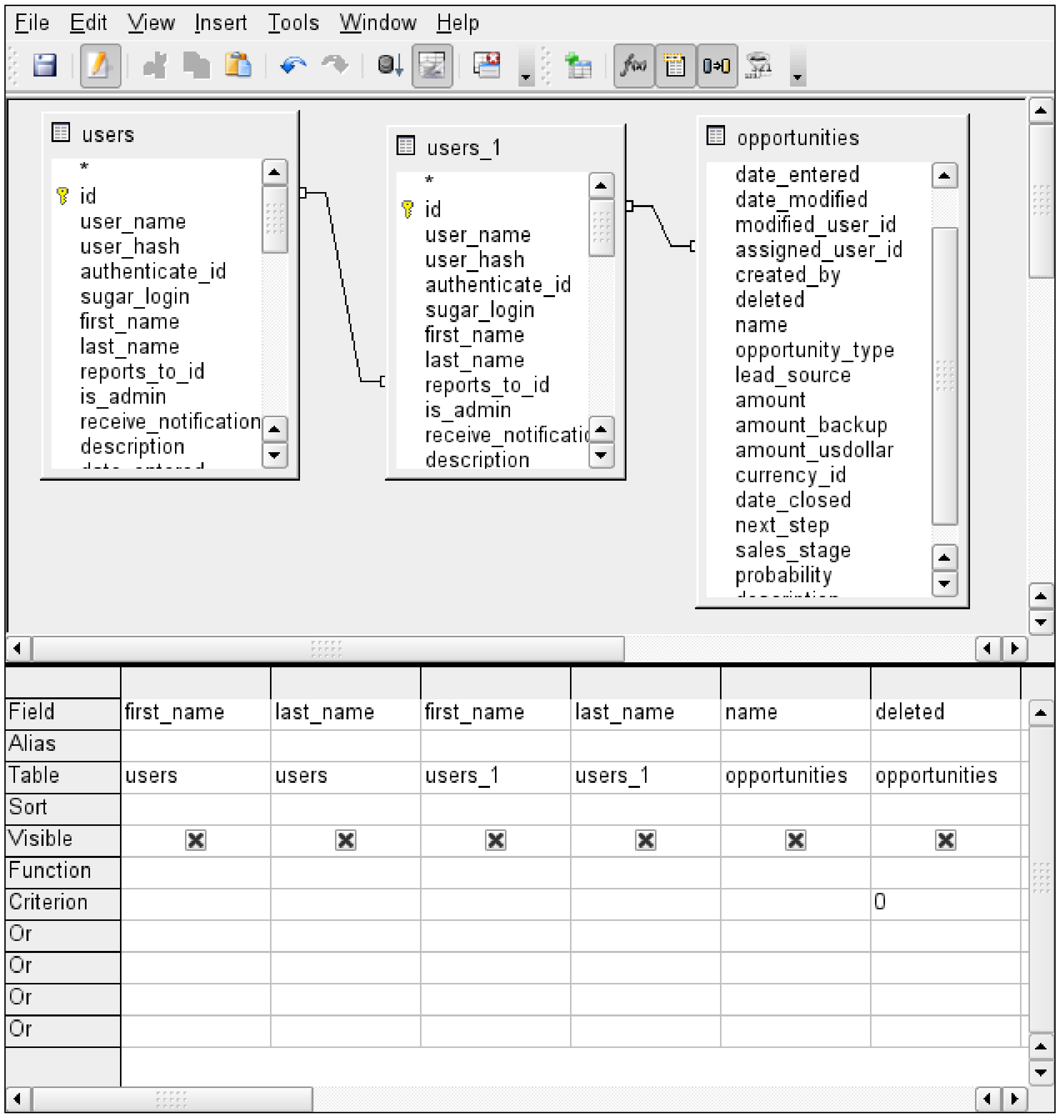
☒ Automatic file name extension

With that done we've got an OOo database that contains a connection to our SugarCRM database. In fact, if you go to the **Tables** section then you'll see the complete table listing:



