

How To add a subpanel to a module

A Walkthru

<i>Written By:</i>	<i>Kenneth Brill, Multicom</i>
<i>Release date:</i>	<i>Dec 6, 2005</i>
<i>Revision Date:</i>	<i>Dec 18, 2005</i>

With this document I plan to take you through the steps of adding a Users subpanel to the ProjectTasks module. These steps would be the same no matter which subpanel you wanted to add to which module.

Adding a subpanel requires that you edit at least 4 files.. You either have to edit or create a new metadata file, edit the vardefs.php, layout_defs.php file and the language for the modules affected. In addition to those files you may have to create subpanel files for a module if they don't exist (like in the Employees module). After you have edited these files you may have to go to the admin panel and run the **Repair Relationships**. This rebuilds the relationship cache and if you skip this step your changes will not work right.

Step #1

The first file you need to modify is in the metadata directory. Inside the metadata directory you should see the relationship definition files.

If you look in the /metadata directory you will see that there are metadata files for all kinds of relationships. If you look there are files for relationships between Accounts & Cases, Calls & Users and Cases & Bugs. You MAY have to create your own file for you new relationship. For the ProjectTask & Users relationship I plan to show you, I use a file that already exists.

If you had to add a metafile, you need to tell SugarCRM about it. To do this you add it's name to the **modules/TableDictionary.php** file. If the file you added, for example, was named **contacts_songsMetadata.php** then you would add a line to the **modules/TableDictionary.php** file that looked like:

```
PHP Code: (modules/TableDictionary.php)

include_once("metadata/contacts_songsMetadata.php");
```

OK, now what is in a metafile? Here is a simple one:

```
PHP Code: (/metadata/meetings_usersMetaData.php)

$dictionary['meetings_users'] = array (
  'table' => 'meetings_users',
  'fields' => array (
    array('name' => 'id', 'type' => 'char', 'len' => '36', 'default' => '')
    , array('name' => 'meeting_id', 'type' => 'char', 'len' => '36',)
    , array('name' => 'user_id', 'type' => 'char', 'len' => '36',)
    , array('name' => 'required', 'type' => 'char', 'len' => '1', 'default' => '1')
    , array('name' => 'accept_status', 'type' => 'char', 'len' => '25', 'default' => 'none')
    , array('name' => 'date_modified', 'type' => 'datetime')
    , array('name' => 'deleted', 'type' => 'bool', 'len' => '1', 'default' => '0', 'required' => true)
  ),
  'indices' => array (
    array('name' => 'meetings_userspk', 'type' => 'primary', 'fields' => array('id'))
    , array('name' => 'idx_usr_mtg_mtg', 'type' => 'index', 'fields' => array('meeting_id'))
    , array('name' => 'idx_usr_mtg_usr', 'type' => 'index', 'fields' => array('user_id'))
    , array('name' => 'idx_meeting_users', 'type' => 'alternate_key',
      'fields' => array('meeting_id', 'user_id'))
  )
  , 'relationships' =>
    array ('meetings_users' => array('lhs_module' => 'Meetings',
      'lhs_table' => 'meetings',
      'lhs_key' => 'id',
      'rhs_module' => 'Users',
      'rhs_table' => 'users',
      'rhs_key' => 'id',
      'relationship_type' => 'many-to-many',
      'join_table' => 'meetings_users',
      'join_key_lhs' => 'meeting_id',
      'join_key_rhs' => 'user_id')
    )
)
```

Well, if you've never dived into one, it's pretty confusing. The only part we are going to worry about to add a subpanel relationship is the part of the file AFTER the word relationships. It's at the bottom, the last grouping.

Relationships Array

The relationships array is used to specify relationships between Beans. Like the Indices array above it, it is a list of names with array values. It is by far the hardest part of this process to understand, so take it slow.

