Elastix
Development manual

ADDONS.ELASTIX.ORG

Elastix Development

Guidance Manual

VERSIONS

Version	Elastix Version	Date	Created by	Details
1	2.3.0	01/16/12	Alberto Santos	Initial documentation

Introduction

The present manual will serve as a guide for developers that wish to use the Elastix Framework to create new modules.

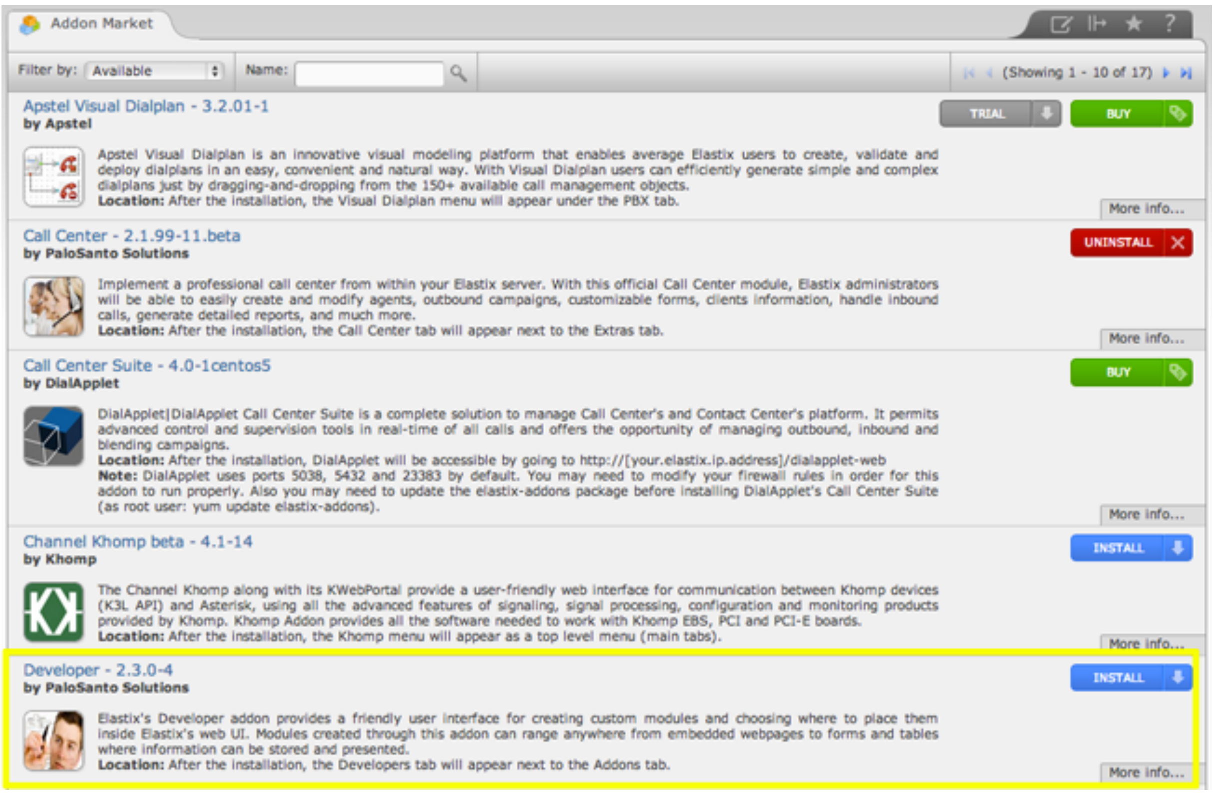
In this manual we'll explain how to create a new Elastix module, Basics of the Elastix Framework, and how to convert our new module into an Elastix Addon.

1. Creation of a new Elastix module

To create a new module on Elastix we must use “Developer” and addon included on the Elastix Market Place.

1.1 Installation of the Developer addon

To proceed with the installation, we have to go to Elastix GUI → Addons. We must be logged in as administrators.



Apstel Visual Dialplan - 3.2.01-1
by Apstel

Apstel Visual Dialplan is an innovative visual modeling platform that enables average Elastix users to create, validate and deploy dialplans in an easy, convenient and natural way. With Visual Dialplan users can efficiently generate simple and complex dialplans just by dragging-and-dropping from the 150+ available call management objects.
Location: After the installation, the Visual Dialplan menu will appear under the PBX tab.

Call Center - 2.1.99-11.beta
by PaloSanto Solutions

Implement a professional call center from within your Elastix server. With this official Call Center module, Elastix administrators will be able to easily create and modify agents, outbound campaigns, customizable forms, clients information, handle inbound calls, generate detailed reports, and much more.
Location: After the installation, the Call Center tab will appear next to the Extras tab.

Call Center Suite - 4.0-1centos5
by DialApplet

DialApplet|DialApplet Call Center Suite is a complete solution to manage Call Center's and Contact Center's platform. It permits advanced control and supervision tools in real-time of all calls and offers the opportunity of managing outbound, inbound and blending campaigns.
Location: After the installation, DialApplet will be accessible by going to http://[your.elastix.ip.address]/dialapplet-web
Note: DialApplet uses ports 5038, 5432 and 23383 by default. You may need to modify your firewall rules in order for this addon to run properly. Also you may need to update the elastix-addons package before installing DialApplet's Call Center Suite (as root user: yum update elastix-addons).

Channel Khomp beta - 4.1-14
by Khomp

The Channel Khomp along with its KWebPortal provide a user-friendly web interface for communication between Khomp devices (K3L API) and Asterisk, using all the advanced features of signaling, signal processing, configuration and monitoring products provided by Khomp. Khomp Addon provides all the software needed to work with Khomp EBS, PCI and PCI-E boards.
Location: After the installation, the Khomp menu will appear as a top level menu (main tabs).

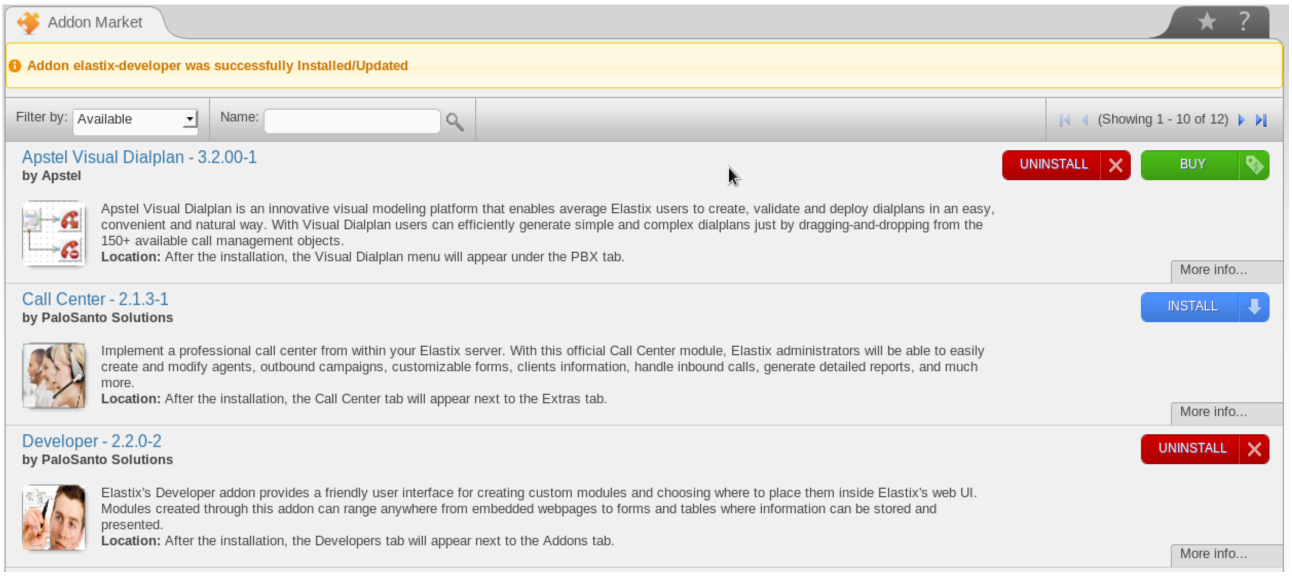
Developer - 2.3.0-4
by PaloSanto Solutions

Elastix's Developer addon provides a friendly user interface for creating custom modules and choosing where to place them inside Elastix's web UI. Modules created through this addon can range anywhere from embedded webpages to forms and tables where information can be stored and presented.
Location: After the installation, the Developers tab will appear next to the Addons tab.

Image 1. - Identifying the addon “Developer”.

- Once we've identified the addon, we click on its respective “Install” button

- After this the addon installation process begins, and we must wait a few minutes for the installation to end.



Addon elastix-developer was successfully Installed/Updated

Filter by: Available Name: (Showing 1 - 10 of 12)

Apstel Visual Dialplan - 3.2.00-1
by Apstel
UNINSTALL BUY
Apstel Visual Dialplan is an innovative visual modeling platform that enables average Elastix users to create, validate and deploy dialplans in an easy, convenient and natural way. With Visual Dialplan users can efficiently generate simple and complex dialplans just by dragging-and-dropping from the 150+ available call management objects.
Location: After the installation, the Visual Dialplan menu will appear under the PBX tab.
More info...

Call Center - 2.1.3-1
by PaloSanto Solutions
INSTALL
Implement a professional call center from within your Elastix server. With this official Call Center module, Elastix administrators will be able to easily create and modify agents, outbound campaigns, customizable forms, clients information, handle inbound calls, generate detailed reports, and much more.
Location: After the installation, the Call Center tab will appear next to the Extras tab.
More info...

