A Chisimba MVC Hello World

This chapter assumes that you have:

- an installed and working Chisimba installation on your computer;
- knowledge of PHP
- read the chapter on Module Catalogue, and carried out the activities in it.

The hello world

- A "hello world" program is a computer program that prints out "Hello, World!" on a display device. It is used in many introductory tutorials for teaching a programming language. Such a program is typically one of the simplest programs possible in a computer language. In this case, we will use it as a very simple example to ensure that you understand the MVC approach, and the minimum structure for a Chisimba module. Once you understand this structure, you can apply your existing skills to Chisimba very easily.

The MVC

- The M in MVC refers to the model, which is typically the *data access layer*. In this case, we are not going to access any external data yet, so we will ignore the model, and concentrate on the *view* and the *controller*.
- Create a directory called *hellochisimba* (or hellowhatever) in your chisimba_modules directory:

  /path/to/chisimba/chisimba_modules/hellochisimba, where */path/to/chisimba/* is the place where your chisimba files are located, for example /var/www/

The security check

Create a text file in that directory and name it controller.php. Open it in your favorite text editor (or IDE) and enter the controller code as follows:

```
<?php
// security check - must be included in all scripts
if (!
$GLOBALS['kewl_entry_point_run']){
die ("You cannot view this page
directly");
}
```

The comments

The next set of lines in your controller is the PHPDocumentor style comments for the controller class.

```
/**
*
  *  Controller class for
  hellochisimba module.  This class
*  greets the user with a hello
message. This is a demonstration
```

Class definition

```
class  hellochisimba extends
controller
{
 }
```

- Note that the class has the same name as the module and the directory that it is in, and must extend the

The init()

- All Chisimba classes that extend the framework have a constructor that is named *init()*, so creating that constructor is the next bit of code needed.
- A common use of the constructor is to set up default values for object properties, and in this case we will just use it to set up the default value for our output string and store it in a property that we will call $greeting.

Init() Code

```
public  function init()
      {
      $this ->greeting = "Hello there
Chisimba user. Welcome to my first
module";
      }
```

The dispatch()

- A controller must have a *dispatch()* method that is invoked by the engine to process the logic of the controller.

```
public  function dispatch()
{

}
```

Dispatch code

In this case, we are going to assign the value of the $greeting property of the class to a variable so that it is available to the template (the view layer) for display.

Variables created in the controller are copied to the template using the *setVar()* method, or are passed by reference using the *setVarByRef()* method, which takes two parameters as strings, the variable name and the variable value. So, add the following line to the dispatch method:

setVarByRef

```
$this ->setVar('greet', $this->greeting);
```

- The variable $greet will thus be available to your template, and will have the value that was assigned to $this->greeting in the init method.

- The next thing the dispatch method does is hand over to the template. In a simple controller, it may do this directly, but in one with more processing, it may hand over to another method to call the template. Here we will call the template from within the controller

Passing control to template

- Let us assume that the template is called *default_tpl.php*, noting that templates are called name_tpl.php, where name is the name of the template. The template is parsed by returning it as a string. So we enter:

```
return "default_tpl.php";
```

as the last line of our dispatch method.

The template

In Chisimba, templates are just PHP files which may be supported by display classes that provide for the formatting of the output.

Create a templates directory under the hellochisimba directory, and under that directory create a content directory. The directory structure is thus

/path/to/chisimba/chisimba_modules/hellochisimba/

Navigate to that directory, and create a file called default_tpl.php.

