# 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.

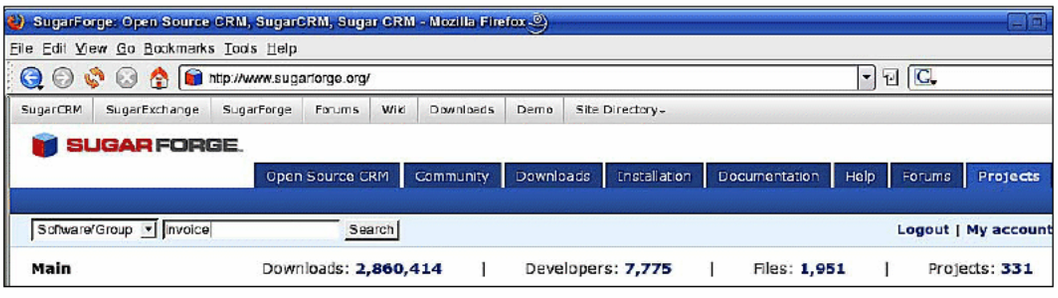
# Adding Third-party Modules

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.

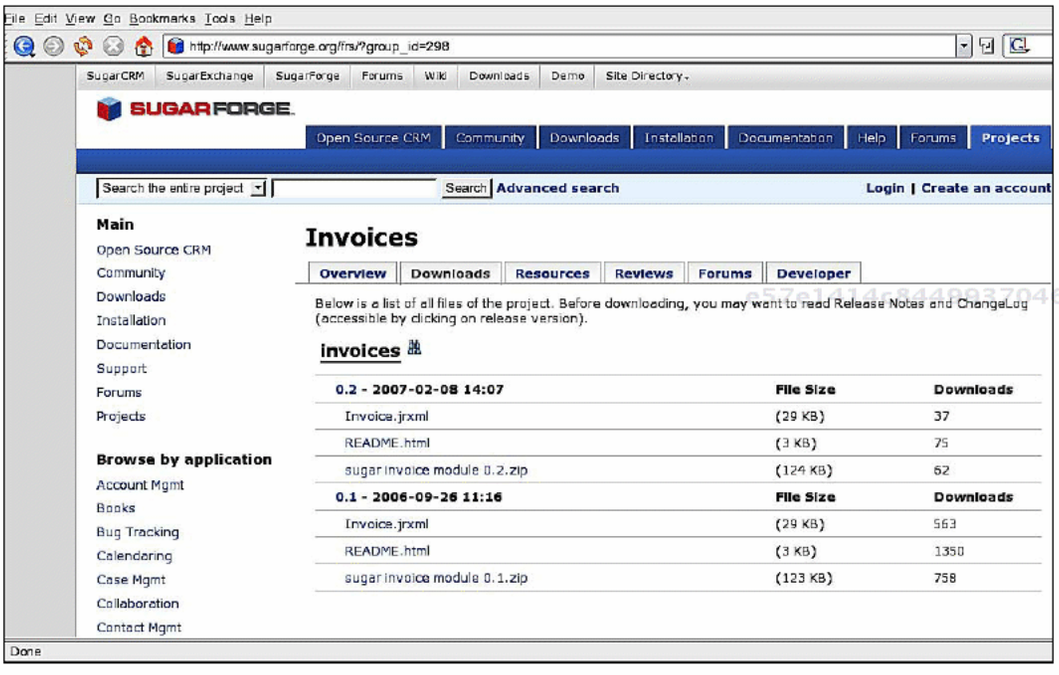
You'll be pleased to know that there are already quite a number of modules that are available to you – the number is increasing all the time, and you can download them from the Internet (of course).