Some of the possible attributes of a relationship entry are:

- 'lhs_module' = The module on the left hand side of the relationship
- 'lhs_table' = The table on the left hand side of the relationship
- 'lhs_key' = The primary key column of the left hand side of the relationship
- 'rhs_module' = The module on the right hand side of the relationship
- 'rhs_table' = The table on the right hand side of the relationship
- 'rhs_key' = The primary key column of the right hand side of the relationship
- 'relationship_type' = The type of relationship ('one-to-many' or 'many-to-many')

Now lets look at what our ProjectTask & Users relationship will look like. The left hand side of the relationship will be ProjectTask and the right hand side will be Users. We will use the project_relations table as it's already there and generic enough for our needs. Getting the table right will be the hardest part of making a new relationship. You may have to create a new table for your relationship and I'll go over that in appendix #1.

Here is our relationship code that we are going to add to that "Relationships" array above.

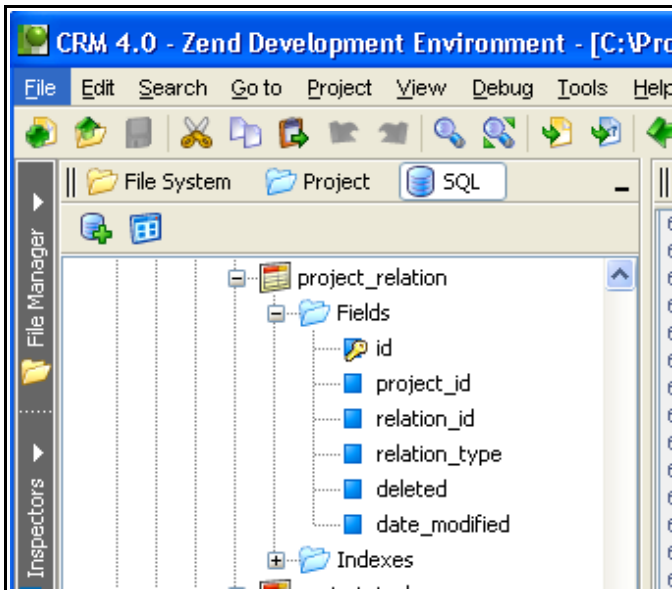
PHP Code (/metadata/project_relationMetaData.php)

```
'projects_users' => array(
    'lhs_module'=> 'ProjectTask',
    'lhs_table'=> 'project_task',
    'lhs_key' => 'id',
    'rhs_module'=> 'Users',
    'rhs_table'=> 'users',
    'rhs_key' => 'id',
    'relationship_type'=>'many-to-many',
    'join_table'=> 'project_relation',
    'join_key_lhs'=>'project_id',
    'join_key_rhs'=>'relation_id',
    'relationship_role_column'=>'relation_type',
    'relationship_role_column_value'=>'Employees'
),
```

OK, the first 8 items are pretty straight forward:

1. **lhs_module** is the left hand module name. It is the name of the module (the directory). In a 'many-to-many' relationship it doesn't matter which module goes in the left side and which one goes in the right. In a 'one-to-many' relationship it does matter. An example of a one-to-many relationship would be account_calls. Each call can only have one account. So Accounts would be the left hand module and Calls would be the right.
2. **lhs_table** is the name of the primary table for that module.
3. **lhs_key** is the column or field from that table that is used as the ID or key.
4. **rhs_module** is the right hand module name. It is the name of the module (the directory)

5. **rhs_table** is the name of the primary table for that module.
6. **rhs_key** is the column or field from that table that is used as the ID or key.
7. **relationship_type** is almost always 'many-to-many'
8. **join_table** is the third table that will hold the relationship.



For the remaining 4 we will have to look at the table shown to the right.

```
'join_key_lhs'=>'project_id',
'join_key_rhs'=>'relation_id',
'relationship_role_column'=>'relation_type',
'relationship_role_column_value'=>'Employees'
```

The first two point to where the lhs_key and the rhs_key will be saved to. In our example the 'id' from project_tasks will be saved in 'project_id' field in the project_relation table. The 'id' from users will be saved in 'relation_id' field in the project_relation table. Clear as mud? How about a little SQL to completely obscure it?