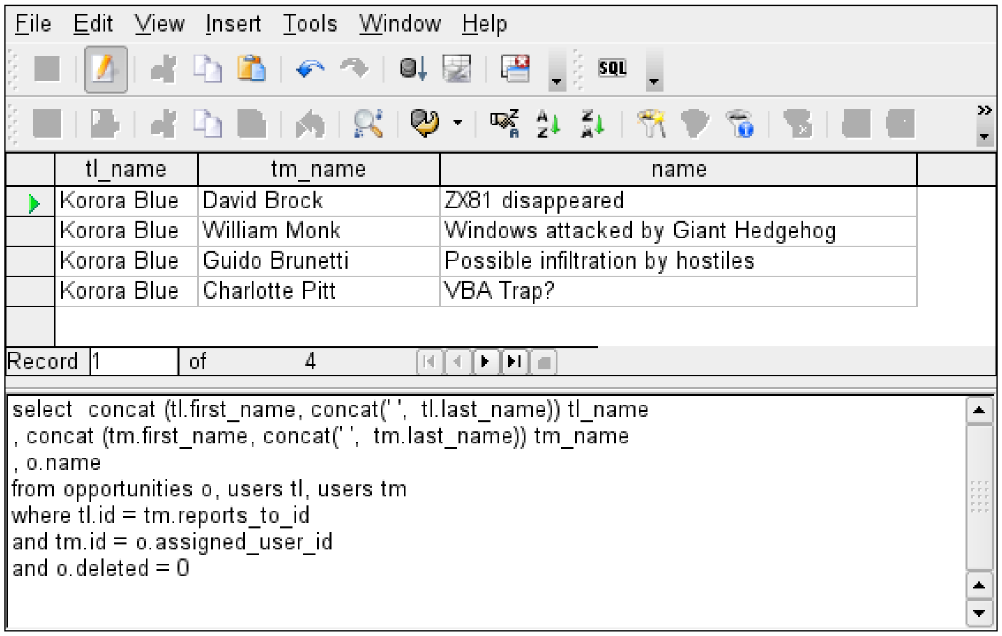
Creating Queries

Before we can generate a report you will need to create the query that is going to compile the data for the report to use. You can do this in the query design mode:



Field	first_name	last_name	first_name	last_name	name	deleted
Alias						
Table	users	users	users_1	users_1	opportunities	opportunities
Sort						
Visible	☒	☒	☒	☒	☒	☒
Function						
Criterion						0
Or						
Or						
Or						
Or						

However, you'll probably find that the SQL design mode gives you more control over what's produced:



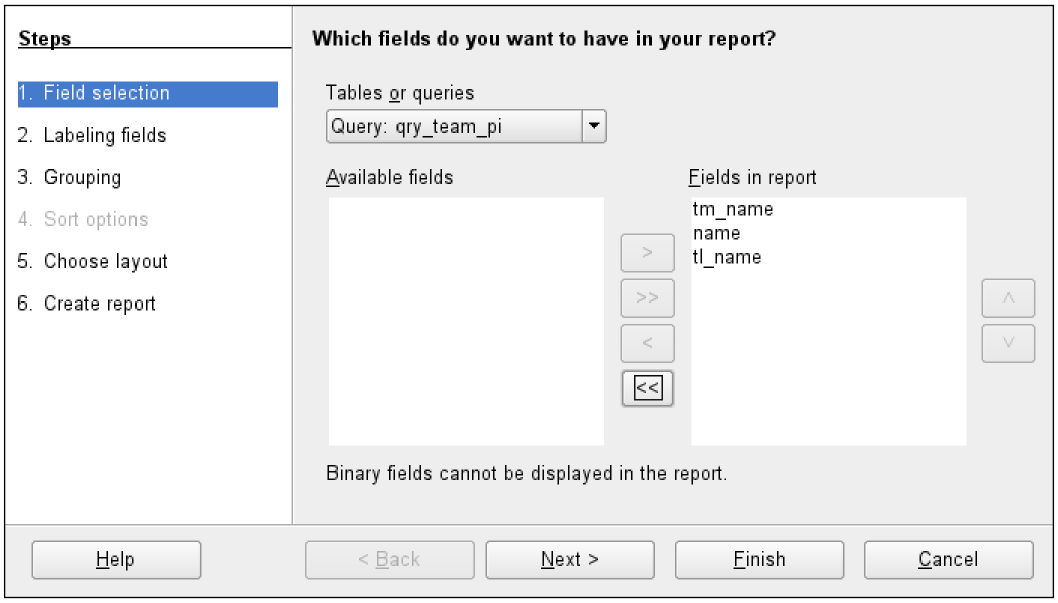
	tl_name	tm_name	name
▶	Korora Blue	David Brock	ZX81 disappeared
	Korora Blue	William Monk	Windows attacked by Giant Hedgehog
	Korora Blue	Guido Brunetti	Possible infiltration by hostiles
	Korora Blue	Charlotte Pitt	VBA Trap?

```
select  concat (tl.first_name, concat(' ',  tl.last_name)) tl_name
, concat (tm.first_name, concat(' ',  tm.last_name)) tm_name
, o.name
from opportunities o, users tl, users tm
where tl.id = tm.reports_to_id
and tm.id = o.assigned_user_id
and o.deleted = 0
```

Once you're happy with your results then save the query and give it an appropriate name.