This file should contain only:

register.conf

- We have one more step to perform, and that is that we have to make our module visible to the Module catalogue for installation. To do this, we need to create a register.conf file in the module's top-level directory.
- Create a file called *register.conf* in */path/to/chisimba/chisimba_modules/hellochisimba/* and add the following code to it:

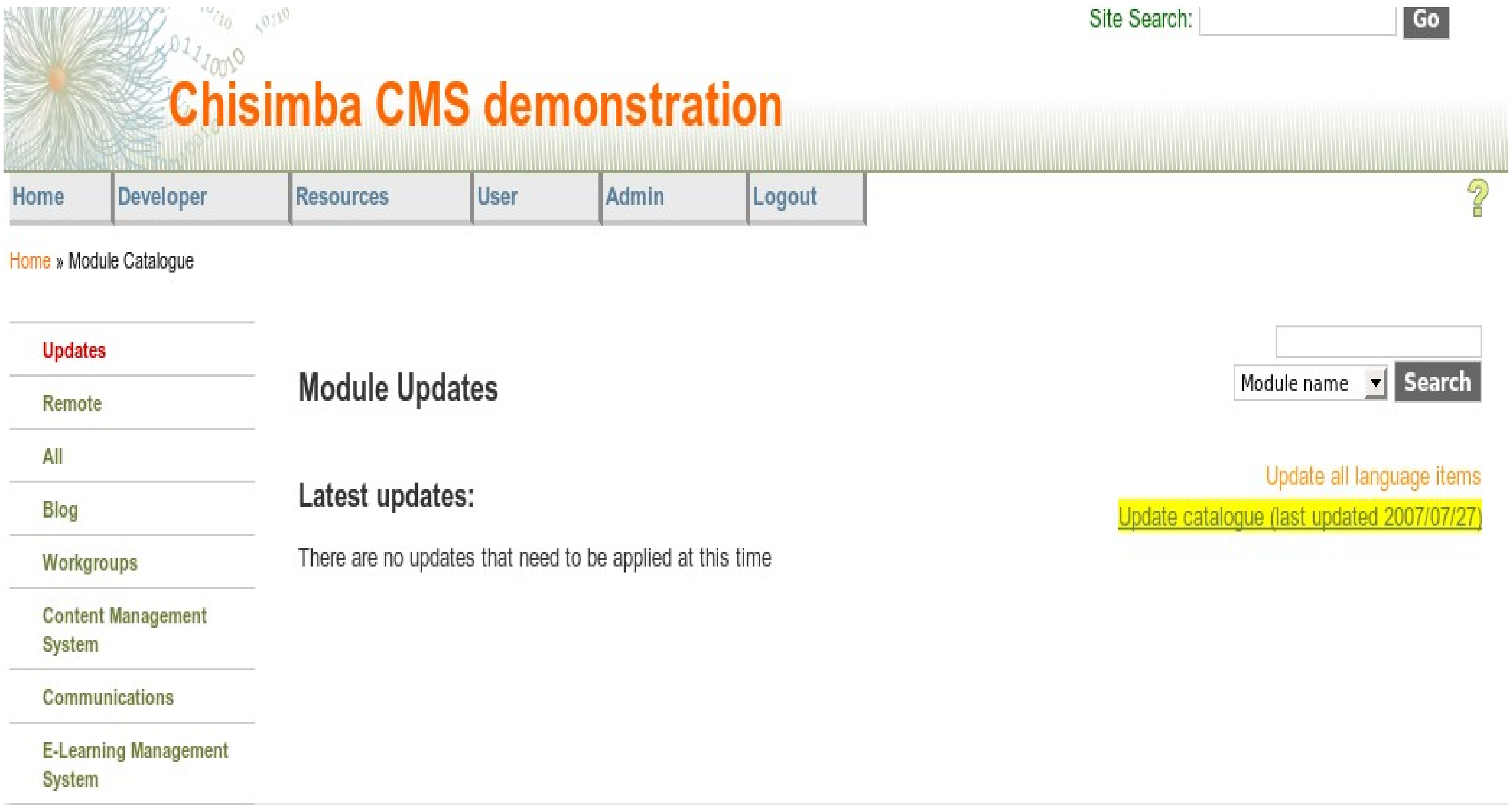
register.conf content

```
MODULE_ID: hellochisimbaMODULE_NAME: Hello Chisimba

MODULE_DESCRIPTION: A demonstration module to show the MVC architecture

MODULE_AUTHORS: Yourname Here

MODULE_RELEASEDATE: 2007 01 01

MODULE_VERSION: 0.1

MODULE_PATH: hellochisimba

MODULE_ISADMIN: 0
```