Developer - 2.2.0-2
by PaloSanto Solutions
UNINSTALL
Elastix's Developer addon provides a friendly user interface for creating custom modules and choosing where to place them inside Elastix's web UI. Modules created through this addon can range anywhere from embedded webpages to forms and tables where information can be stored and presented.
Location: After the installation, the Developers tab will appear next to the Addons tab.
More info...

Image 2. - End of the installation of the “Developer” addon.

- After this we must exit the GUI and login again as administrator. We should see a new menu tab with the name “Developer”.

We will refer to this addon as “Elastix Developer”

1.2 Using the Elastix Developer

Elastix Developer allows us to transparently create the initial skeleton for the code necessary to create an Elastix module, which can help save hours of work in many cases.

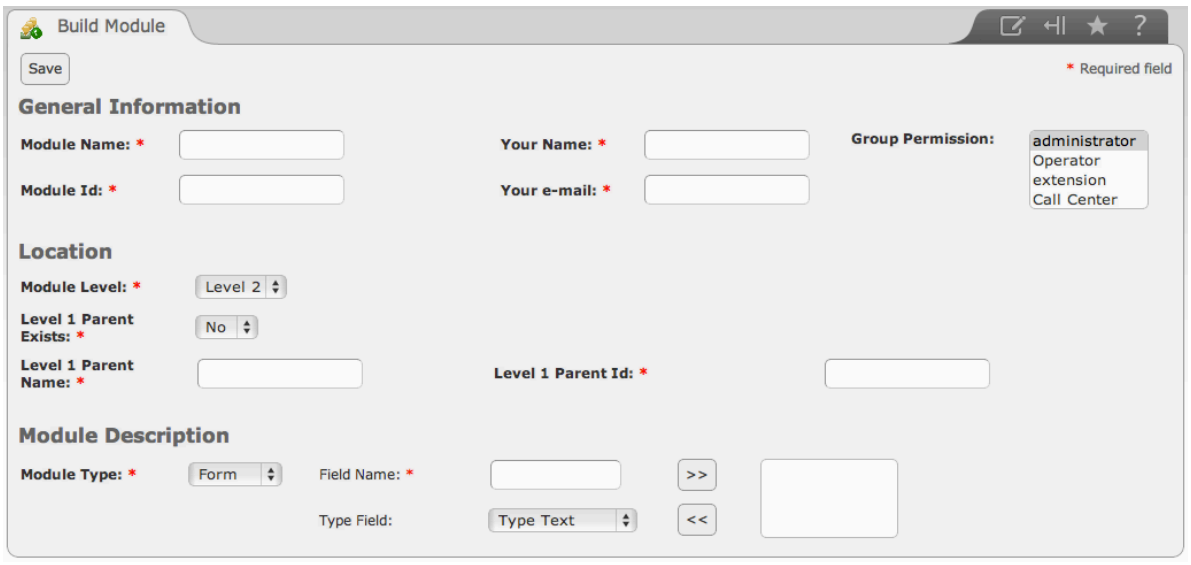
The Elastix Developer has three submenus, but the one we are interested is named “Build Module” and is the first submenu.

The function of the Build module is to generate the code skeleton of our new module, so we can continue with the development by extending this initial code.

As we create the module from here, we will generate the following process:

- The menu (or menus) within the Elastix Web interface. That is, we won't have to modify the database menu.db manually.
- The code skeleton, which will be placed in a folder with the same name as the module's ID. This folder will, itself, be located inside the folder `/var/www/html/modules` which is where all the Elastix modules are found. That is, we won't have to create this folder manually.
- Prototype screens. There are three types of screens: form, report, and frame.

Let's see how the module looks.



Build Module

Save

* Required field

General Information

Module Name: *		Your Name: *		Group Permission:	administrator, Operator, extension, Call Center
Module Id: *		Your e-mail: *			

Location

Module Level: *	Level 2		
Level 1 Parent Exists: *	No		
Level 1 Parent Name: *		Level 1 Parent Id: *	

Module Description

Module Type: *	Form	Field Name: *		>>
		Type Field:	Type Text	<<

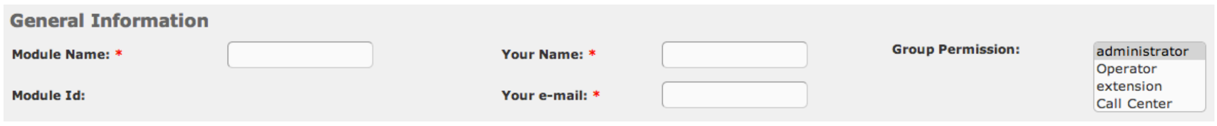
Image 3.- The module builder included in the addon Developer

As we can observe, the Build module is divided into three parts:

- General information.
- Location.
- Module Description.

General Information

This section allows us to define the name and unique identifier of the module, as well as the information of the person creating the module so that they'll appear in the header comments of every PHP program file. It also allows us to configure the level of accessibility that the module will have through the parameter *"Group permission".*



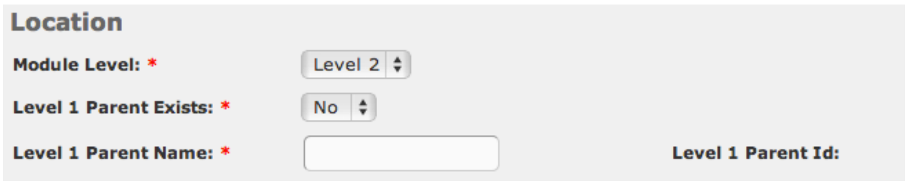
General Information

Module Name: *		Your Name: *		Group Permission:	administrator Operator extension Call Center
Module Id:		Your e-mail: *			

Image 4.- General information section within the module builder

Location

In this section we must setup the location, within the general menu, where we will place the new module. By default, this section starts at level 2.

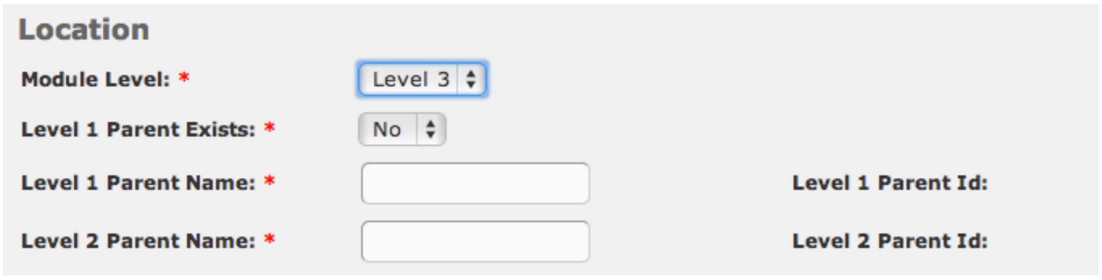


Location

Module Level: *	Level 2		
Level 1 Parent Exists: *	No		
Level 1 Parent Name: *		Level 1 Parent Id:	

Image 5.- Section where the module's location is configured

If we select level 3, then new fields will appear automatically for level 1 and level 2 identifiers.



Location

Module Level: *	Level 3		
Level 1 Parent Exists: *	No		
Level 1 Parent Name: *		Level 1 Parent Id:	
Level 2 Parent Name: *		Level 2 Parent Id:	

Image 6.- New fields appear if it says that the module will be level 3

To clarify the meaning of the fields shown in this section, let's look at the following table:

Field Name	Description
Module Level	Here we define the hierarchy level of the module
Level 1 Parent Exists	Here we define whether we place a new menu under an existing parent menu of level 1, or if we create a new one. To locate the new module under an existing menu we pick "Yes" and it will automatically display a list with the existing

	modules of level 1. If we pick “No”, we will create a new level 1 menu, then two new fields will appear to specify the name and the identifier.
Level 1 Parent Name	Here we define the name of the parent module. The first letter should be capital.
Level 1 Parent Id	Here we define the identifier of the parent module. All in lower case and with no spaces.
Level 2 Parent Exists	This option will appear if we choose level 3 in the field “Module level.” Similarly to what happens in: “Level 1 Parent Exists”, if we choose “No”, then two more fields will appear.
Level 2 Parent Name	Here we define the name of the level 2-parent module. The first letter should be capitalized.
Level 2 Parent Id	Here we define the identifier of the level 2-parent module. All in lower case and with no spaces.

Module description

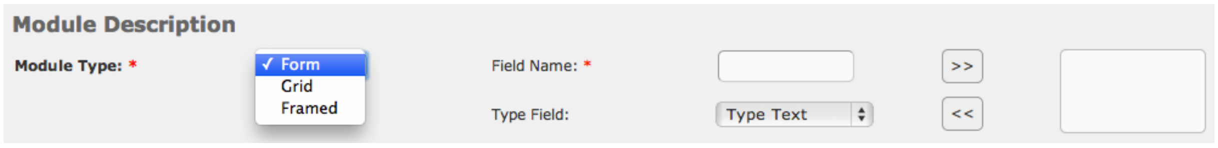
This last section is the most interesting since it is where the actual content of the module is created.

Below we'll explain the three types of modules that can be created.

- **Form:** Is used to gather data from the user. The Build module counts with support for the most common types of HTML fields such as: text, selection, date, text area, check box, radio, password, hidden, and file.

- **Grid:** It shows data organized in the form of a table, where some useful controls, such as navigation buttons, are included automatically. After generating the necessary code with the Build module, it is easier to link the Grid with a database in order to show useful information.

- **Framed:** Embeds an arbitrary URL in the screen. Very useful when we want to integrate external applications in the Elastix Web Interface. An example of this is the integration of vtigerCRM.