Creating the Report

In the **Reports** section you'll find another wizard that will, rather obviously, create a new report for you. Once you've started it you'll need to tell it which query you want to use as the source for the report and which fields you want to be included:



Steps

1. Field selection
2. Labeling fields
3. Grouping
4. Sort options
5. Choose layout
6. Create report

Which fields do you want to have in your report?

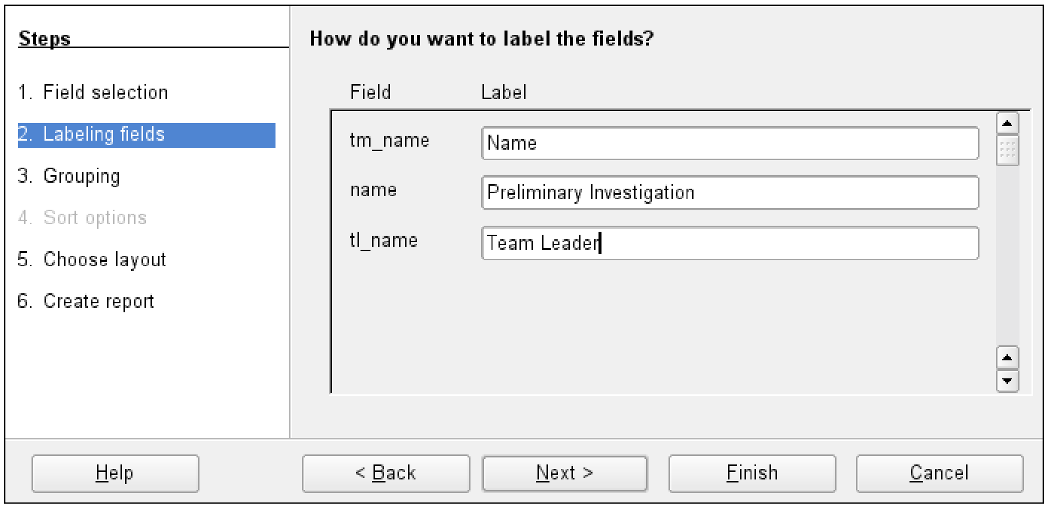
Tables or queries: Query: qry_team_pi

Available fields

Fields in report: tm_name, name, tl_name

Binary fields cannot be displayed in the report.

Then you can label each of the fields:



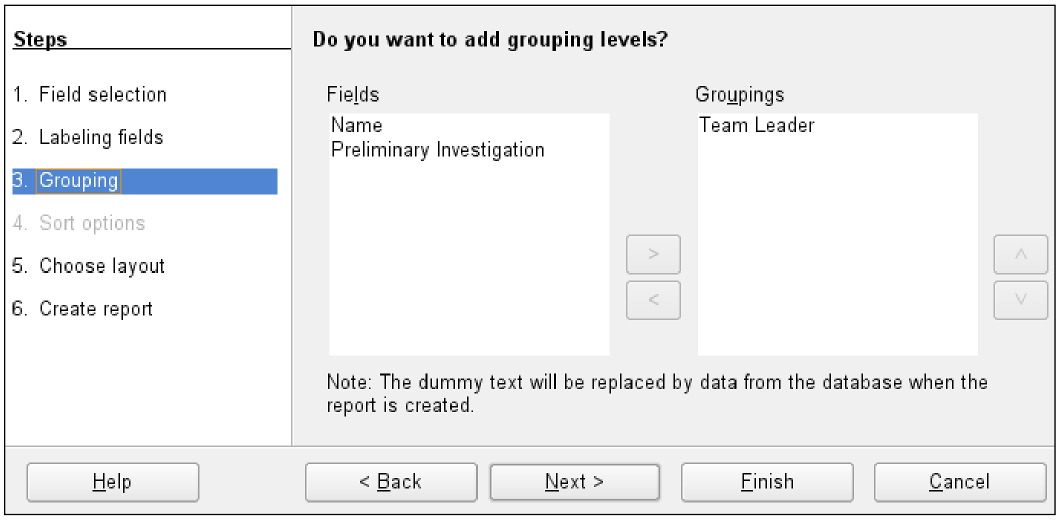
Steps

1. Field selection
2. Labeling fields
3. Grouping
4. Sort options
5. Choose layout
6. Create report

How do you want to label the fields?