Installing the module

- The next step is to install your module. Open your chisimba installation in a web browser, and login as a user with administrative rights. If you still have your default Chisimba installation, the default user with these rights is “admin”, with the password “a” (be sure to change it).
- On the Admin menu, locate Module Catalogue, and open module catalogue. In module catalogue, you will find a link (highlighted in yellow, as shown below) that says Update catalogue. Click that link to be sure that your

Module install screen

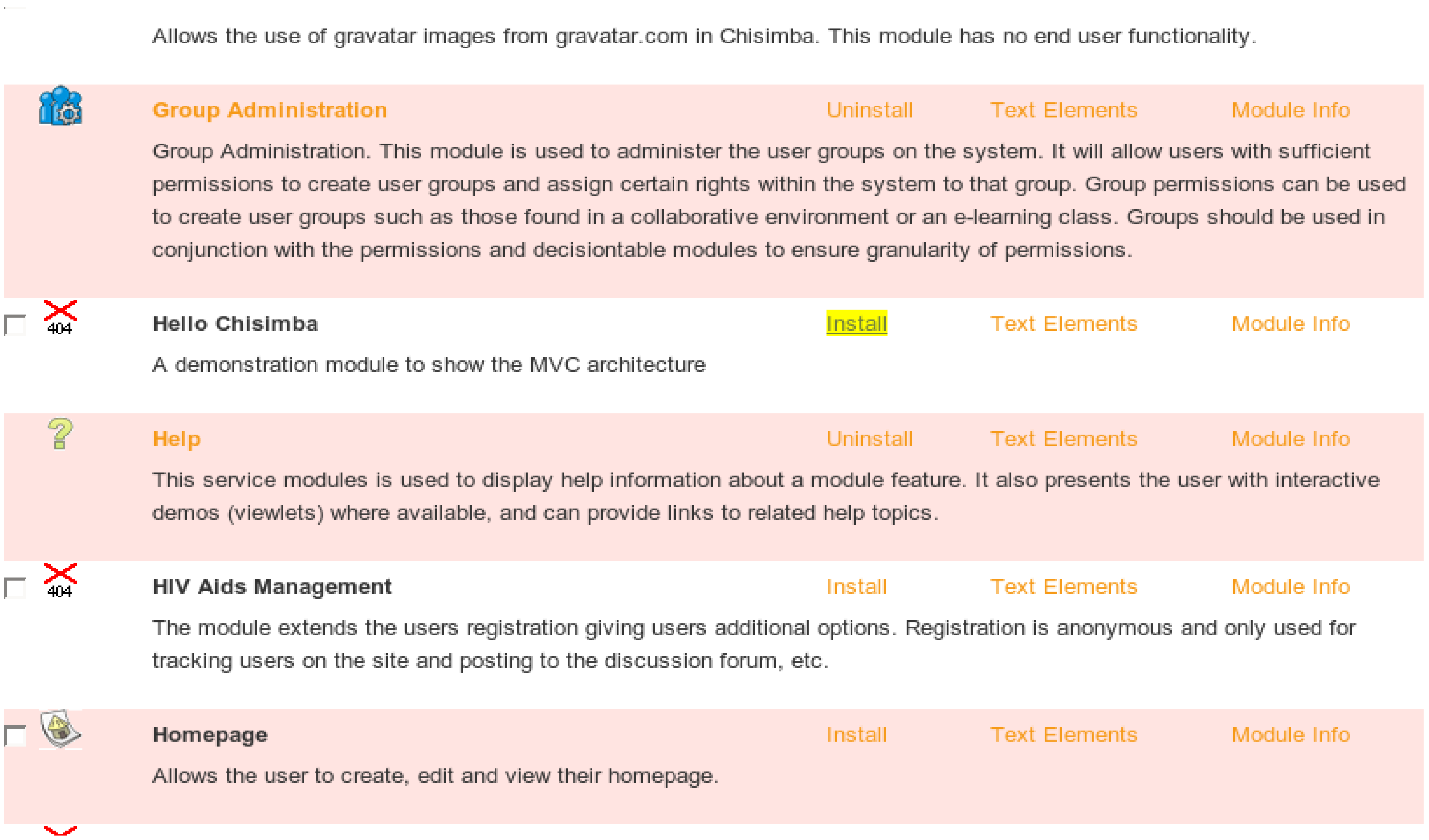

Site Search:
Go
Chisimba CMS demonstration
Home
Developer
Resources
User
Admin
Logout
?
Home » Module Catalogue
Updates
Remote
All
Blog
Workgroups
Content Management System
Communications
E-Learning Management System
Module name
Search
Update all language items
Update catalogue (last updated 2007/07/27)

Module Updates

Latest updates:

There are no updates that need to be applied at this time

Click All on the menu at the left side (the third item from the top as shown above). You will see a list of available local modules as shown below. Your module is called Hello Chisimba, as per your *register.conf* file.