Module Description

Module Type: * ✓ Form / Grid / Framed

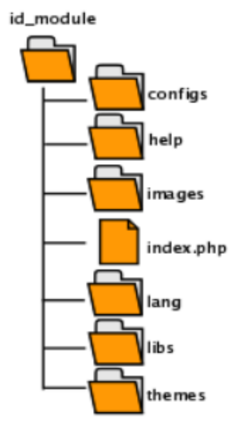
Field Name: *

Type Field: Type Text

Image 7.- Creation of a Form type screen

Buttons ">>", "<<" help us to add or remove fields from our form.

As we save the information established for the new module the following directory tree will be created in **documentRoot/modules** where “documentRoot” is `/var/www/html` for Elastix. There´s and exception for “framed” modules, where the module is simply a link to an existing URL.



As you will note, a module has the same general architecture of the framework, MVC2 web architecture. Therefore we define this as a MVC2 within another MVC2, a recursive definition of grade 2.

This allows us to make a similarity between the folders that have the following relations:

- themes is the view layer.
- index.php is the control layer.
- libs is the model layer.

1.3 Brief description of every folder created inside the module

configs

This folder is created by default with a file named “default.conf.php”, this file must contain the basic configuration for the module.

This file storage configuration variable that will be used very often in the module, e.g. DSN

Below there are two DSNs used in Elastix:

- Sqlite3: “`sqlite3:///$arrConf[elastix_dbdir]/base.db`”
- Mysql: We can use the function “generarDSNSistema” (Generate DSN System), which returns the connection string. This function can be found in `/var/www/html/libs/misc.lib.php`
- Below we'll see the function in more detail.

Note: The following information is only available in Spanish. We are providing a translation for explanation purposes.

```
/**
 * Función para construir un DSN para conectarse a varias bases de datos
 * frecuentemente utilizadas en Elastix. Para cada base de datos reconocida, se
 * busca la clave en /etc/elastix.conf o en /etc/amportal.conf según corresponda.
 *
```

Translation: Function to build a DSN to connect to multiple databases frequently used in Elastix. The key for each recognized database is looked up on one of the following files /etc/elastix.conf or /etc/amportal.conf.

```
* @param  string  $sNombreUsuario    User name to interrogate
* @param  string  $sNombreDB         Database name for DSN
* @param  string  $ruta_base         Base route for library inclusion
*
* @return  mixed   NULL if the user is not recognized, or the DSN with the correspondent key.
*/
function generarDSNSistema($sNombreUsuario, $sNombreDB, $ruta_base='')
```

help

In this folder we can find a file named “id_modulo.hlp”. This is the file that is used to show the help embedded in the module.

It's recommended that once you have finished your whole module you create an embedded help that is nothing more than a user's manual, indicating the different options offered in the module. It's recommended that it is written in English.

Below there is a simple example of the code in this file. This help will only show the title of the module followed by a brief description and an image in the indicated path.

```
<html>
<header>
  <link rel="stylesheet" href="/themes/{$THEMENAME}/styles.css">
  <link rel="stylesheet" href="/themes/{$THEMENAME}/help.css">
</header>
<body>
```

```
<h1>{$node_name}</h1>

<p align="Justify">This is the embedded help for my module.</p>

<div><img src="../modules/{$node_id}/images/image.png" border="0"></div>
<div>Figure 1</div><br/>

</body>
</html>
```

images

In this folder all the module's images will be saved, for example the icon, images used in the embedded help, etc.

lang

As we know, the Elastix Framework supports language translation. This folder stores the translations for our module. Each translation for each language is stored in a different file according to the following chart:

File	***Language***
bg.lang	Bulgarian
br.lang	Portuguese
ca.lang	Catalan
cn.lang	Simplified Chinese
da.lang	Danish
de.lang	German
en.lang	English
el.lang	Greek
es.lang	Spanish
fa.lang	Persian
fr.lang	French
it.lang	Italian
hu.lang	Hungarian
hr.lang	Croatian
pl.lang	Polish
ro.lang	Romanian
ru.lang	Russian
sl.lang	Slovenian
sv.lang	Swedish
ko.lang	Korean
ja.lang	Japanese
sr.lang	Serbian

We recommend creating at least two translation files, one with the developer's native language and one in English

Therefore, if we want to create the translation of our module to Spanish we need to create a file in this folder named **es.lang**. This file is basically a single array in which the key is the word to translate and the value is the translated key. For example:

```
<?php
global $arrLangModule;
$arrLangModule=array(
"Module" => "Módulo",
"This is a test module" => "Este es un módulo prueba”,
);
?>
```

It is recommended that the keys in our array of translations are in English

libs

As was mentioned previously, this is our model layer. In this folder there will be a library that will be in charge of making queries to databases, modifying files, etc.

All the queries should have parameters, which will prevent arbitrary SQL code injection to our base. We will explain how to do this later.

Our apache service works using the user “asterisk” so we will only have access to files and folders to which “asterisk” has access. You will not be able to read or write to a file or folder that has, for example, only root permissions.

Do not use console commands that require “root” privileges. This can lead to our script or library becoming a potential security hole.

But what happens if our script or library must use commands that require “root” privileges? That is when the script elastix-helper comes into play.

1.4 Use of elastix-helper

elastix-helper is a script located in `/usr/bin`. This script allows the execution of privileged scripts found in the following folder:
`“/usr/share/elastix/privileged”`.

It’s here where we'll find the script that will carry out tasks that require “root” privileges. This script must be owned by “root” and belong to the “root” group” as well as having 755 permissions. To use it we must execute the following code in the function we wish to use from our library:

```
exec("/usr/bin/elastix-helper script_privileged parameter", $output, $ret);
```

- script_privileged.- is the name of the privileged script found on that path.
- parameter.- is the parameter that will be given to the script “script_privileged”. One, or more parameters can be given to it, simply separate them with spaces; it is also possible not to give parameters.
- If the value of $ret is 0, that means that there were no problems.

themes

This folder is the view layer. Inside this folder we will find another folder named “default” which contains all the themes available for the module. For example, a module may have a grid as main view, this means that it will need a theme (a tpl file) containing filters for the grid (in case they exist, otherwise a tpl file would not be needed), and it is also possible that a form is shown when we press a button, for which we would need another tpl file with the form fields. Below we will show a typical template for a tpl theme and another for a form.

[Continues in the next page]

Typical template for a module with a grid theme

```
<table width="99%" border="0" cellspacing="0" cellpadding="0" align="center">
    <tr class="letra12">
       <td width="12%" align="left"><input class="button" type="submit"

           name="new" value="{$New}"></td>
       <td width="10%" align="left">&nbsp;&nbsp;</td>
       <td width="10%" align="right">
       {$filter_type.LABEL}:&nbsp;&nbsp;{$filter_type.INPUT}&nbsp;&nbsp;
       {$filter_txt.INPUT}
          <input class="button" type="submit" name="show" value="{$SHOW}" />
       </td>
   </tr>
</table>
```

As we can see, this theme has a “New” button with a filter.

Typical template for a module with a form theme

```
<table width="99%" border="0" cellspacing="0" cellpadding="4" align="center">
    <tr class="letra12">
       <td align="left"><input class="button" type="submit" name="save"
                       value="{$SAVE}"></td>
    </tr>
</table>
<br />
<div class="tabForm" style="font-size: 16px; height: auto;" width="100%">
    <table style="font-size: 16px;" width="100%" cellspacing="0" cellpadding="8">
            <tr class="letra12">
                <td align="left" width="130px"><b>{$manufacturer.LABEL}: </b></td>
                <td align="left">{$manufacturer.INPUT}</td>
            </tr>
   </table>
</div>
```

For this form it is enough to have a “Save” button with a field named “manufacturer.”

These are basic templates; in fact, we can make them as complex as we require.

If we want to include javascripts in our module, we only need to create a folder named “js” inside themes/default, and place the javascripts we want (they must have the extension .js). If we want to include css we need to create a folder named “css” inside themes/default and place the css we want (they must have the extension .css). The framework will take care of including these files automatically.

index.php

The file index.php of our module represents the control layer; the framework is responsible of directing the petition to this file calling on the function "_moduleContent". This file is in charge of communicating with the view and model layers.

If we used the Elastix Developer to create the module, we will see that this file is already written with a default template. We can make all the modifications we need over this template according to the requirement of our module.

2. Using the Elastix Framework

We have created our module with a basic configuration, but we still do not have a clear knowledge of the different facilities and libraries offered to us by the Elastix framework.

In the following sections we will detail some of these libraries and classes with their most important functions. All of these libraries are found at `/var/www/html/libs`

2.1 Library misc.lib.php

In this library we will find several types of functions. It is not necessary to include it in the module since the Elastix framework is in charge of this task. We will list the most relevant functions and the ones we will use often.

- `function _tr($s)`

This is the function required for text translation. It receives the text to translate as a string parameter.

The function works in the following way:

The function looks for the string passed as a parameter in the keys of the language array of the framework and module (.lang file inside lang folder on the module; it will choose the corresponding folder to the language in which the Elastix server is configured), in case it finds it, the function returns the value of the array for that key, otherwise it will return the same string passed as parameter.

Example:

If we have selected Spanish as our language and we want a translation of "Hi this is my first module" (which is previously defined in "es.lang"), we simply need to do the following:

```
$translate = _tr("Hi this is my first module");
```

- `function getParameter($parameter)`

This function is used when we want to obtain a parameter that has been sent by POST or GET. First it searches for the parameter in the array $_POST and, if it exists, it will return $_POST[$parameter]. If it doesn't exist, it'll look up in the array $_GET, if it exists, it will return $_GET[$parameter], if it doesn't exist in $_POST nor $_GET the function returns **NULL**.

Example:

Let's assume that we have a module with a save button with the name "save" and we want to know if that button was pressed,

We would have to do the following:

```
if(getParameter("save"))
```

In this way it will enter "if", if the save button was pressed.

