Vol

CCS Developer

OCTOBER 2004

VOLUME NO. 1 ISSUE NO. 3

DataObjx L.L.C.

Website: www.dataobjx.net

DataObjx L.L.C.
Meriden, CT USA

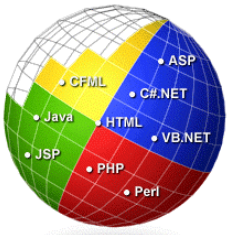
Office: 860/919-2536

Email: administrator@dataobjx.net

Website: www.dataobjx.net

DataObjx L.L.C.

Extra: Building A Document Management System - Part 1 for PHP Included In This Issue.



Need More Answers?
Click here to access the Code Charge Studio Forums

In This Issue

Dynamic Sub Menus - In this article we examine a technique that will enable you to rapidly create and integrate a sub-menu system for the various sections of your website. This method allows for dynamic link generation and allows you to create as many separate sub menus as you want.

Building A Document Management System Part 1 – PHP - Included in this issue is the conversion of "Part 1" of the Document Management System. Translated By Sixto Santos. If you're a PHP developer, download Issue #1 in conjunction with this article and code.

Dynamically Formatting Row Colors – PHP -In this article the technique used to dynamically alter the color of a row (php) is discussed. Read this article in conjunction with Issue #1.

Producing Excel Spreadsheet Using The PHP Common Interface -In this article Fernando Sibaja explains how you can produce excel spreadsheets for any table or query using the PHP Common Interface.

Building A Document Management System – Part 3 -We wrap up our three (3) part series on building a document management system. Together we'll integrate Check-In and Check-Out capability plus the ability to un-do a check out. Core document management functionality is now at your disposal.

Editors Comments

Viewpoint...

In our third (3rd) issue we complete the core functionality for our Document Management System.

We also demonstrate a technique that will allow you to build menus dynamically for your 'sub-pages'. A method that you will find useful in a number of situations.

Encapsulated within both of these articles are various methods and techniques that will make building your next application much easier.

We also have a number of PHP articles contributed by Fernando Sibaja and Sixto Santos. Both of these developers have years of experience and share some of their knowledge with you by providing you with real world examples.

Furthermore, Sixto Santos has translated Part 1 of the Document Management System – so PHP developers – take note.

In addition, we really need your help to keep this magazine relevant to the types of applications that you're building today.

We cannot stress how important your feed-back is to us and we encourage you to write to administrator@dataobjx.net with your suggestions.

Once again, we're asking for CodeCharge Studio developers within the community to contribute to this initiative by writing an article for CCS Developer Magazine. This is very important to us because and to you – our readers.

We hope you enjoy issue #3 of CCS Developer Magazine.

Best Regards,

Martin Hamilton
www.dataobjx.net

Expanding The CodeCharge Studio Portal – Part 2

Building Navigational Menus With CCS.

Language: .ASP
Database: N/A

Finding A Menu

When we build applications with CodeCharge Studio, we often find ourselves presented with an ever-growing number of menu options.

Some developers choose to implement a menu system immediately while others tend to wait until they know what the menu options will be in the end.

Regardless of whether you fall into the first group, the second group or somewhere in between you still find that you need to integrate 'sub-level' navigation into certain areas of your application.

It's not always practicable to implement a high-end menu system in every minor section either and often, these menus are java script driven and have hard coded values – while you likely require dynamically generated links.

For example, let's say that you have a medium to large application. For our example, let's say that it's an integrated package that contains within it – a mini email type package.

The email package would have additional links relating only to it, such as In Box, Sent Items, etc.

You want to keep these links relative to the page the user is on (the email page) and therefore do not want to place such links on the primary menu system.

You want them to be encapsulated and available only when the user is within the email package of your application.

We're going to show you a technique you can use that will enable you to produce a relatively dynamic menu system.

The menu will dynamically light up the menu option relating to the page you're on, and can be graphically enhanced by adjusting the CSS styles.