SQL Code

```
INSERT INTO join_table ( join_key_lhs, join_key_rhs, relationship_role_column)
VALUES ('lhs_key', 'rhs_key', 'relationship_role_column_value')
```

Would translate into:

```
INSERT INTO project_relation ( project_id, relation_id, relation_type)
VALUES ('14ce631f-2130-3427-0c39-438fca8d9ec5',
'6ef012f4-abeb-5579-645d-43961ecc7617',
'Employees')
```

The last two lines of the relationship are pretty easy if you line them up with the above SQL code:

```
'relationship_role_column'=>'relation_type',
'relationship_role_column_value'=>'Employees'
```

The SQL above will give you an entry that will link record '14ce631f-2130-3427-0c39-438fca8d9ec5' in project_task to record '6ef012f4-abeb-5579-645d-43961ecc7617' in users with a relation type or Employees. That is actually an example of a bad relation_type because it is not descriptive enough. It should be something like 'ProjectTask_users' mainly because this table was really intended to store relationships between Projects and other modules and I am extending it to also store relationships from ProjectTasks and I may what to assign users to both. I can't imagine why but I might. The relation_type field allows you to store several different relationships in a single table.

For example in my project_relation table I store these relationships:

lhs	rhs
Projects	Accounts
Projects	Contacts
Projects	Opportunities
Projects	Cases *
ProjectTasks	Users *
Projects	Documents *

*I added these

I think that should get you through metafiles. If you followed closely and know a little SQL it should have made sense to you. I hope it did at least. If you made it this far the rest is pretty easy.

Step #2

The vardefs.php file is the next file we are going to tackle. Here is the vardefs.php entry that would be placed in the 'fields' array (which is part of the dictionary array) we are going to make for our ProjectTask => Users relationship.

PHP Code: (from ProjectTask/vardefs.php)

```
1      'users' => array (
2          'name' => 'users',
3          'type' => 'link',
4          'relationship' => 'projects_users',
5          'module'=>'Users',
6          'source'=>'non-db',
7          'ignore_role'=>true,
8          'vname'=>'LBL_USERS',
          ),
```

It doesn't matter in what position you put it in the array. It can be first, last or in the middle. I usually put it at the end of the 'fields' array just before the 'indices' array.

1. The name of the array and the field 'name' (#2) will almost always be the same. It can be whatever you want but should be kept simple.
2. The name of your field
3. The type of link for what we are doing should always be 'link'
4. This is where you refer back to the relationship that you created in the metadata.
5. This is the name of the rhs module from your relationship.
6. This should always be 'non-db', I don't know why it would be called 'non-db'
7. I don't know exactly what ignore_role does. I would assume that it would allow the subpanel to work even if your role doesn't have access normally.
8. This is a tag from the language file. I can't find anywhere that this one is used in the code but define it anyway. We will go over the language file later.

Step #3

The layout_defs.php file is the file that actually puts the subpanel on the screen. The 'subpanel_setup' array contains all of the tabs data. This is the code we are going to add here:

```
PHP Code: (ProjectTask/layout_defs.php)

'users' => array(
    'top_buttons' => array(
        array(
            'widget_class' => 'SubPanelTopSelectButton',
            'popup_module' => 'Users'),
        ),
    'order' => 80,
    'module' => 'Users',
    'subpanel_name' => 'default',
    'get_subpanel_data' => 'users',
    'add_subpanel_data' => 'users_id',
    'title_key' => 'LBL_USERS_SUBPANEL_TITLE',
),
```

This is the least documented part of the process. I don't know what a lot of this does, but let's try and figure it out.

The top part renders the buttons at the top of the subpanel. A subpanel without any buttons would be read-only as there would be no way to 'Select' new items to add. There isn't a widget_class for a 'Create New User' button. But one could be written. The widgets are in the *include/generic/SugarWidgets* directory. If you want more than one button you need the top_buttons array to be an array of arrays like this:

```
PHP Code:

'top_buttons' => array(
    array('widget_class' => 'SubPanelTopCreateNoteButton'),
    array('widget_class' => 'SubPanelTopArchiveEmailButton'),
    array('widget_class' => 'SubPanelTopSummaryButton'),
),
```

SubPanelTopCreateNoteButton, SubPanelTopArchiveEmailButton, SubPanelTopSummaryButton are simply references to files in *include/generic/SugarWidgets*.