- `function obtenerClaveCyrusAdmin($ruta_base='')`

This function returns the "admin" password for Cyrus. The parameter $ruta_base must be entered as "`/var/www/html/`" if we call this function outside of this path. This function parses the file `/etc/elastix.conf` and searches for the keyword cyrususerpwd, in case it finds it, it will return its value; otherwise it will return "palosanto" which is the default password.

- `function obtenerClaveAMIAdmin($ruta_base='')`

This function returns the AMI password (Asterisk Manager Interface) for the admin user. The parameter $ruta_base must be passed as "`/var/www/html/`" if we call this function outside of this path. This function searches for the password in the file `etc/elastix.conf`, if the keyword “amiadminpwd” is found, it returns its value; otherwise it will return “elastix456” which is the default AMI password in Elastix.

- **function** `generarDSNSistema($sNombreUsuario, $sNombreDB, $ruta_base='')`

We have seen this function before in this manual, now we will explain more about it. As we said before, this function returns the DSN (Data Source Name) for a connection with mysql.

The parameter $ruta_base must be passed as `"/var/www/html/"` if we call this function outside of that path.

The parameter $sNombreUsuario is the user we use to connect to mysql; it can be “root” or “asteriskuser”.

The parameter $sNombreDB is the name of the mysql database to which we will connect.

- **function** `writeLOG($logFILE, $log)`

This function allows writing in a log. The parameter $logFILE is the name of the log file, which will be located under `/var/log/elastix`.

The parameter $log is the text that will be written in the log. If the $logFILE file doesn't exist, it creates it, if it already exists, it adds the text to the end of the $log file.

Once more, it’s important to remember that the httpd service used by Elastix has “asterisk” as a user, meaning that it will only be able to write to files to which “asterisk” has permissions. In case that $logFILE doesn't exist, there is no problem since “asterisk” is owner of the `/var/log/elastix` folder, so that it will be able to create files there without any problems.

Example:

If we want to write “You have entered the test module” in a log called “myModule.log” we would have to do the following:

```
writeLOG("myModule.log"," You have entered the test module");
```

In this way, the log “myModule.log” will be created (if it doesn't exist), containing the following:

```
[Jan 19 14:48:11] You have entered the test module
```

The date to the left is the date of the server at the moment of writing the message in the log.

2.2 paloSantoDB.class.php Class

This class is in charge of creating an object with the connection to a database. The objective of this class is to encapsulate the database connection process so that the developer will simply have to instance this class and carry out the queries he needs. It’s not necessary to include this class in our module since the Elastix framework already takes care of this.

To instance the class, we must pass the DSN of the database.

Example:

Let's suppose that we want to instance a paloDB class in our module to create a connection to a mysql database named “myBase”, accessible just by the “root”. We would have to do the following:

```
$dsn =  generarDSNSistema(“root”,”myBase”);
$pDB = new paloDB($dsn);
```

Once this class is instanced, remember to pass it for reference to the other functions required, this will be very important, especially when we work with transactions.

Below we've detailed some of its functions.

- `function genQuery($query, $param = NULL)`

Is the procedure to execute a SQL sentence that does not return queues or results. In case of an error it assigns to the variable class $this->errMsg. It is only used to manipulate data of the database.

The parameter $query is the string that contains the query that will be executed.

The parameter $param is an array that is only passed on to this function when parameterized queries are carried out, it is recommended that all queries are parameterized, especially when there are variables entered into the server by the client.

Example:

Let's suppose that we want to insert a new entry in the table “myTable”. The fields are “field1” and “field2” with the values $value1 and $value2 respectively, and arrive to the server by the client. We would have to do the following (let's suppose that we have already instanced the class with the corresponding base's DSN in the $pDB variable).

```
$query = "INSERT INTO myTable (field1,field2) VALUES (?,?)";
$arrParam = array($value1, $value2);
$result = $pDB->genQuery($query,$arrParam);
if($result == FALSE)
      echo _tr("Query Error")." ".$pDB->errMsg;
else
      echo _tr("Query successfully executed");
```

The order in which the array $arrParam is placed is very important since queries will be assigned according to this order.

- **function** fetchTable($query, $arr_colnames = **FALSE**, $param = **NULL**)

Is the procedure that recovers all the rows resulting from an SQL petition that returns one or more rows.

The parameters $query and $param have the same purpose as the ones described in the previous function.

The parameter $arr_colnames will be **FALSE** if we want that each tuple has and incremental number as an index, if it is **TRUE** each tuple will have the name of the column as an index.

Example:

Let's suppose that we want to print, on the screen, the values of the column “field1” from the table “myTable” when “field2” has as value $value2, and we assume that we already have the object $pDB.

```
$query = "SELECT field1 FROM myTable WHERE field2=?";
$arrParam = array($value2);
$result = $pDB->fetchTable($query,TRUE,$arrParam);
if($result === FALSE)
      echo _tr("Query Error")." ".$pDB->errMsg;
else{
      if(count($result) > 0){
            foreach($result as $value){
                  echo $value["field1"]."<br />";
            }
      }
      else
            echo _tr("There is no data for the criteria search");
}
```

Whenever we use the functions fetchTable or getFirstRowQuery described ahead, we need to compare the value returned by these functions with a triple equality ===, remember that these functions can return empty arrays that are also evaluated as **FALSE**, using === we do not only compare by value but by data type as well.

- `function getFirstRowQuery($query, $arr_colnames = FALSE, $param = NULL)`

This function is the procedure to recover a single row from the query that returns one or more rows. It returns a row with fields if the query returns at least one row, otherwise it returns an empty array or **FALSE** in case of error.

The parameters $query, $arr_colnames and $param have the same purposes as the ones described in the fetchTable function.

Example:

Let's suppose that we want to print, on the screen, the number of entries in the table “myTable” whose “field1” has as value $value1

```
$query = "SELECT COUNT(*) FROM myTable WHERE field1=?";
$arrParam = array($value1);
$result = $pDB-> getFirstRowQuery($query,FALSE,$arrParam);

if($result === FALSE)
      echo _tr("Query Error")." ".$pDB->errMsg;
else
      echo $result[0];
```

- `function beginTransaction()`

This function is the procedure to initiate a transaction. Remember that a transaction is used when we want the database to return to its previous state if an unwanted or unexpected event occurs.

- `function rollBack()`

This function is the rollBack procedure for a transaction.

- `function commit()`

This function is the commit procedure for a transaction.

Example:

Let's suppose that we want to create a function that inserts $value1 and $value2 in “field1” and “field2” on “myTable1”, respectively* (this table has “id” as an auto incrementing id field). Later, on “myTable2”, we want to insert the id of the entry we recently registered on “id_myTable1” of “myTable1”, and the actual date of the field “date”.

***Note:** There cannot be another entry with the exact same values both in “field1” and “field2”.

```
function insertRegister($value1, $value2, &$pDB, &$errMsg)
{
      $pDB->beginTransaction();
      $query1 = "INSERT INTO myTable1 (field1,field2) VALUES (?,?)";
      $arrParam1 = array($value1, $value2);
      $result1 = $pDB->genQuery($query1,$arrParam1);
      if($result1 == FALSE){
            $pDB-> rollBack();
            $errMsg = $pDB->errMsg;
            return FALSE;
      }
```

```
        else{
              $query2 = "INSERT INTO myTable2 (id_myTable1,date)
                     VALUES((SELECT id FROM myTable1 WHERE
                field1=? AND field2=?),?)";


              $arrParam2 = array($value1,$value2,date('Y-m-d H:i:s'));
              $result2 = $pDB->genQuery($query2,$arrParam2);
              if($result2 == FALSE){
                    $pDB->rollBack();
                    $errMsg = $pDB->errMsg;
                    return FALSE;


              }
              else{
                    $pDB->commit();
                    return TRUE;
              }
        }
  }
```

As we can observe, the object $pDB was passed by reference to the function, which is necessary so that the transaction has the desired behavior.

NOTE: *Take into account that there are database engines that do not support transactions, such as the MyISAM mySQL engine. In these cases we must assure that the table we want to use has a compatible engine with transactions, such as InnoDB or BDB.*

2.3 Class paloSantoACL.class.php

This class is in charge of administrating access control lists for the different kinds of users. The Elastix framework includes it automatically.

To instance the class one must pass the DSN string for the connection to the database "acl.db" or one can also pass it an object that is an instance of the class paloDB that was passed the DSN for "acl.db".

The variable $arrConf['elastix_dsn']['acl'] already contains the DSN for "acl.db", this variable is created by the framework. Therefore, if we want to instance this class in our module, we would do the following:

```
global $arrConf;
$pACL = new paloACL($arrConf['elastix_dsn']['acl']);
```

This class is somewhat delicate and could compromise the system if used incorrectly. It is recommended to only use the functions described below and to let the Elastix default modules take care of other user administration tasks.

- **function** `getUserExtension($username)`

Procedure to obtain the extension of a user through the username. As can be imagined, the $username parameter is the name of the user whose associated extension we want to obtain.

Example:

Let us suppose that we want to obtain the extension associated with the user that is logged on:

```
//The logged-in username is stored in the $_SESSION["elastix_user"] session
variable
$username = $_SESSION["elastix_user"];
$extension = $pACL->getUserExtension($username);
```

- **function** `isUserAdministratorGroup($username)`

Procedure to find out if a user belongs to the “administrator” group or not. Here too, the parameter $username is the name of the user about whom we want to learn whether it belongs or doesn't belong to the “administrator” group.

Example:

If we want our module to carry out certain tasks if the logged user is in the “administrator” group and others if he is not, we could do the following:

```
$username = $_SESSION["elastix_user"];
if($pACL->isUserAdministratorGroup($username)){
      //Do some task for administrators
}
else{
      //Do some task for non administrators
}
```

2.4 Class paloSantoConfig.class.php

This class is very useful especially for parsing configuration files, allowing us to read or write to them.