Field	Label
tm_name	Name
name	Preliminary Investigation
tl_name	Team Leader

And then, quite an important step—how you want the data grouped:



Steps
1. Field selection
2. Labeling fields
3. Grouping
4. Sort options
5. Choose layout
6. Create report

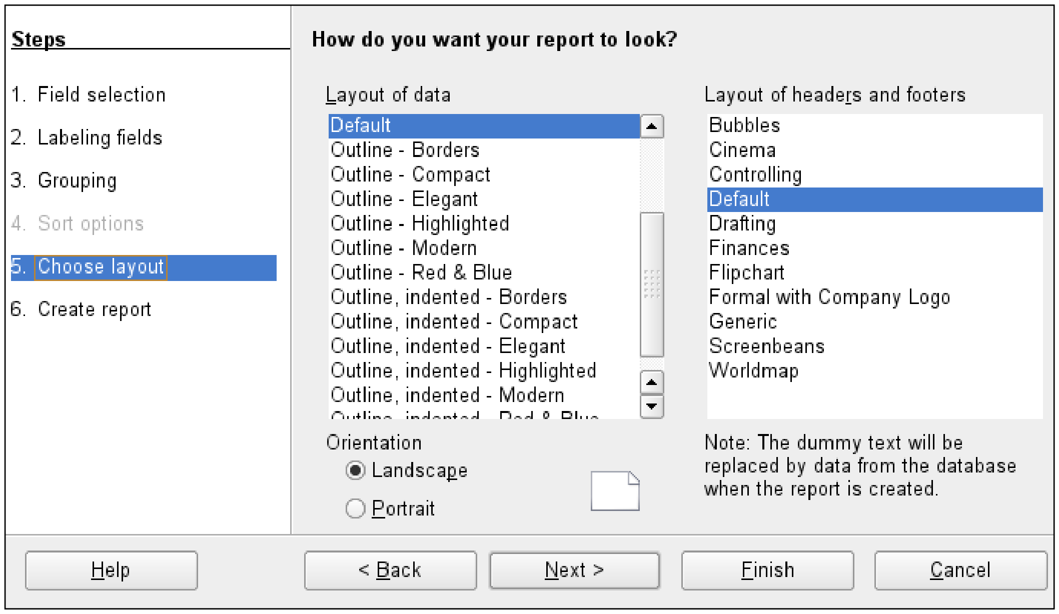
Do you want to add grouping levels?

Fields: Name, Preliminary Investigation

Groupings: Team Leader

Note: The dummy text will be replaced by data from the database when the report is created.

Next you'll have to decide on the look of the report.



Steps
1. Field selection
2. Labeling fields
3. Grouping
4. Sort options
5. Choose layout
6. Create report

How do you want your report to look?

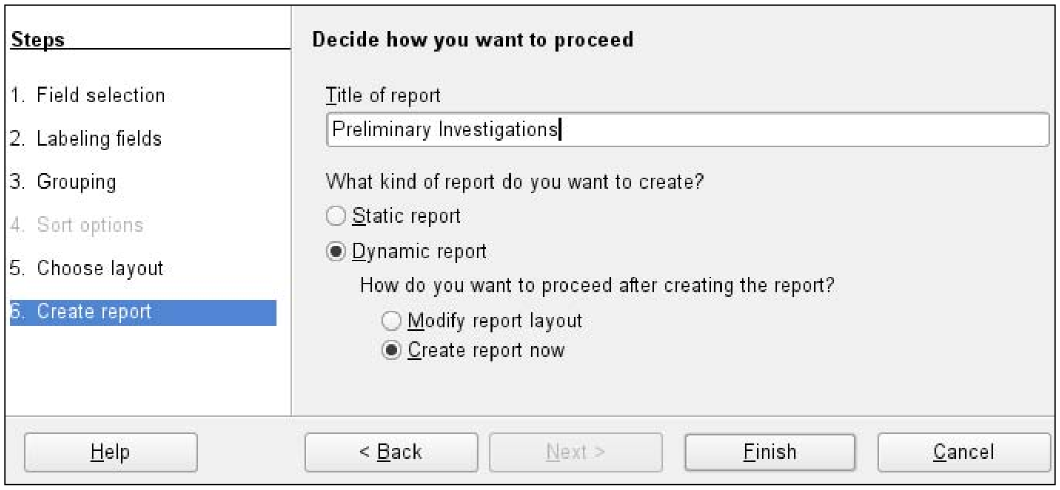
Layout of data: Default (selected), Outline - Borders, Outline - Compact, Outline - Elegant, Outline - Highlighted, Outline - Modern, Outline - Red & Blue, Outline, indented - Borders, Outline, indented - Compact, Outline, indented - Elegant, Outline, indented - Highlighted, Outline, indented - Modern, Outline, indented - Red & Blue

Layout of headers and footers: Bubbles, Cinema, Controlling, Default (selected), Drafting, Finances, Flipchart, Formal with Company Logo, Generic, Screenbeans, Worldmap

Orientation: [✓ Landscape] Portrait

Note: The dummy text will be replaced by data from the database when the report is created.