Putting Things Into Context

Open the zip file that accompanies this article and copy all of the menu_test... files and the menu_1_styles.css file into the project directory of your choice.

If the project file is already open, you may need to perform a right click on the top node of CCS Page Tree and execute a refresh of the project files.

New Cascading Style Sheet (CSS) File

Notice that the style sheet file is called menu_1_styles.css (plural). This file contains the CSS styles used by the demonstration. Refer to Figure One (1).

Some web sites have slightly different colors/branding depending on which area of the site a visitor is on.

The technique demonstrated in this article allows you to use a different style sheet for each menu. Simply copy and save the menu_1_styles.css file with another name and modify your menu's CSS file reference accordingly.

The menu_1_styles.css file contains a number of additional CSS definitions that will make your menu more attractive, but feel free to add additional attributes such as background-images, mouse_over code and so forth.

You may be surprised at the number of attributes being set by the CSS definitions, but this is necessary if we want browsers other than IE to render the page properly.

The CSS in this file has been tested under IE 6.0, FireFox 0.10.1 and Netscape 7.1 and appears to render similarly – at least for these versions.

Generate and publish all of the files, then open your browser to your project and open the menu_test.asp page.

Click on each of the menus. You should see each menu options toggle/change color and linking dynamically as you click on each page.

Before We Get Under The Hood

Before we begin to look at the code, it's important to realize that this type of menu system is certainly not applicable to every situation. Furthermore, it's important to understand that the menu does not offer many of the slick features found in many of the more powerful java script menus.

However, in some instances where a 'finite set of options' are available for a 'given' page, this simple yet effective method is often more than ample.

Since the application that you build may have more than one page requiring a sub-menu (each with their own relative links), we're going to encapsulate all of the functionality that's needed into an include file. This makes it easy to modify and maintain each menu and provides an easy mechanism by which menu's are kept separated.

Finally, notice how we maintained a modular approach to the menu. The entire logic and code necessary for the menu to work is encapsulated in the menu file itself.

The remaining files simply have the menu file inserted as an include file.

This makes this code *very* transportable and enables you to utilize this technique rapidly in any application that you build.

Once you've worked your way through the article and code, you'll begin to see spots where you can immediate apply the technique in your own applications.

Creating The Menu Include

We begin by creating a new page with the includable property set to yes. Since we may have more than one menu in our application, we should adopt some naming convention that will enable us to rapidly locate such menu files. We chose to use "menu_" as the prefix for any such menu file.

When you create your own menu, you might save this new page as menu_YourWebSiteArea.

In this example however, the menu page itself is called "menu_test_menu.ccp".

Open this file

Expanding The CodeCharge Studio Portal – Part 2

Building Navigational Menus With CCS, continued…

Language: .ASP
Database: N/A

Since we need to add label controls to the page, CCS tells us that we need to create a record/form and prompts us to generate the necessary code. We decided to call the record - 'menu'.

Click on the HTML tab and you'll notice that we've modified the HTML for the table slightly, because we are going to dynamically create the "<td>…</td>" blocks of code in the before_show event for the record.

The benefit of doing the "<td>…</td>" blocks this way is that we are able to programmatically control the number of columns with code rather than hard coding the columns into a HTML table simply by adding another label and plugging in the necessary code.

While we're on the HTML tab, also notice that the menu_1_styles.css page has been included. This means that each menu_pagename can use it's own menu_websitearea_styles.css file.

If we didn't put the style sheet inside the menu itself, you'd be forced to add the CSS File link in every page using the menu.

This is not a good idea, keep things modularized as much as you can. That way, you create more and more modular objx's (sorry 'objects') that you can use over and over again in your projects and applications.

How Does The Menu Work?

Essentially, whenever we need another menu option to our menu, we'll simply add another label control and set its content type to HTML.

Click on the first label {lnkHome} in our example page. Select the Events tab and perform a right click – show code on the before_show event for the label.