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:



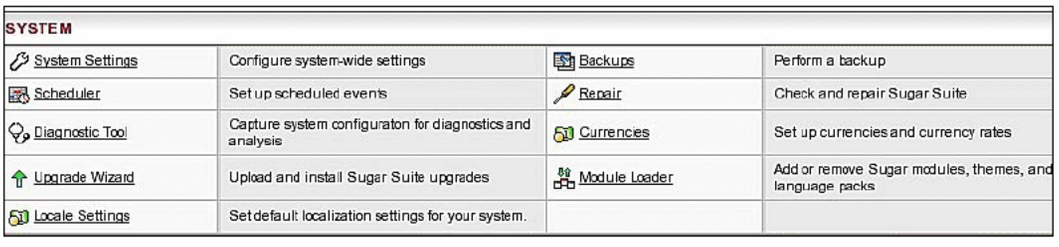
You'll find that an invoice module has already been created (by Ray Gauss II), and that you can download it:



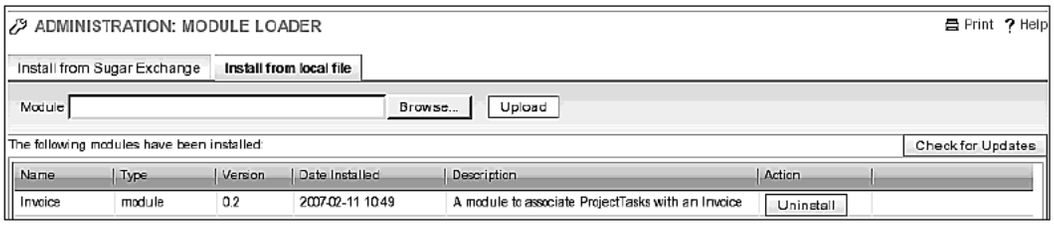

Now, you don't have to download the ZIP file to the web server—just your normal desktop will suffice. And, don't unzip it either—SugarCRM will do all of the work for you when you load the module. So, next you'll need to know how to load the module.

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:





With the module loader you can browse for the `invoice.zip` file, upload it onto the server, and install the new module:




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:

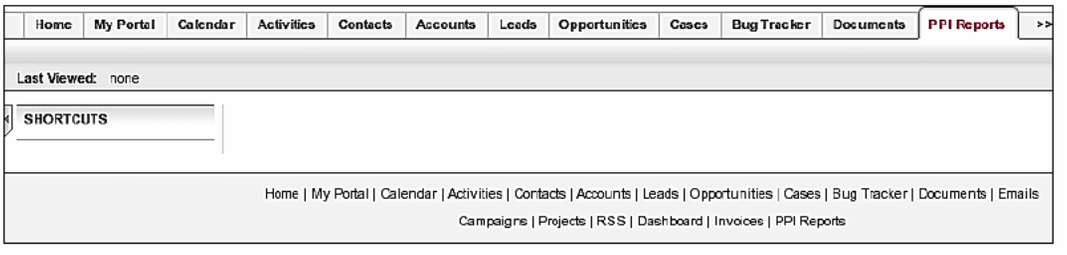
```
$moduleList[] = 'ppi_reports';
```



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:




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:




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:

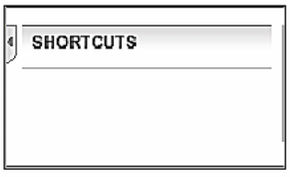
```
$report = $_REQUEST['report'];
if ($report == "monthly_new_prelim_invest")
{
  $sql = "SELECT o.name, CONCAT(u.first_name, CONCAT(' ', u.last_name))
         FROM opportunities o, users u
         WHERE MONTH(o.date_entered) = MONTH(NOW())
         AND o.assigned_user_id = u.id";

  $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:


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:

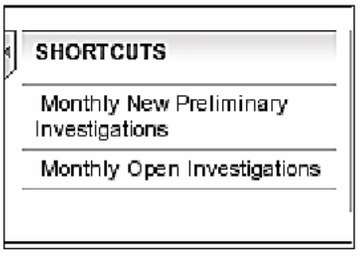



Not very impressive at the moment, but we can change this by adding a file to our module's directory. This file is `Menu.php`:



```
<?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:






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), );
?>
```

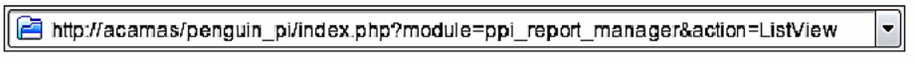
## Creating the List View

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:




Export | Selected: 0 — Start Previous (1 - 2 of 2) Next End

| 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' |

Export | Selected: 0 — Start Previous (1 - 2 of 2) Next End

Clear All

## 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:

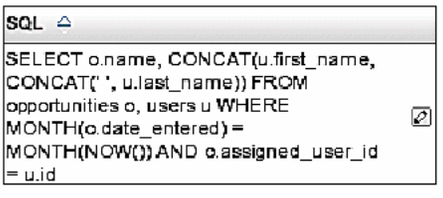
```
<?php
if(!defined('sugarEntry') || !sugarEntry) die('Not A Valid Entry
Point');