Then the 'order' field is how you affect in what order the subpanels will be drawn. This one is set to '80' so in this file it would be drawn in 8th place as the others are numbered in tens (10,20,30,40 etc). I don't believe it matters how they are numbers, the program will simply render them in order from the lowest number to the highest number.

The next 4 lines after 'order' point to the Module that the subpanel is from. In this case Users.

Tag	Description
'module' => 'Users',	This is the module that the subpanel will be pulled from
'subpanel_name' => 'default',	This is the actual name of the subpanel php file. For example this setting would get a file called 'default.php' out of the subpanel directory in the Users module directory (modules/Users/subpanels). If it said ' subpanel_name ' => ' appointments ', then the program would get a file called appointments.php out of the subpanel directory in the Users module directory.
'get_subpanel_data' => 'users',	Not Sure
'add_subpanel_data' => 'users_id',	Not Sure
'title_key' => 'LBL_USERS_SUBPANEL_TITLE',	the tag from the language file that will rendered as the title of the subpanel.

Step #4

This is the easiest step of all. The Language file. You need to drop the two keys into the modules language file. The language files are contained in the language directory in the module. So for Users the directory path would be modules/Users/language. For US English file you are looking for is en_us.lang.php. You should insert the two new lines at the top like this.

PHP Code:

```
$mod_strings = array (  
    'LBL_USERS_SUBPANEL_TITLE' => 'Employees',  
    'LBL_USERS' => 'Employees',  
    'LBL_MODULE_NAME' => 'Project Tasks',  
    'LBL_MODULE_TITLE' => 'Project Task: Home',  
    'LBL_SEARCH_FORM_TITLE' => 'Project Task Search',  
    'LBL_LIST_FORM_TITLE' => 'Project Task List',
```

and thats it. Run the **Repair Relationships** command from the admin menu and marvel at your new subpanels.

Appendix 1: How to create a new relationship table.

Remember this file:

PHP Code:

```
$dictionary['meetings_users'] = array (
  'table' => 'meetings_users',
  'fields' => array (
    array('name' => 'id', 'type' => 'char', 'len' => '36', 'default' => '')
    , array('name' => 'meeting_id', 'type' => 'char', 'len' => '36',)
    , array('name' => 'user_id', 'type' => 'char', 'len' => '36', )
    , array('name' => 'required', 'type' => 'char', 'len' => '1', 'default' => '1')
    , array('name' => 'accept_status', 'type' => 'char', 'len' => '25', 'default' => 'none')
    , array('name' => 'date_modified', 'type' => 'datetime')
    , array('name' => 'deleted', 'type' => 'bool', 'len' => '1', 'default' => '0', 'required' => true)
  )
  ,
  'indices' => array (
    array('name' => 'meetings_userspk', 'type' => 'primary', 'fields' => array('id'))
    , array('name' => 'idx_usr_mtg_mtg', 'type' => 'index', 'fields' => array('meeting_id'))
    , array('name' => 'idx_usr_mtg_usr', 'type' => 'index', 'fields' => array('user_id'))
    , array('name' => 'idx_meeting_users', 'type' => 'alternate_key',
      'fields' => array('meeting_id', 'user_id'))
  )
  , 'relationships' =>
    array ('meetings_users' => array('lhs_module' => 'Meetings',
      'lhs_table' => 'meetings',
      'lhs_key' => 'id',
      'rhs_module' => 'Users',
      'rhs_table' => 'users',
      'rhs_key' => 'id',
      'relationship_type' => 'many-to-many',
      'join_table' => 'meetings_users',
      'join_key_lhs' => 'meeting_id',
      'join_key_rhs' => 'user_id')
    )
)
```

It is a metadata file. The code in blue (I believe) is the table generation code. If you created a new metadata file with a new table definition, added the metadata file name to the tableDictionary.php file and then went to the admin panel and run the [Repair Databases](#) command the program would create your new table. Some knowledge of SQL would be needed to create working code and this document isn't the place to teach you SQL.