Now scroll all of the way up to the top of the _events.asp page. Notice that there are two global variables dimensioned and there is a function called 'GetPageName()'.

This is the controlling function for the dynamic menu, so we need to understand its function and purpose. Refer To Figure two (2).

Figure 1

```
*{
padding:0;
border-style:0;
margin:0;
}
.MenuSelected{
	border-left-width : 1;
	border-bottom-width : 1;
	border-right-width : 1;
	border-top-width : 1;
	border-left-style : solid;
	border-bottom-style : solid;
	border-right-style : solid;
	border-top-style : solid;
	border-left-color : #C0C0C0;
	border-bottom-color : #000000;
	border-right-color : #000000;
	border-top-color : #C0C0C0;
	padding-left : 2;
	padding-bottom : 2;
	padding-right : 1;
	padding-top : 1;
	margin-left : 1;
	margin-bottom : 1;
	margin-right : 1;
	margin-top : 1;
	color: white;
	background-color: blue;
	font-size: 11px;
	font-weight: bold;
	font-family : Verdana, Arial, Helvetica, sans-serif;
}
.MenuNormal{
	border-left-width : 1;
	border-bottom-width : 1;
	border-right-width : 1;
	border-top-width : 1;
	border-left-style : solid;
	border-bottom-style : solid;
	border-right-style : solid;
	border-top-style : solid;
	border-left-color : #C0C0C0;
	border-bottom-color : #000000;
	border-right-color : #000000;
	border-top-color : #C0C0C0;
	padding-left : 2;
	padding-bottom : 2;
	padding-right : 1;
	padding-top : 1;
	margin-left : 1;
	margin-bottom : 1;
	margin-right : 1;
	margin-top : 1;
	color: #000000;
	background-color: #C0C0C0;
	font-size: 11px;
	font-weight: bold;
	font-family : Verdana, Arial, Helvetica, sans-serif;}

.MenuSubCaptionTD{
	border-left-width : 1;
	border-bottom-width : 1;
	border-right-width : 1;
	border-top-width : 1;
	font-weight: bold;
	font-family : Verdana, Arial, Helvetica, sans-serif;}  CONTINUED IN CSS FILE.....
```

The GetPageName() Function

The purpose of the GetPageName function is, as you may have guessed – to programmatically let you know what page is being executed.

When the function executes, it

Expanding The CodeCharge Studio Portal – Part 3

Building Navigational Menus With CCS, continued…

Language: .ASP
Database: N/A

populates the two (2) global variables with the appropriate path and name information.

NOTE: This function may need further tweaking if you have a deeply nested directory structure.

To see the function in action, remove the apostrophes from the two (2) response.write statements in the menu_test_menu_BeforeShow() event.

Doing so will cause the same information that is being populated to the global variables to be displayed.

Now that we know how the menu derives the name of the page that the user has accessed, the rest is simply logic and CSS.

How Does The Menu Change Like That?

The effect that we want to achieve here is to cause the "Link" to be a href link if the user is not on the page for that action. In other words, if they're on the home page, the menu should display the menu link as a label. When we navigate off of the Home page, the menu should display the link to the home page as a link.

The logic therefore is relatively simple. If the name of the page returned by the GetPageName() function is the same as the page that the user is currently on, then we need to display the menu option as a label. If the page names do not match we need to display the menu option with a corresponding link.

Generating Menu Links Dynamically

Figure three (3) demonstrates the code used to create the menu links dynamically.

