

9

Developing a Custom Workflow within SugarCRM

e57e1414c8449937046ae0ac788e770c
ebruary

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'.

e57e1414c8449937046ae0ac788e770c
ebruary

e57e1414c8449937046ae0ac788e770c
ebruary

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.

e57e1414c8449937046ae0ac788e770c
ebruary

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:

The screenshot shows the 'Mass Update' form in SugarCRM. It has buttons for 'Update' and 'Delete'. Below these are fields for 'Assigned to:', 'Type:', 'Lead Source:', 'Investigation stage', and 'Surveillance Start Date'. The 'Investigation stage' dropdown menu is open, showing a list of stages: '-None-', 'Prospecting', 'Qualification', 'Needs Analysis', 'Value Proposition', 'Id. Decision Makers', 'Perception Analysis', 'Proposal/Price Quote', 'Negotiation/Review', 'Closed Won', and 'Closed Lost'. The 'Select' button is to the right of the dropdown. At the bottom of the form, there are tabs for 'Calendar', 'Activities', 'Contacts', and 'Pr'. The 'Investigation stage' field is currently set to '-None-'.

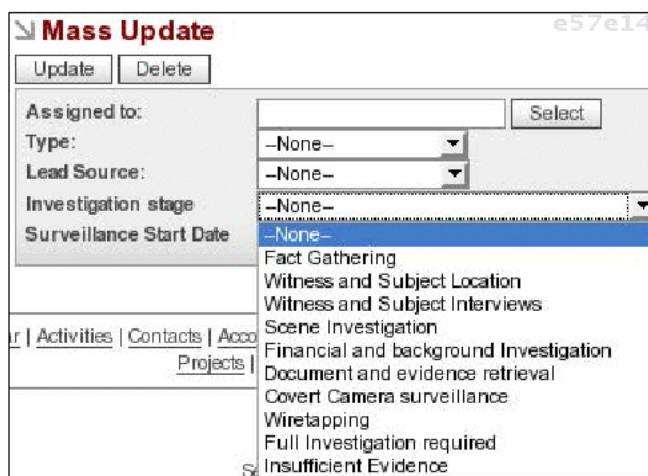
e57e1414c8449937046ae0ac788e770c
ebruary

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/laungage/en_us.lang.php` file, and refresh your browser, then you'll see the new stages:



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

Developing a Custom Workflow within SugarCRM

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 business 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 business 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');
?>
```

e57e1414c8449937046ae0ac788e770c
ebruary

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)
    {

    }
}
?>
```

e57e1414c8449937046ae0ac788e770c
ebruary

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;
}
```

e57e1414c8449937046ae0ac788e770c
ebruarye57e1414c8449937046ae0ac788e770c
ebruarye57e1414c8449937046ae0ac788e770c
ebruary

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 Help

Save Cancel * Indicates required field

Preliminary Investigation Name: Currency:

Account Name: Amount:

Type: Expected Close Date: yyyy-mm-dd

Lead Source: Next Step:

Surveillance Required?: Investigation stage:

Assigned to: Probability (%):

Description:

Surveillance Started?:

Save Cancel

However, as we saw in Chapter 3, we can use the SugarCRM Studio to add the field that we're going to need:

☒ Preliminary Investigation Name *

☒ Account Name: *

☒ Type:

☒ Lead Source:

☒ Surveillance Required?

☒ Assigned to:

☒ Description:

Data Type:

Field Name:

Field Label:

Help Text:

Default Value: ☐

Required Field: ☐

Audit ? : ☐

Duplicate Merge:

Save Cancel

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:

The screenshot shows a web form titled "Preliminary Investigations: ZX81 disappeared". The form has a "Save" button and a "Cancel" button at the top left. A red asterisk indicates required fields. The form is divided into several sections: "Preliminary Investigation Name" (text input with "ZX81 disappeared"), "Account Name" (text input with "Sinclair" and a "Select" button), "Type" (dropdown menu with "-None-"), "Lead Source" (dropdown menu with "Email"), "Surveillance Required?" (radio button with "Yes" selected), "Assigned to" (text input with "varadyf" and a "Select" button), "Description" (large text area), "Currency" (dropdown menu with "British Pounds: £"), "Amount" (text input with "760.00"), "Expected Close Date" (calendar icon with "2006-11-17" and a "yyyy-mm-dd" label), "Next Step" (text input), "Investigation stage" (dropdown menu with "Fact Gathering"), and "Probability (%)" (text input with "20"). At the bottom, there are two checkboxes: "Surveillance Started?" (radio button with "Yes" selected) and "Investigation Stage Completed" (checkbox).

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)
);
```