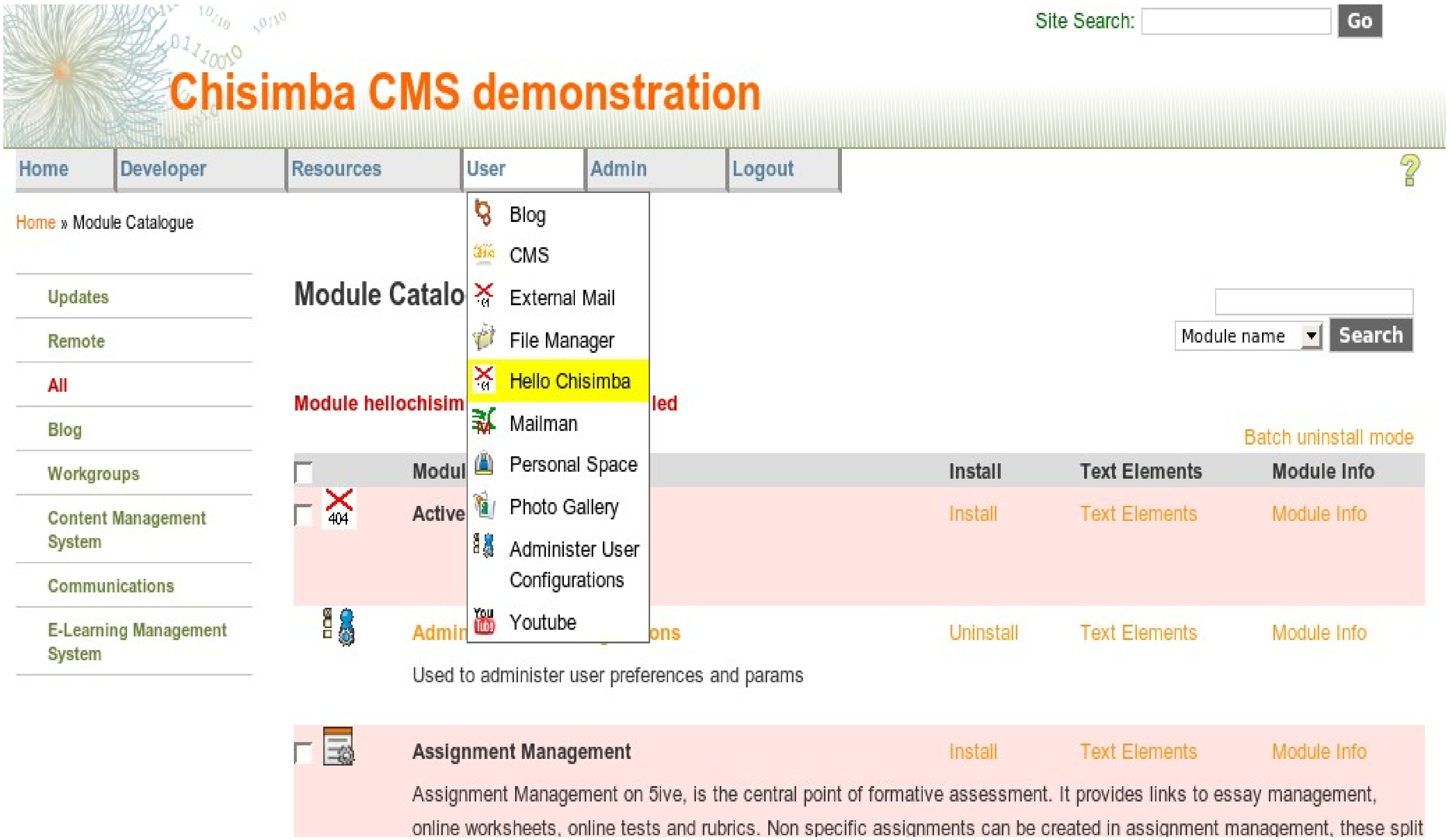


Allows the use of gravatar images from gravatar.com in Chisimba. This module has no end user functionality.

Group Administration	Uninstall	Text Elements	Module Info
Group Administration. This module is used to administer the user groups on the system. It will allow users with sufficient permissions to create user groups and assign certain rights within the system to that group. Group permissions can be used to create user groups such as those found in a collaborative environment or an e-learning class. Groups should be used in conjunction with the permissions and decisiontable modules to ensure granularity of permissions.			
Hello Chisimba	Install	Text Elements	Module Info
A demonstration module to show the MVC architecture			
Help	Uninstall	Text Elements	Module Info
This service modules is used to display help information about a module feature. It also presents the user with interactive demos (viewlets) where available, and can provide links to related help topics.			
HIV Aids Management	Install	Text Elements	Module Info
The module extends the users registration giving users additional options. Registration is anonymous and only used for tracking users on the site and posting to the discussion forum, etc.			
Homepage	Install	Text Elements	Module Info
Allows the user to create, edit and view their homepage.			

Module install cnt'd

- Click on the Install link, highlighted in yellow above to install the module. Since we gave it a menu category of user in register.conf (MENU_CATEGORY: user), the module will appear on the User menu of the Chisimba toolbar as shown below, where it is called Hello Chisimba. Note that there is no module icon at this stage.