We've also included in the example the way that you would use to include any querystring parameters that are

Figure 2 - Function GetPageName()

```
<%
Dim szPath
Dim szPage

Public Function GetPageName(zPath, zPage)
'On Error Resume Next
Dim strCurrentPageName
Dim pos
  strCurrentPageName = Request.ServerVariables("SCRIPT_NAME")
  pos = InStrRev(strCurrentPageName, "/")
  zPath = UCase(Mid(strCurrentPageName, 1 ,pos))
  zPage = UCase(Mid(strCurrentPageName, pos+1 ,Len(strCurrentPageName)))
  zPage = Replace(zPage, "/", "")
End Function
%>
```

Figure 3 - labelname_BeforeShow()

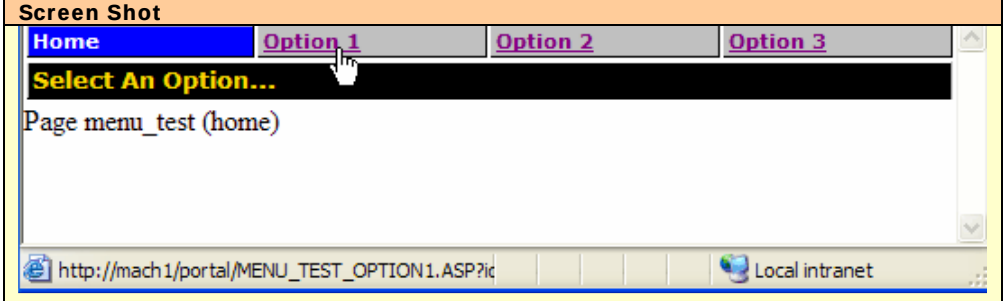
```
Dim sPageNameToMatch 'The Name Of the Page To Match To The Global szPage variable
Dim ID 'Example querystring param being passed to this page
Dim sHREF 'holds the dynamic href string
Dim sResult 'the final result to be placed in label.value
Dim sTitle 'the title to be displayed for the menu option
Dim startTD 'the starting <td> tag
Dim EndTD 'the ending </td> tag
Dim sCSSClass 'the name of the class to be used – dynamic, based on page name
Dim sCaption 'a sub caption to display additional instructions for information

        sPageNameToMatch = UCase("menu_test.asp")
        sTitle          = "Home"
        ID                      = Request.Querystring("id")
        sCSSClass       = "MenuNormal"

        If szPage = sPageNameToMatch Then
                sCSSClass       = "MenuSelected"
                sHREF           = sTitle
                sCaption = "Select An Option..."
                menu_test_menu.menu.lblCaption.value = sCaption
        Else
                sHREF           = "<a href=""" & sPageNameToMatch & "?id=" & ID & """>" & sTitle & "</a>"
        End If

        startTD = "<td class=""" & sCSSClass & """ width=""5%"" nowrap align=""left"" valign=""top"">"
        endTD   = "</td>"
        sResult = startTD & sHREF & endTD

        menu_test_menu.menu.lnkHome.value = sResult
```

Expanding The CodeCharge Studio Portal – Part 2

Building Navigational Menus With CCS, continued…

Language: .ASP
Database: N/A

Screen Shot



When you click on a link, the applicable page will be loaded. As the page loads, a check is made to determine whether the name of the .asp page is matches the name of the page required by the before_show event of the label.

If they match, a title is displayed. If they don't match, an appropriate link is displayed.

being sent to the page. This example dimensions and uses the "ID" variable. You would rename this according to the variable being passed to your page and if more variables were required you would simple dimension them and add them to the 'sHREF' string.

The important variables are;

- sPageNameToMatch,
- sTitle,
- sCaption and
- sCSSClass

The sPageNameToMatch variable is probably the most important. Since we're looking at the before_show event of {lnkHome}, you'll notice that the variable sPageNameToMatch is the name of the "Home Page" or menu_test.asp.

Quickly scan the before_show events for the other labels and you'll notice that in each case, the name of the page that the link would otherwise access is the name of the page the label represents.

So, there's no mystery here. If you're on the page that the label represents, the function displays sTitle as well, a Title - and if you're on another page (and thus the page name you're on doesn't match the sPageNameToMatch variable for the label, you'll get a href instead.

The caption should reflect whatever additional information you want to display to the user.