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:




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");
?>
```

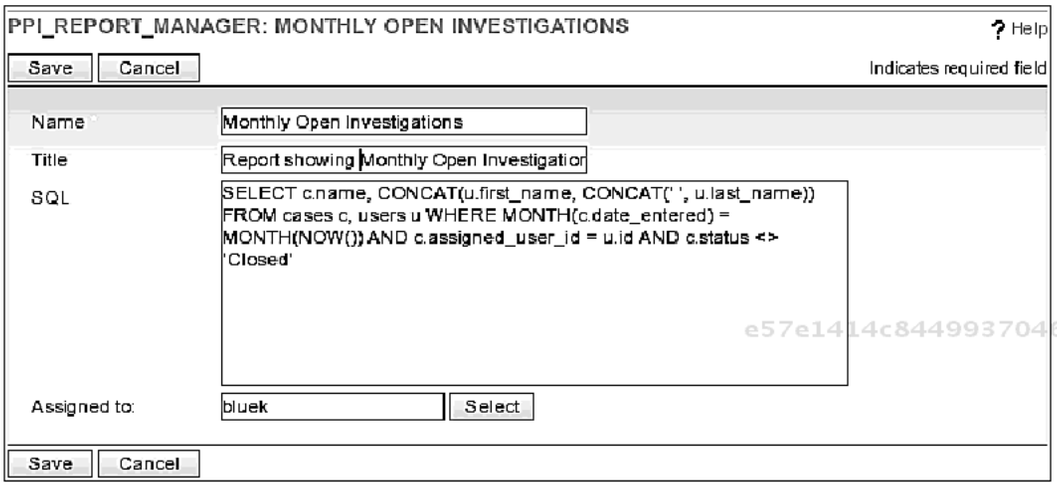
If you look through the code then you'll see that it references a file that doesn't exist yet—`modules/ppi_report_manager/EditView.html`.

## 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:




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.

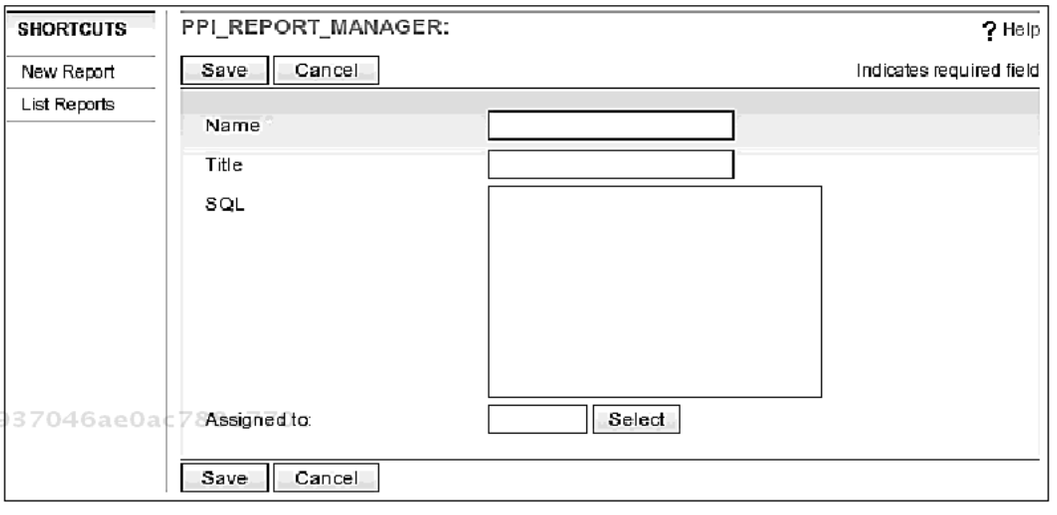
# Creating New Reports

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.