It is necessary to include this class in our module in case we require it.

```
include_once "libs/paloSantoConfig.class.php";
```

This class has the following builder:

- **function** paloConfig($directorio, $archivo, $separador="", $separador_regexp="", $usuario_proceso=**NULL**)

Where $directorio (directory) is the path where the file is found, $archivo (file) is the file to be parsed, $separador (separator) is the string that separates the keywords with its value, $separador_regexp is a regular expression to be interpreted as a separator, and $usuario_proceso (process user) is the user that initiates the process. As can be observed, only $directorio and $archivo are required parameters, the rest are optional and have default values.

Let's suppose that we have the configuration file `/etc/myModule.conf` which contains the following:

user = user
password = 12345
email = user@domain.com
privileges = all

Now we will instance to the class paloConfig to parse this file.

```
$pConfig = new paloConfig("/etc","myModule.conf"," = ","\s*=\s*");
```

In this way we will go to parse the file "/etc/myModule.conf" which has as separator the "=" symbol and can be or not be accompanied with blank spaces to either side.

Now we will detail some of the main functions of the class paloConfig.

- **function** leer_configuracion($bComentarios=**true**)

This procedure initiates the reading of a file by storing it in an associative array that is returned as an answer. If the parameter $bComentarios is **FALSE** then only the values that are not comments in the array will be found, and the return array will have as indexes the keywords in the file, but if it is **TRUE** then in the returned array one will find both comments and configuration values, and the indexes for this will be numerical.

- `function escribir_configuracion($arr_reemplazos, $overwrite=FALSE)`

This procedure is used to write in the configuration file. The parameter $arr_reemplazos is an array that contains the changes to be made, where the index of the array represents the keyword to be modified in the file. If the keyword is found in the file then it is modified and if it is not found then it adds it. If the parameter $overwrite is **FALSE** then the changes contained in $arr_reemplazos will be made, but the rest of the file will be kept intact, but if it is TRUE, then the file will be overwritten by $arr_reemplazos

- `function privado_get_valor($lista, $clave)`

Procedure that returns the value of a keyword in the file. The parameter $lista is the array that contains the configuration file. The parameter $clave is the keyword to be searched for in the file.

Example:

Let's suppose that we have the file `/etc/myModule.conf` with the same information that was described above, and we want to do the following: if “password” is equal to “12345” then we will change it to “new12345”.

```
include_once "libs/paloSantoConfig.class.php";
$pConfig = new paloConfig("/etc","myModule.conf"," = ","\s*=\s*");
$content = $pConfig->leer_configuracion(FALSE);
$password = $pConfig->privado_get_valor($content,"password");
if($password == "12345"){
      $arrReplaces = array("password" => "new12345");
      $pConfig->escribir_configuracion($arrReplaces);
}
```

2.5 Class paloSantoForm.class.php

This class is used to easily manage the form-type modules. It is necessary to include it in our module in case we wish to use it.

The builder for this class has the following form:

- `function paloForm(&$smarty, $arrFormElements)`

Where $smarty is an instance of smarty, (which is passed to the function _moduleContent in our module) and $arrFormElements is an array of arrays that contains the elements of the form. In the main array, the indexes represent the id of the element and in the secondary array, there must always exist an index named “INPUT_TYPE” that indicates the type of element that is required. The value of “INPUT_TYPE” can be one of the following: “TEXTAREA”, “TEXT”, “CHECKBOX”, “PASSWORD”, “HIDDEN”, “FILE”, “RADIO”, “SELECT” or “DATE”. Where each of these words represents the element desired. Below we show some examples of each of these types of elements. After these examples, more about the other elements of the array will be explained.

TEXTAREA

Example:
A textarea is wanted with the label “descripción” and 6 columns and 4 rows.

```
$arrFormElements = array(
   "description" => array(   "LABEL"   => _tr("Description"),
            "REQUIRED"                 => "yes",
            "INPUT_TYPE"               => "TEXTAREA",
            "INPUT_EXTRA_PARAM"        => array("style" => ″width:400px"),
            "VALIDATION_TYPE"          => "text",
            "VALIDATION_EXTRA_PARAM"   => "",
            "ROWS"                     => "4",
            "COLS"                     => "6"
                   ),
        );
```

TEXT

Example:
An input is wanted to enter the name of a client.

```
$arrFormElements = array(
        "name"   => array(   "LABEL"  => _tr("Name"),
            "REQUIRED"                => "yes",
            "INPUT_TYPE"              => "TEXT",
            "INPUT_EXTRA_PARAM"       => array("style" => ”width:200px"),
            "VALIDATION_TYPE"         => "text",
            "VALIDATION_EXTRA_PARAM"  => ""
                    ),
        );
```

CHECKBOX

Example:
A checkbox is wanted that says “Enable”.

```
$arrFormElements = array(
        "enable"   => array( "LABEL"  => _tr("Enable"),
            "REQUIRED"                => "yes",
            "INPUT_TYPE"              => "CHECKBOX",
            "INPUT_EXTRA_PARAM"       =>  " ",
            "VALIDATION_TYPE"         => "text",
            "VALIDATION_EXTRA_PARAM"  => ""
                    ),
        );
```

NOTE: When creating elements of this type, the framework automatically creates two elements, one is the actual checkbox and the other is a hidden element whose value is “on” in case the checkbox is activated or “off” otherwise. Therefore, if in our module we want to know if the checkbox was activated or not, we would have to do the following:

```
$enable = getParameter(“enable”);
if($enable == “on”){
      //Do something
}
else{
      //Do something
}
```

PASSWORD

Example:
A field is wanted where the user enters a password.

```
$arrFormElements = array(
    "password"   => array( "LABEL"    => _tr("Password"),
            "REQUIRED"                => "yes",
            "INPUT_TYPE"              => "PASSWORD",
            "INPUT_EXTRA_PARAM"       =>array("style" => ″width:200px"),
            "VALIDATION_TYPE"         => "text",
            "VALIDATION_EXTRA_PARAM"  => ""
                   ),
     );
```

HIDDEN

Example:
A hidden field is wanted in order to store a user's id.

```
$arrFormElements = array(
        "id"   => array( "LABEL"      => " ",
            "REQUIRED"                => "yes",
            "INPUT_TYPE"              => "HIDDEN",
            "INPUT_EXTRA_PARAM"       =>  " ",
            "VALIDATION_TYPE"         => "text",
            "VALIDATION_EXTRA_PARAM"  => ""
                   ),
        );
```

FILE

Example:
A field is wanted in order to pass it the path of a file.

```
$arrFormElements = array(
        "file"   => array(   "LABEL"  => _tr("File"),
            "REQUIRED"                => "yes",
            "INPUT_TYPE"              => "FILE",
            "INPUT_EXTRA_PARAM"       =>  " ",
            "VALIDATION_TYPE"         => "filename",
            "VALIDATION_EXTRA_PARAM"  => ""
                  ),
        );
```

RADIO

Example:

Two radio buttons are wanted in order to indicate the gender of a person.

```
$gender = array("m" => _tr("Male"), "f"  => _tr("Female"));
$arrFormElements = array(
        "gender"   => array( "LABEL"  => _tr("Gender"),
            "REQUIRED"                => "yes",
            "INPUT_TYPE"              => "RADIO",
            "INPUT_EXTRA_PARAM"       =>  $gender,
            "VALIDATION_TYPE"         => "text",
            "VALIDATION_EXTRA_PARAM"  => ""
                 ),
        );
```

SELECT

Example:

A combo is desired to select the payment method that a client will realize, with the options “cash”, “credit card”, “check”, or “bank transfer.”

```
$paymentMethod = array("cash" => _tr("Cash"), "credit_card" => _tr("Credit
Card"), "check" => _tr("Check"),"bank_transfer" => _tr("Bank Transfer"));
$arrFormElements = array(
        "paymentMethod" => array("LABEL"  => _tr("Payment Method"),
            "REQUIRED"                    => "yes",
            "INPUT_TYPE"                  => "SELECT",
            "INPUT_EXTRA_PARAM"           =>  $paymentMethod,
            "VALIDATION_TYPE"             => "text",
            "VALIDATION_EXTRA_PARAM"      => ""
                  ),
        );
```

DATE

Example:

A field is wanted in order to enter the date in which a payment took place.

```
$arrFormElements = array(
     "paymentDate"   => array( "LABEL"    => _tr("Payment Date"),
         "REQUIRED"                 => "yes",
         "INPUT_TYPE"               => "DATE",
         "INPUT_EXTRA_PARAM"        => "",
         "VALIDATION_TYPE"          => "ereg",
         "VALIDATION_EXTRA_PARAM"   => "^[[:digit:]]{1,2}[[:space:]]+[[:alnum:]]
                                          {3}[[:space:]]+[[:digit:]]{4}$"
                  ),
        );
```

The format by default for the date is the day in two digits, then space followed by the month in three letter format, then space, and the year in four digits. If one wants to change this format, one would simply have to place in “INPUT_EXTRA_PARAM” the new format and, if we want, we can also enter the time by using “TIME” => true.

For example, if we want the format to be yyyy-mm-dd h:m:s

```
"INPUT_EXTRA_PARAM"=>array("TIME"=> true, "FORMAT" => "%Y-%m-%d %H:%M:%S"),
```

As could be seen in the previous examples, there are indexes where all coincided and others that were simply specific to the type of element, like for example, the index “COLS” in “TEXTAREA”. Now we will explain about obligatory and common indexes.

LABEL: The value of this index, as its name suggests, will be the descriptive label that is shown beside an element.

REQUIRED: The value of this index can be “yes” or “no”, if it is “yes” then this field is obligatory in order to be able to save the form, otherwise this can be left blank.