And finally, the sCCSClass will dynamically cause the .menunormal or .menuselected style to be displayed.

Some of you may also have noticed that there is an additional definition at the top of the style sheet. Refer to Figure five (5).

This is a CSS hack that causes virtually all browsers to properly align themselves. Using this in your CSS files will assist you in achieving a consistent appearance regardless of the browser your visitor is using.

Now that you know how to easily implement a basic dynamic menu into your applications, you'll find plenty of use for this technique.

This method is useful for both production and is especially useful if you need to have a simple form of navigation while you're building your application, even if you intend to swap it out later should you find you need something more. Quite often,

Figure 5 – A Special CSS Tag

```
*{
padding:0;
border-style:0;
margin:0;
}

This is an important CSS tag
that you should automatically
include in all of you style
sheets.
```

Building A Document Management System – Part 1 / PHP

Part 1 Of A Three Part Series …

Language: .PHP
Database: MySQL

Editors Comment

In our first issue, we began building our Document Management System using .ASP

CCS Magazine is pleased to announce the conversion of that code from .ASP to .PHP. and we extend our thanks to Mr. Sixto Santos for the next two conversions.

If you haven't downloaded issue 1 of CCS Developer Magazine (FREE) yet, you may want to because the article explains what the application is doing and why it's being done.

It is assumed that the reader has downloaded Issue #1 of CCS Developer Magazine and is familiar with the logic and explanations provided in the relevant article.

Setting Up The Database

Locate the "DocManager_MySQL.sql" file or highlight and copy the SQL located in Figure two (2).

Use this SQL to create the tables, etc. required by the application.

You will need to copy the "dbUpload.php" & "test_download.php" files from your development directory to the DocumentManager directory on your web server.

Ensure that you open both of these files and modify the database connection settings appropriately.

Generating The Document Manager

Create a directory on your web server and create a directory on your development machine. We will call that directory DocumentManager.

Unzip the file:

DocumentManager_ASP+PHP.zip

to the DocumentManager directory on your development machine.

Open the DocumentManager.ccs file (with CodeCharge Studio) and ensure that the project settings indicate that the "Code Language" is set to .PHP.

If necessary, select .PHP as the "Code Language" for publication.

Also, ensure that the publication path to the DocumentManager directory on your web server is correct.

Create The DSN!

Having implemented the data schema mentioned earlier in this article; and having checked the project settings mentioned in this article.

You now need to refer to the Document Management System Article in Issue #1 of CCS Developer Magazine.

Locate the section relating to "Creating The ODBC/DSN Connection" for and create the DocManager DSN.

You may choose to point this DSN at the MS Access Database that accompanies this article or you may point it to your MySQL database.

Note: This code is intended to be executed with a DSN pointed at your MySql database.

Now You Can Generate The PHP Code…

Now, generate the code, publishing it to the DocumentManager directory on your web server.

Open your web browser and point it to the appropriate location on your web server.

You can see the .PHP code in action at;

http://docmanager.tecnoapoyo.com/documentmanager/Login.php

Username is **admin**, password **1**
or **guest1**, password **1**

NOTE:
Mr. Santos pointed out that…
One small detail that you should be aware of is that the maximum filesize for the PHP version is 1MB. This is so because the default installation of MySQL restricts input packages (hence queries) to that size. It is possible to have bigger packages, but that would require a redefinition of the max_allowed_package runtime parameter, something that may be impossible to modify if using a shared hosting service.

About

Sixto Luis Santos

Mr. Santos has been an IT Consultant and Professional Programmer for 15 years.

He started developing applications in CodeCharge since the early days, and moved to CodeCharge Studio as soon as it was released.

His expertise is in PHP and MySQL, but also develops traditional desktop applications in C/C++ and Visual Basic.

He is 34 years old, married, with two daughters Andrea and Ariana, and a still unborn son, soon to be called Bryan.

Building A Document Management System – Part 1 / PHP

Part 1 Of A Three Part Series ...