Chisimba CMS demonstration

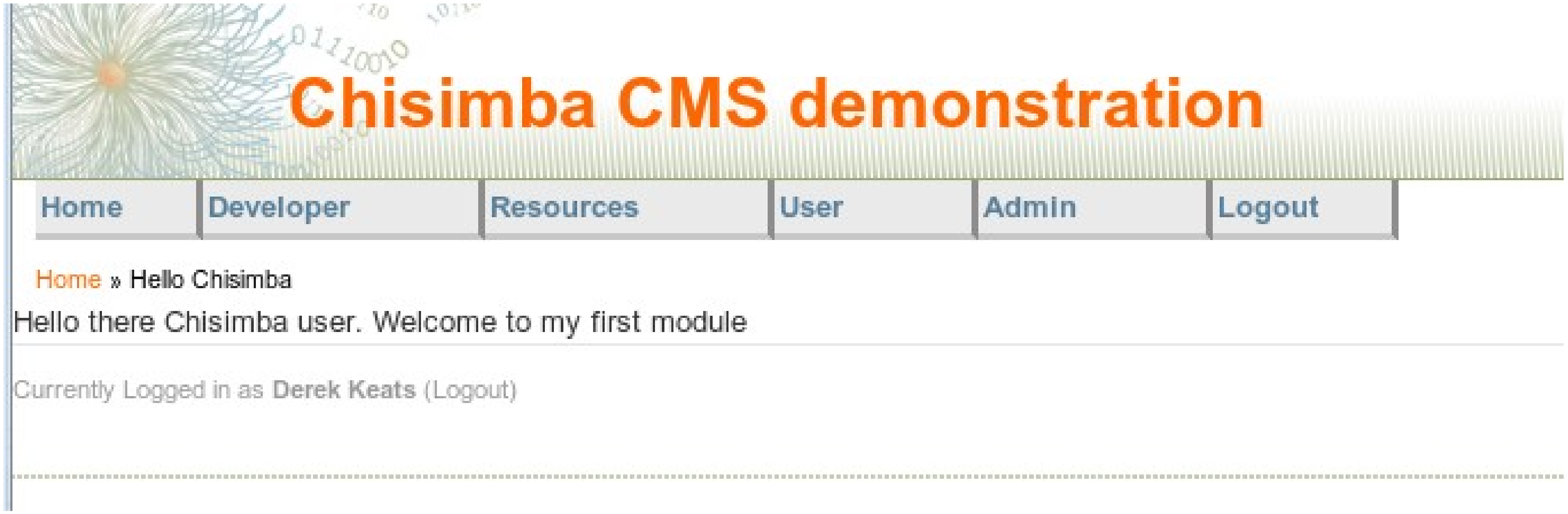
Home » Module Catalogue

Module Catalo

Module hellochisim led

		Modul	Install	Text Elements	Module Info
		Active	Install	Text Elements	Module Info
		Admin ons Used to administer user preferences and params	Uninstall	Text Elements	Module Info
		Assignment Management Assignment Management on 5ive, is the central point of formative assessment. It provides links to essay management, online worksheets, online tests and rubrics. Non specific assignments can be created in assignment management, these split	Install	Text Elements	Module Info

- Selecting Hello Chisimba from the menu, as highlighted above, opens the module, and your content is rendered to the screen in a Chisimba interface as shown below.



Hello there Chisimba user. Welcome to my first module

Currently Logged in as **Derek Keats** (Logout)

Making your text localizable

- In the previous example, we had text that was hard coded in the controller code. This code would never be accepted into a real Chisimba module because it is not multi-lingualized. Therefore, this section addresses multi-lingualization of code using the Chisimba framework.
- The first step in multi-lingualization is to define the text codes in the *register.conf* file located in the module's directory. The *register.conf* file provides for two methods of adding text strings

Convention

The convention is that the variable is named in three parts *mod_modulecode_identifier*, where mod indicates that it belongs to a module, *modulecode* indicates the module that owns it, and *identifier* is any combination of meaningful character that can identify the string. A *description* is provided to help translators, and is separated from the text string, containing what will display on the interface, by a pipe (|) character.

Thus, for our example, the register.conf file would

Using language code

To make this work, we need to introduce and add the framework language object, which comes from a helper module called language. First, add a property to hold the object:

```
/**
*  @var $objLanguage String object
property for holding the
*language object
```

Instantiate language object

- Then, instantiate the language object in our *init()* method of the controller. Note, that we instantiate it here so it is available to other methods, which we will add later. In most cases, we would instantiate the class close to where we use it. Insert the code below in the *init()* method of the controller.

//Instantiate the language object

```
$this->objLanguage =
```

New init()

The init() method now contains the following:

```
  public  function init()
  {
      //Instantiate the language object
  $this ->objLanguage = $this-
>getObject('language', 'language');
      //Assign the value of the greeting
  to the language element
  mod hellochisimba greeting
```

Language item not found

Now if you go to your hellochisimba module, and select it from the menu, you will see that the interface displays:

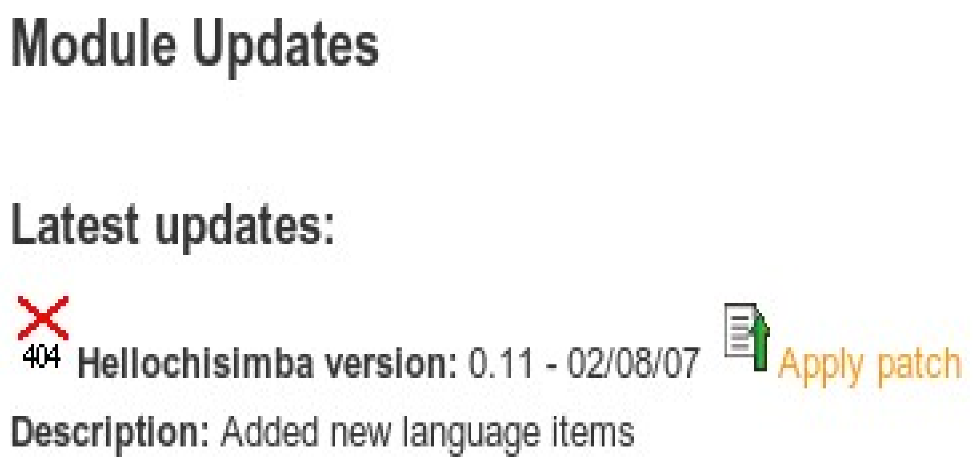
> **Language item not found: mod_hellochisimba_greeting from hellochisimba**

The reason for this is because you have effectively made a patch, but you have not run the patch through module catalogue. Therefore, you

Increasing version number

- This increases the version number so that module catalogue can find it, and automatically apply your language patch.
- If you increase the final digit each time you add language elements, you will make sure that you never break the interface with missing language elements. Thus, patching interface elements is very simple in Chisimba.

Applying patch



Module Updates

Latest updates:

Hellochisimba version: 0.11 - 02/08/07

Description: Added new language items

code2txt()

- Another method allows the replacement of one or more codes in a string with a variable name. This method is code2txt(), and it helps ensure that translations containing variables are rendered in the correct syntax for the translation language.
- Add the following line from to your register.conf file, increase the version number and save it. Go to module catalogue and apply the patch.

Using code2txt()

```
TEXT: mod_hellochisimba_helloperson|The
text that the helloChisimba module renders
to say hello to a particular user by their
name|Hello there [-FIRSTNAME-], you
interesting person whose username is [-
USERNAME-].
```

- Note the convention for including the variables, enclosed in [- and -], and in capital letters (actually, they are not case sensitive, but it is easier to see them if they are in capitals stick with the convention).

Using code2txt()

To see the benefit of this we need to introduce another framework object, the user object. Language and user are probably the two most often used objects in the framework's helper modules.

Rather than cluttering up the init() method with this kind of thing, let us create a method to parse this language element. Into that method, add the following line:

User is another framework element that comes from the security module. It provides a set of properties and methods for working with basic user information. Add the following lines to the method:

```
// Get the first name of the logged in user
$firstName = $objUser->getFirstName();
```

// Get the user name of the logged in user

Then we can parse the language item to replace the codes with those values. The code to do that is shown:

```
  $rep   = array(
 'FIRSTNAME'   => $firstName,
 'USERNAME'   => $userName);
```

//Return the string with the codes replaced by the values

First we create an array with indexes corresponding to the codes to be replaced, and assign them the values of $firstName and $userName. This array is then passed as a parameter to the code2Txt method of the language object.

Now our method contains the following code:

Now our method contains the following code:

```
  public  function greetUser()
  {
          // Get the user object
     $objUser  =  $this-
>getObject('user', 'security');
     // Get the first name of the
logged in user
```

Now it just remains to change our dispatch method to include:

```
$this ->greeting = $this-
>greetUser();
```

Reloading the Hello Chisimba now should produce:

Summary

Thus in this chapter we have introduced some key concepts of Chisimba programming, including

- the files that make up the basic MVC pattern, and where they are located in a module
- how to create a module
- how to create a register.conf for a module
- how to install the module
- how to make it multi-lingual
- how to add and patch language items by