INPUT_TYPE: Type of element, already described previously.

INPUT_EXTRA_PARAM: Additional input parameters, can be extra styles or attributes. These must be passed as an array, otherwise it must be an empty string.

VALIDATION_TYPE: Indicates the type of validation that will be applied to the value entered by the user. The values can be the following:

text – The user can enter anything.

ereg – A regular expression must be passed in VALIDATION_EXTRA_PARAM and the user will only be allowed to enter a text that matches the regular expression that was entered.

filename – Validates that what is entered into that field is the name of a file.

domain – Validates that what is entered into that field is the name of a domain.

filepath – Validates that what is entered into that field is a path to a file.

ip – Validates that what is entered into that field is an IP address.

mask – Validates that what is entered into that field is a network mask.

ip/mask – Validates that what is entered into that field is an IP address followed by "/" and the network mask in decimal format.

numeric – Validates that what is entered into that field is a number.

float – Validates that what is entered into that field is a floating number (decimal with a point as a separator).

numeric_array **–** Validates that what is entered into that field is an array whose elements are numbers.

ereg_array **–** Validates that what is entered into that field is an array with values that must match the regular expression passed in VALIDATION_EXTRA_PARAM.

email **–** Validates that what is entered into that field is an email address.

VALIDATION_EXTRA_PARAM: An additional parameter is passed in case VALIDATION_TYPE is required, as is the case with "ereg".

Below are described the functions that this class offers.

- `function fetchForm($templateName, $title, $arrPreFilledValues = array())`

This function generates a chain that contains an HTML form. To do this, take a form template (which is passed in the parameter $templateName) and insert into it the elements of the form. The parameter $title is the title that the form will have and the parameter $arrPreFilledValues is an array that contains the default values for the form, where the index is the element's id and its value is the value that the field would have by default.

- `function validateForm($arrCollectedVars)`

This function returns **TRUE** in case that the data entered into the form was correct, and otherwise **FALSE**. The parameter $arrCollectedVars contains the values entered into the form.

Example:

Let's suppose that we want a form-type module that has the following fields: "Name" which is a text box, "Last name" which is a text box, "Gender" which is a radio button that can be either masculine or feminine, "Email" that is a text box, and "Marriage status” that can either be single, widowed, married, divorced, or civil union. All of these fields are required. There is also a “Save” button that, when pressed, validates the data entered, in case of an error it maintains the persistence of the data and the error is indicated, otherwise it shows a message showing the data entered.

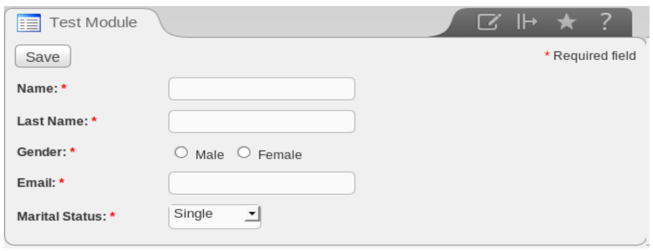
File themes/default/form.tpl

```
<table width="100%" border="0" cellspacing="0" cellpadding="4"
align="center">
    <tr class="letra12">
        <td align="left">
            <input class="button" type="submit" name="save"
value="{$SAVE}">
        </td>
        <td align="right" nowrap><span class="letra12"><span
class="required">*</span> {$REQUIRED_FIELD}</span></td>
    </tr>
</table>
<table class="tabForm" style="font-size: 16px;" width="100%" >
    <tr class="letra12">
        <td align="left"  width="130px"><b>{$name.LABEL}: <span
class="required">*</span></b></td>
        <td align="left">{$name.INPUT}</td>
    </tr>
    <tr class="letra12">
        <td align="left"><b>{$last_name.LABEL}: <span
class="required">*</span></b></td>
        <td align="left">{$last_name.INPUT}</td>
    </tr>
    <tr class="letra12">
        <td align="left"><b>{$gender.LABEL}: <span
class="required">*</span></b></td>
        <td align="left">{$gender.INPUT}</td>
    </tr>
    <tr class="letra12">

        <td align="left"><b>{$email.LABEL}: <span
class="required">*</span></b></td>
        <td align="left">{$email.INPUT}</td>
    </tr>
    <tr class="letra12">
        <td align="left"><b>{$marital_status.LABEL}: <span
class="required">*</span></b></td>
        <td align="left">{$marital_status.INPUT}</td>
    </tr>
</table>
```

File index.php

```php
<?php

include_once "libs/paloSantoForm.class.php";

function _moduleContent(&$smarty, $module_name)
{
    //include module files
    include_once "modules/$module_name/configs/default.conf.php";

    //include file language agree to elastix configuration
    //if file language not exists, then include language by default (en)
    $lang=get_language();
    $base_dir=dirname($_SERVER['SCRIPT_FILENAME']);
    $lang_file="modules/$module_name/lang/$lang.lang";
    if (file_exists("$base_dir/$lang_file")) include_once "$lang_file";
    else include_once "modules/$module_name/lang/en.lang";
    //global variables
    global $arrConf;
    global $arrConfModule;
    global $arrLang;
    global $arrLangModule;
    $arrConf = array_merge($arrConf,$arrConfModule);
    $arrLang = array_merge($arrLang,$arrLangModule);

    //folder path for custom templates
    $templates_dir=(isset($arrConf['templates_dir']))?
$arrConf['templates_dir']:'themes';

$local_templates_dir="$base_dir/modules/$module_name/".$templates_dir.'/'.
$arrConf['theme'];

    //conexion resource
    //$pDB = new paloDB($arrConf['dsn_conn_database']);
    $pDB = ""; //In this case we do not use a database

    //actions
    $action = getAction();
    $content = "";

    switch($action){
        case "save":
            $content = saveTestModule($smarty, $module_name,
$local_templates_dir);
            break;
        default: // view_form
           $content = viewFormTestModule($smarty, $module_name,
$local_templates_dir);
```

```
            break;
    }
    return $content;
}

function viewFormTestModule($smarty, $module_name, $local_templates_dir)
{
    $arrFormTestModule = createFieldForm();
    $oForm = new paloForm($smarty,$arrFormTestModule);

    //begin, Form data persistence to errors and other events.
    $_DATA  = $_POST;

    $smarty->assign("SAVE", _tr("Save"));
    $smarty->assign("REQUIRED_FIELD", _tr("Required field"));
    $smarty->assign("icon", "images/list.png");

    $htmlForm = $oForm->fetchForm("$local_templates_dir/form.tpl",_tr("Test
Module"), $_DATA);
    $content = "<form  method='POST' style='margin-bottom:0;'
action='?menu=$module_name'>".$htmlForm."</form>";

    return $content;
}

function saveTestModule($smarty, $module_name, $local_templates_dir)
{
    $arrFormTestModule = createFieldForm();
    $oForm = new paloForm($smarty,$arrFormTestModule);
    if(!$oForm->validateForm($_POST)){

        // Validation basic, not empty and VALIDATION_TYPE
        $smarty->assign("mb_title", _tr("Validation Error"));
        $arrErrores = $oForm->arrErroresValidacion;
        $strErrorMsg = "<b>"._tr("The following fields contain
errors").":</b><br/>";
        if(is_array($arrErrores) && count($arrErrores) > 0){
            foreach($arrErrores as $k=>$v)
                $strErrorMsg .= "$k, ";
        }
        $smarty->assign("mb_message", $strErrorMsg);
        return viewFormTestModule($smarty, $module_name,
$local_templates_dir);
    }
    else{
        //Here are extra validations
            $name = getParameter("name");
            $last_name = getParameter("last_name");
            $gender = getParameter("gender");
            $email = getParameter("email");
            $marital_status = getParameter("marital_status");
            if(!in_array($gender,array("male","female"))){
                  $smarty->assign("mb_title", _tr("Validation Error"));
                  $smarty->assign("mb_message", _tr("The gender can only be
\"male\" or \"female\""));
                  return viewFormTestModule($smarty, $module_name,
$local_templates_dir);
            }
            elseif(!in_array($marital_status,array("single","widowed","marr
ied","divorced","cohabiting"))){
                  $smarty->assign("mb_title", _tr("Validation Error"));
                  $smarty->assign("mb_message", _tr("The marital status can
only be \"single\", \"widowed\", \"married\", \"divorced\" or
\"cohabiting\""));
                  return viewFormTestModule($smarty, $module_name,
$local_templates_dir);
            }
            else{
                $smarty->assign("mb_title", _tr("Message"));
                $message = _tr("The following data was entered").":<br />";
                $message .= "<b>"._tr("Name").":</b>
".htmlentities($name)."<br /><b>"._tr("Last Name").":</b>
".htmlentities($last_name)."<br /><b>"._tr("Gender").":</b>
".htmlentities($gender)."<br /><b>"._tr("Email").":</b>
".htmlentities($email)."<br /><b>"._tr("Marital Status").":</b>
".htmlentities($marital_status);
                $smarty->assign("mb_message",$message);
            }

    }
    return viewFormTestModule($smarty, $module_name, $local_templates_dir);
}

function createFieldForm()
{
    $gender = array("male" => _tr("Male"), "female" => _tr("Female"));
    $marital_status = array("single" => _tr("Single"), "widowed" =>
_tr("Widowed"), "married" => _tr("Married"), "divorced" => _tr("Divorced"),
"cohabiting" => _tr("Cohabiting"));

    $arrFields = array(
            "name"   => array("LABEL"          => _tr("Name"),
                    "REQUIRED"                 => "yes",
                    "INPUT_TYPE"               => "TEXT",
                    "INPUT_EXTRA_PARAM"        => "",
                    "VALIDATION_TYPE"          => "text",
                    "VALIDATION_EXTRA_PARAM"   => ""
                  ),
            "last_name"    => array("LABEL"      => _tr("Last Name"),
                    "REQUIRED"                   => "yes",
                    "INPUT_TYPE"                 => "TEXT",
                    "INPUT_EXTRA_PARAM"          => "",
                    "VALIDATION_TYPE"            => "text",
                    "VALIDATION_EXTRA_PARAM"     => ""
                  ),
            "gender"       => array("LABEL"      => _tr("Gender"),
                    "REQUIRED"                   => "yes",
                    "INPUT_TYPE"                 => "RADIO",
                    "INPUT_EXTRA_PARAM"          => $gender,
                    "VALIDATION_TYPE"            => "text",
                    "VALIDATION_EXTRA_PARAM"     => ""
                  ),
            "email"        => array("LABEL"      => _tr("Email"),
                    "REQUIRED"                   => "yes",
                    "INPUT_TYPE"                 => "TEXT",
                    "INPUT_EXTRA_PARAM"          => "",
                    "VALIDATION_TYPE"            => "email",
                    "VALIDATION_EXTRA_PARAM"     => ""
                  ),
         "marital_status" => array("LABEL"         => _tr("Marital Status"),
                    "REQUIRED"                   => "yes",
                    "INPUT_TYPE"                 => "SELECT",
                    "INPUT_EXTRA_PARAM"          => $marital_status,
                    "VALIDATION_TYPE"            => "text",

                    "VALIDATION_EXTRA_PARAM"     => ""
              ),
         );
  return $arrFields;
}

function getAction()
{
    if(getParameter("save"))
         return "save";
    else
         return "report";
}
?>
```