e57e1414c8449937046ae0ac788e770c
ebruary

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
```

e57e1414c8449937046ae0ac788e770c
ebruary

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"

Save Cancel

☐ Allow users to configure tabs

Display Tabs	Hide Tabs
<ul style="list-style-type: none">■ Home■ Web Sites■ Calendar■ Activities■ Contacts■ Accounts■ Leads■ Preliminary Investigations■ Investigations■ Documents■ Emails■ Campaigns■ Projects■ RSS■ Dashboard■ Bug Tracker■ Daily Tasks	<ul style="list-style-type: none">■ PPI Workflow

e57e1414c8449937046ae0ac788e770c
ebruary

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 ppl_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,ppl_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;
```

e57e1414c8449937046ae0ac788e770c
ebruary

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>
```

e57e1414c8449937046ae0ac788e770c
ebruary 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>
</tr>
</form>
```

e57e1414c8449937046ae0ac788e770c
ebruary

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	<input type="text" value="bluek"/>	Preliminary Investigation Stage	<input type="text" value="Fact Gathering"/>	Predecessor	<input type="text" value="None"/>
<input type="button" value="Save"/>					

e57e1414c8449937046ae0ac788e770c
ebruary

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;
        }
    }
}
?>

```

e57e1414c8449937046ae0ac788e770c
ebruary

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

EditDuplicateDelete

View Change Log

Preliminary Investigation Name: **ZX81 disappeared**

Account Name: **Sinclair**

Type:

Lead Source: **Existing Customer**

Assigned to: **pittc**

Description:

Amount:(GBP £) **760.00**

Expected Close Date: **2007-11-17**

Next Step:

Investigation stage **Witness and Subject Interviews**

Probability (%): **20**

Last Modified: **2007-03-19 21:39 by bainm**

Date Created: **2006-11-13 23:52 by bainm**

AllSalesMarketingSupportActivitiesCollaboration

Activities

Create TaskSchedule MeetingSchedule CallCompose Email

CloseSubjectStatusContactDue DateAssigned User

Witness Interviews

Not Started

pittc

Subject Interviews

Not Started

monkw

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);
}
?>
```

e57e1414c8449937046ae0ac788e770c
ebruarye57e1414c8449937046ae0ac788e770c
ebruary

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 you 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 [Print] [Help]

Edit Duplicate Delete

View Change Log Return to List Start Previous (1 of 4) Next End

Preliminary Investigation Name: **ZX81 disappeared**
Account Name: **Sinclair**
Type: **Sinclair**
Lead Source: **Existing Customer**
Assigned to: **brockd**
Description:

Amount:(GBP £) **760.00**
Expected Close Date: **2007-11-17**
Next Step:
Investigation stage: **Scene Investigation**
Probability (%): **20**
Last Modified: **2007-03-20 18:11 by bainnn**
Date Created: **2006-11-13 23:52 by bainnn**

All Sales Marketing Support Activities Collaboration

Activities
Create Task Schedule Meeting Schedule Call Compose Email

Close Subject Status Contact Due Date Assigned User

History
Create Note or Attachment Archive Email View Summary

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'.

e57e1414c8449937046ae0ac788e770c
ebruary

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.

e57e1414c8449937046ae0ac788e770c
ebruary

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.