And finally, you'll input the title for the report, and also whether you want the report to be static (i.e. a snapshot of the current data) or dynamic (i.e. uses fresh data every time you run the report):



Decide how you want to proceed

Title of report: Preliminary Investigations

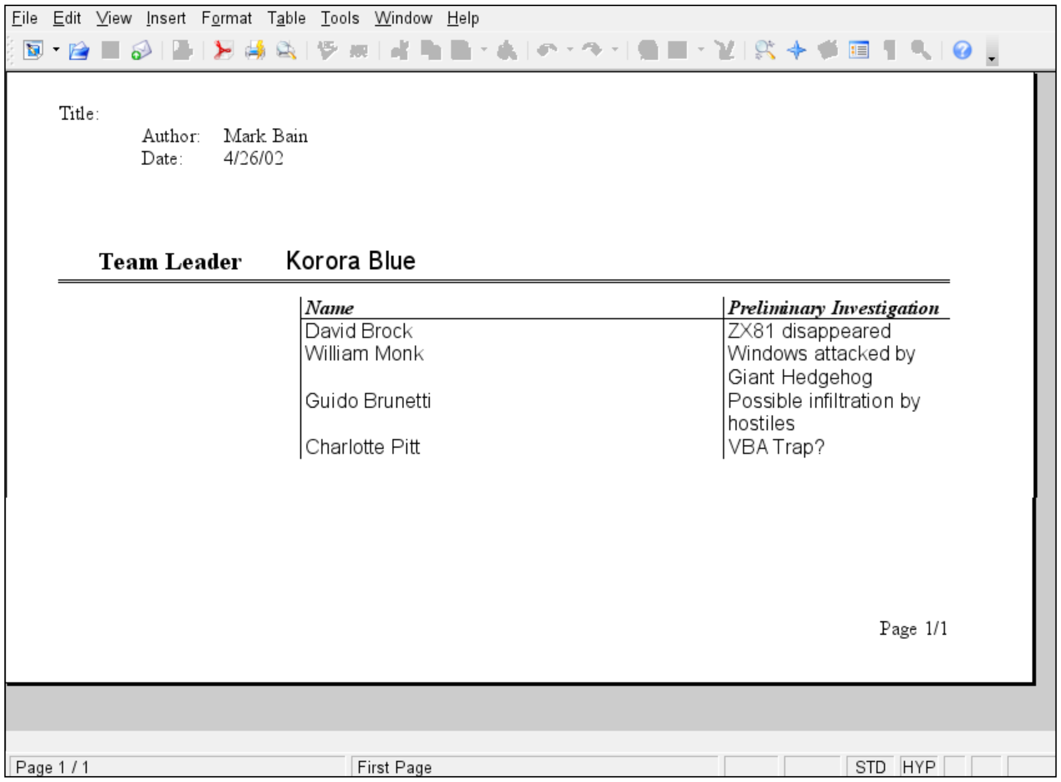
What kind of report do you want to create?

Static report [✓ Dynamic report]

How do you want to proceed after creating the report?

Modify report layout [✓ Create report now]

And now we can see the result of all your hard work:



Title:

Author: Mark Bain

Date: 4/26/02

Team Leader Korora Blue

Name	*Preliminary Investigation*
David Brock	ZX81 disappeared
William Monk	Windows attacked by Giant Hedgehog
Guido Brunetti	Possible infiltration by hostiles
Charlotte Pitt	VBA Trap?

Page 1/1

Summary

In this final chapter we've looked at various techniques for helping you to optimize the performance and your SugarCRM implementation, and a few more ways of extending the application.

And so you can now ride off into the sunset, safe in the knowledge that you left Korora and her team with a SugarCRM application that does exactly what they need. It's fast, stable, and they can even do nice reports.

Or maybe you're just straight into taming another wild SugarCRM implementation—at least now you can go in all guns blazing, knowing that you've got the fire power to handle any situation.

Index