Name: *	
Last Name: *	
Gender: *	○ Male ○ Female
Email: *	
Marital Status: *	Single

Image 8.- View of the form-type module according to the code described previously

2.6 Class paloSantoGrid.class.php

This class allows us to easily manage grid type modules. It is necessary to include it in our module in case we wish to use it.

The builder for this class has the following form:

- `function paloSantoGrid($smarty)`

Where $smarty is an instance of smarty (which is passed to the function _moduleContent of our module.)

This class has the following functions:

- `function addNew($task="add", $alt="New Row", $asLink=false)`

This function allows adding an element to the grid whose functionality is of adding a new piece of data to be shown in the grill. The parameter $task will be the “name” attribute for the element, the parameter $alt is the label that will be shown for the element, and the parameter $asLink is **TRUE** if the element will be the link, otherwise it will be a submit-type input.

- `function customAction($task="task", $alt="Custom Action", $img="", $asLink=false)`

This function allows adding an element to the grid. The parameter $task will be the “name” attribute of the element, the parameter $alt is the label that will be shown for the element, and the parameter $asLink is **TRUE** if the element is a link, otherwise it will be a submit-type input and the parameter $img is the path to an image that will be the representative icon of the element.

- `function deleteList($msg="", $task="remove", $alt="Delete Selected", $asLink=false)`

This function allows adding an element to the grid whose functionality is to eliminate one or more entries shown in the grid. The element $msg will be the confirmation message that will appear, the parameter $task will be the element's “name” attribute, the $alt paramenter is the label that will be shown for the element, and the parameter $asLink is **TRUE** if the element will be a link, otherwise it will be a submit-type input.

- `function addLinkAction($href="action=add", $alt="New Row", $icon=null, $onclick=null)`

This function allows adding a link-type element to the grid. The parameter $href is where the link goes, the parameter $alt is the label that will be shown for the element, the parameter $icon is the path to an image that will be the elements representative icon, and the parameter $onclick is the element that will be linked when that link is clicked (optional).

- `function addSubmitAction($task="add", $alt="New Row", $icon=null, $onclick=null)`

This function allows adding a submit-type element to the grid. The parameter $task is the element's “name” attribute, the parameter $alt is the label that will be shown for the element, the parameter $icon is the path to an image that will be the element's representative icon, and the parameter $onclick is the event that will be linked to when clicking on that element (optional.)

- `function addButtonAction($name="add", $alt="New Row", $icon=null, $onclick="javascript:click()")`

This function allows adding a button-type element to the grid. The parameter $name is the element's “name” attribute, the parameter $alt is the label that will be shown for the element, the parameter $icon is the path to an image that will be the element's representative icon, and the parameter $onclick is the event that will be linked when clicking on that element (optional.)

- `function addInputTextAction($name_input="add", $label="New Row", $value_input="", $task="add", $onkeypress_text=null)`

This element allows adding an element to the grid with the type “input text”. The parameter $name_input is the element's “name” attribute, the parameter $label is the label that will be shown for the element, the parameter $value_input is the input's default value, the parameter $task is the action that is sent to the server when the “button” associated to the text box is pressed and the $onkeypress_text parameter is the event that will be associated with the textbox each time a key is pressed.

- **`function`** `addComboAction($name_select="cmb", $label="New Row", $data=array(), $selected=`**`null`**`, $task="add", $onchange_select=`**`null`**`)`

This function allows adding an element to the grid with the type “combo box”. The parameter $name_select is the element's “name” attribute, the parameter $label is the label that will be shown for the element, and the parameter $data is an array with the combo's data, the parameter $selected is the element selected by default, the parameter $task is the action that is sent to the server when the “button” associated to the combo is pressed and the parameter $onchange_select is the event that will be associated to the combo each time the combo's value is changed.

- **`function`** `addHTMLAction($html)`

This function allows adding a new html-type element to the grid. The parameter $html is the html code to be added.

- **`function`** `addFilterControl($msg, &$arrData, $arrFilter = array(), $always_activated=`**`false`**`)`

This function allows adding a filter controller to the grid (purple-colored messages show up when the filter is applied.) The parameter $msg is the message that will appear when the filer is applied, the parameter $arrData is the array with the data that lets us know if the filter is being applied or not, the parameter $arrFilter is an array that lets us associated two or more filters as a single one, where the array's key is the name of the filter's element and the value is the default value it will have. The parameter $always_activated will be true if we want it to always show the filter controller, otherwise it is left on false.

2.7 Class paloSantoJSON.class.php

This class is used to code in JSON format. It can be of great help when an AJAX petition is made to the server, in order to send the answer to the client in JSON format. It codes a three-element array, whose indexes are “error”, where errors are stored in case there are any; “statusResponse”, which stores the response's status, which is set to OK by default; and “message”, which stores the wanted response.

We must include this class in our module in case we want to use it.

The builder is the following:

- `function PaloSantoJSON()`

It has the following functions:

- `function createJSON()`

Codes the response in JSON with the format mentioned previously.

- `function set_error($error)`

Sets the value of the “error” index with what is contained in the parameter $error.

- `function set_status($status)`

Sets the value of the index “statusResponse” with what is contained in the parameter $status.

- `function set_message($message)`

Sets the value for the index “message” with what is contained in the parameter $message.

3. AJAX in Elastix

The Elastix Framework also counts with a JavaScript function to carry out AJAX petitions! This function is the following:

- `function request(url,arrParams, recursive, callback)`

The parameter “url” is the address to which the petition is made (it will usually be “index.php”). The parameter “arrParams” is an array that contains the parameters that the server will receive. The parameter “recursive” is a boolean, this will be **TRUE** if we want the same AJAX petition to be carried out again once the server answers and if we want to stop it at any moment, the function “callback” must return **TRUE**. Finally, the parameter “callback” is a function that will be invoked each time the server responds.

Example:

If you wish to create a JavaScript function that carries out an AJAX petition to the module “testModule”, this function must receive as parameter a text that will be sent to the server. The server must answer the text translated to the language selected on the Elastix server in case we have that translation, otherwise it will return the same text. In the client an alert with the text returned by the server must be shown.

File testModule/themes/default/js/javascript.js

```
function getTextTranslate(text)
{
      var arrAction                          = new Array();
       arrAction["menu"]            = "testModule";
            arrAction["action"]        = "translate";
       arrAction["text"]            = text;
           arrAction["rawmode"]    = "yes"; //Remember, this is necessary
because in this way the server will only response the content of the module
that will be a JSON.

     request("index.php",arrAction,false,
            function(arrData,statusResponse,error)
            {
        //The variable statusResponse contains the value assigned to
statusResponse in the JSON response.
        //The variable error contains tha value assigned to error in the
JSON response.
        //The variable arrData contains the value assigned to message in
the JSON response.
        alert(arrData);
          }
     );
}
```

Archivo testModule/index.php

```
<?php

include_once "libs/paloSantoForm.class.php";
include_once "libs/paloSantoJSON.class.php";

function _moduleContent(&$smarty, $module_name)
{
      /* Typical headers of the module like the last example.
      .....
      .....
      .....
      */

      $action = getAction();
      $content = "";

      switch($action){
           case "save":

                $content = saveTestModule($smarty, $module_name,
$local_templates_dir);
                     break;
           case "translate":
                $content = translateText();
                break;
                  default: // view_form
                     $content = viewFormTestModule($smarty, $module_name,
$local_templates_dir);
                 break;
      }
      return $content;
}

/* Here goes the functions  saveTestModule and viewFormTestModule like the
last example
....
....
....
*/

function translateText()
```

```
{
    $jsonObject = new PaloSantoJSON();
    $text   = getParameter("text");
    $translated = _tr($text);
    $jsonObject->set_message($translated);
    return $jsonObject->createJSON();
}

function getAction()
{
    if(getParameter("save"))
        return "save";
    elseif(getParameter("action") == "translate")
        return "translate";
    else
        return "report";
}
```

4. Convert a module into an addon

It is common to get confused and to think that a *module* and an *addon* are the same thing. In the end, the two generally end up becoming a new menu in Elastix. However, an *addon* is something much more complex than a *module.*

An *addon* is a software package certified by PaloSanto Solutions and that is available in RPM format through the official repository. An *addon* can contain a module, but can also contain other software components, written in any language supported by Elastix. The installation of an *addon* is carried out simply and intuitively through the “Addons” menu from the Elastix Web Interface.

That said, it is important to clear up that an Elastix module can be converted into an *addon*. To do this, it is necessary to package the module in RPM format and to start the interoperable software certification with Elastix.

4.1 Source skeleton for packaging

Usually the source will contain an XML file and two main folders. The file and the folders are respectively: “menu.xml”, “modules” and “setup”.

- The file menu.xml is an XML that contains the modules that will be integrated into Elastix, including location, name, type, permissions, etc.

<u>Example:</u>

We want to create the file menu.xml, for a parent module named “Parent Module” with id “parent_module” located under the PBX module in position 6. There will also be two other modules. The first will be “Test Module” with id “test_module” located inside “parent_module” in the first position and the module “Link Module” with id “link_module” which is the link-type module to access the server through the 8080 port. By default, only users with the administrator group can gain access to these modules.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<module>
  <menulist>
    <menuitem menuid="parent_module" desc="Parent Module"
parent="pbxconfig" module="no" link="" order="6">
      <permissions>
        <group id="1" name="administrator" desc="total access"></group>
      </permissions>
    </menuitem>
    <menuitem menuid="test_module" desc="Test Module"
parent="parent_module" module="yes" link="" order="61">
      <permissions>
        <group id="1" name="administrator" desc="total access"></group>
      </permissions>
    </menuitem>
    <menuitem menuid="link_module" desc="Link Module"
parent="parent_module" module="no" link="http://{NAME_SERVER}:8080"
order="62">
      <permissions>
        <group id="1" name="administrator" desc="total access"></group>
      </permissions>
    </menuitem>
  </menulist>
</module>
```

As can be seen, inside the label “menulist” are all the modules and each module is described with the label “menuitem” where the attribute “menuid” is the id of the module, the attribute “desc” is the label that will be shown in the web interface, the attribute “parent” is the id of the module that will contain it, the attribute “module” can be “yes” if it is an actual module or “no” if it is a parent module or if it is a link-type module, the attribute “link” is the link to which the module will lead and the module “order” is the order that the module will occupy.

In the label “permission” are detailed the user groups that will have access to the module by default.

Now the only thing to be done is to use the script elastix-menumerge giving it the XML file so that the Elastix integration is carried out. Also, if we want to remove a menu, the script elastix-menuremove is executed, giving it the menu's id. This must be carried out in the spec file.

- The folder “modules” contains all the modules included in the addon.

- The folder “setup” contains configuration files or necessary scripts for the correct functioning of a module. Also, in case of a local database being used it will count with a folder named “db.” Inside this folder there will have to be a file named “db.info” and three folders: “install”, “update” and “delete”.

The file *db.info* contains necessary information about the database. It has a header indicating the name of the database, then the keyword “ignore_backup” that will be “yes” in case you do not wish to make a backup of an existing database with that name, otherwise it will be “no.” It also has the keyword “engine” that indicates the engine used by the database (in Elastix we use “sqlite3” or “mysql”), then the keyword “path” that indicates the path where the database is located (for sqlite3 it's “/var/www/db” and for mysql it is “/var/lib/mysql”) and finally the keyword deletable which will be “yes” if one wishes to be able to eliminate the database when the package is uninstalled, otherwise it will be “no” (having this field in “yes” doesn't mean that the database will be eliminated automatically when the package is uninstalled, it simply opens that possibility, what you will need to do is place a database elimination script in the “delete” folder.)

Example:

Build a db.info file for an addon that will have a sqlite3 database named “myDBSqlite” which must have a backup in case it exists and it must not be deletable, and a mysql database named “myDBMysql” with the same characteristics.

```
[myDBSqlite]
ignore_backup = no
engine = sqlite3
path = /var/www/db
deletable = no

[myDBMysql]
ignore_backup = no
engine = mysql
path = /var/lib/mysql
deletable = no
```

In the folder "install" there will be a folder for each database **with the same name as the database**. Inside this folder will be sql scripts **that will only be executed when the package is installed** (and will be applied to the database corresponding to the name of the folder). These scripts will have the names "1_schema.sql", "2_schema.sql", "3_schema.sql", etc (usually you will only need one), the prefix number is important because it indicates the execution order. For "sqlite3" databases, tables are created and entries are added by default (in case they're necessary) but for mysql databases it is also necessary to create the database (CREATE DATABASE dbname;) followed by a USE dbname; so that, from there, one can continue creating tables and the rest.

In the folder "update" there will be a folder for each database **with the same name as the database**. Inside each of these folders will be another folder named "version_sql" that will contain the update sql scripts. **The name of these scripts is very important**, they must have the following structure:

#number_#lastVersion_#newVersion.sql

Where:

#number is the order of execution for the script
#lastVersion is the last existing version.
#newVersion is the new version to be launched
This script will only be executed in versions lower than *#newVersion*.

Example:
What will be the name for update sql script for:

1. An update script that must be the first to be executed and in its moment the last launched version was 2.2.0-2, only versions lower than 2.2.0-3 must be executed.
2. Another update script that is the second to be executed and in its moment the last launched version was 2.2.0-6, only versions lower than 2.2.0-7 must be executed.

For the first script it would be: *1_2.2.0-2_2.2.0-3.sql*
For the second script it would be: *2_2.2.0-6_2.2.0-7.sql*

Note that it will always be *#lastVersion* < *#newVersion*, also, the execution order for the scripts goes hand in hand with the versions, that is the higher the order-number the higher the versions.

In an installation, apart from executing the scripts from the “install” folder, **all** the scripts in the “update” folder are executed.

The “delete” folder also contains folders with the same name as the database, which will contain sql scripts for the uninstallation of the database with the names “1_dbname.sql”, “2_dbname.sql”, “3_dbname.sql”, etc.

It is recommended no to delete the databases when uninstalling the package.

The script in charge of reading and executing these files is “elastix-dbprocess” that must be invoked in the spec file.

4.2 Spec file

Once the source is built, it is a question of creating the spec file, to finish with the RPM building or packaging process.

Below we show a small example from a spec file without going into more details, since it is assumed that the developer has clear knowledge in regards to the building of RPM packages.

```
%define modname example

Summary: Elastix Module Example
Name:    elastix-%{modname}
Version: 2.2.0
Release: 1
License: GPL
Group:   Applications/System
Source0: %{modname}_%{version}-%{release}.tgz
BuildRoot: %{_tmppath}/%{name}-%{version}-root
BuildArch: noarch
Prereq: elastix-framework >= 2.2.0-25

%description
Elastix Module Example

%prep
%setup -n %{modname}
```

```
%install
rm -rf $RPM_BUILD_ROOT

# Files provided by all Elastix modules
mkdir -p    $RPM_BUILD_ROOT/var/www/html/
mv modules/ $RPM_BUILD_ROOT/var/www/html/

# The following folder should contain all the data that is required by the
installer,
# that cannot be handled by RPM.
mkdir -p    $RPM_BUILD_ROOT/usr/share/elastix/module_installer/%{name}-
%{version}-%{release}/
mv setup/   $RPM_BUILD_ROOT/usr/share/elastix/module_installer/%{name}-
%{version}-%{release}/
mv menu.xml $RPM_BUILD_ROOT/usr/share/elastix/module_installer/%{name}-
%{version}-%{release}/


%pre
mkdir -p /usr/share/elastix/module_installer/%{name}-%{version}-%{release}/
touch /usr/share/elastix/module_installer/%{name}-%{version}-
%{release}/preversion_%{modname}.info
if [ $1 -eq 2 ]; then
    rpm -q --queryformat='%{VERSION}-%{RELEASE}' %{name} >
/usr/share/elastix/module_installer/%{name}-%{version}-
%{release}/preversion_%{modname}.info
fi

%post
pathModule="/usr/share/elastix/module_installer/%{name}-%{version}-
%{release}"

# Run installer script to fix up ACLs and add module to Elastix menus.
elastix-menumerge $pathModule/menu.xml

pathSQLiteDB="/var/www/db"
mkdir -p $pathSQLiteDB

preversion=`cat $pathModule/preversion_%{modname}.info`

if [ $1 -eq 1 ]; then #install
  # The installer database
    elastix-dbprocess "install" "$pathModule/setup/db"
elif [ $1 -eq 2 ]; then #update
    elastix-dbprocess "update"  "$pathModule/setup/db" "$preversion"
fi

%clean
```

```
rm -rf $RPM_BUILD_ROOT

%preun
if [ $1 -eq 0 ] ; then # Validation for desinstall this rpm
  echo "Delete example menus"
  elastix-menuremove "%{modname}"

  # Here you should call to elastix-dbprocess for deleting, the same way
that it was for  install, just that instead of word "install" goes word
"delete". But this is not often used due to the databases usually are not
deleted
fi

%files
%defattr(-, asterisk, asterisk)
%{_localstatedir}/www/html/*
/usr/share/elastix/module_installer/*

%changelog
* Mon Jan 30 2012 Alberto Santos <asantos@palosanto.com> 2.2.0-1
   -  Initial version.
```

4.3 Finalization of the certification process.

Once the RPM is created, one proceeds with signing the Elastix Software Certification agreement and sending the rpm package to PaloSanto Solutions to the email address addons@palosanto.com with a copy to asantos@palosanto.com, along with a users manual in order to continue with its certification revisions. Once this is approved, it will go on to become a part of the Elastix repositories and will be turned into an Elastix Addon.

If you have any doubts about the certification process or any other aspect of the Elastix Framework, you can send an email to any of the addresses mentioned previously or visit our web page at http://addons.elastix.org