Language: .PHP
Database: MySQL

Figure 2 - MySQL Data Schema For DocManager Application

```
# Database : docmanager

CREATE TABLE `DocumentManagerDepartments` (
  `id` int(11) NOT NULL auto_increment,
  `file_area` varchar(50) NOT NULL default '',
  PRIMARY KEY  (`id`)
) TYPE=MyISAM;

INSERT INTO `DocumentManagerDepartments` (`id`, `file_area`) VALUES (1, 'Tools (Public)');

INSERT INTO `DocumentManagerDepartments` (`id`, `file_area`) VALUES (2, 'Source Code (Public)');

INSERT INTO `DocumentManagerDepartments` (`id`, `file_area`) VALUES (3, 'Components (Public)');

# --------------------------------------------------------

CREATE TABLE `DocumentManagerUpload` (
  `UploadID` int(11) NOT NULL auto_increment,
  `user_id` int(11) NOT NULL default '0',
  `UploadDT` datetime NOT NULL default '0000-00-00 00:00:00',
  `Title` varchar(50) NOT NULL default '',
  `Description` text NOT NULL,
  `Keywords` text NOT NULL,
  `DataSize` int(11) NOT NULL default '0',
  `Data` mediumblob,
  `ContentType` varchar(64) NOT NULL default '',
  `SourceFileName` varchar(128) NOT NULL default '',
  `file_area` int(11) NOT NULL default '0',
  `publish` smallint(6) NOT NULL default '0',
  `times_downloaded` int(11) NOT NULL default '0',
  PRIMARY KEY  (`UploadID`),
  KEY `IDX_KEYWORDS` (`Keywords`(128))
) TYPE=MyISAM;

# --------------------------------------------------------

CREATE TABLE `users` (
  `User_ID` int(11) NOT NULL auto_increment,
  `user_login` varchar(20) NOT NULL default '',
  `user_password` varchar(20) NOT NULL default '',
  `first_name` varchar(50) NOT NULL default '',
  `last_name` varchar(50) NOT NULL default '',
  `title` varchar(50) NOT NULL default '',
  `group_id` int(11) NOT NULL default '0',
  PRIMARY KEY  (`User_ID`)
) TYPE=MyISAM;


INSERT INTO `users` (`User_ID`, `user_login`, `user_password`, `first_name`, `last_name`, `title`, `group_id`) VALUES (1, 'Admin', '1', 'The', 'Administrator', 'Site Administrator', 5);

INSERT INTO `users` (`User_ID`, `user_login`, `user_password`, `first_name`, `last_name`, `title`, `group_id`) VALUES (2, 'guest1', '1', 'guest', 'level 1', 'Level 1 Guest', 1);
```

Formatting Row Colors / PHP

Changing Row Color According To Database Values…

Language: .PHP
Database: MS Access

Editors Comment

In this second conversion, Mr. Santos translates the .ASP code from our first issue to .PHP.

If you haven't downloaded issue 1 of CCS Developer Magazine (FREE) yet, you may want to because the article explains what the application is doing and why it's being done.

It is assumed that the reader has downloaded Issue #1 of CCS Developer Magazine and is familiar with the logic and explanations provided in the relevant article.

Create a directory on your web server and create a directory on your development machine. We will call that directory RowColors.

Unzip the file:

RowColors_PHP.zip

to the RowColors directory on your development machine.

Open the RowColors.ccs file (with CodeCharge Studio) and ensure that the project settings indicate that the "Code Language" is set to .PHP.

If necessary, select .PHP as the "Code Language" for publication.

Also, ensure that the publication path to the RowColors directory on your web server is correct.

Create The NorthWind Products Table Using MySQL

Warning!

The SQL script referred to below will create a table called 'Products'.

If your MySQL database already contains a table called 'Products', you may need to alter the name of the table in the SQL script before you run it - so that you do not mistakenly modify your real 'Products' table.

If you want to use the Northwind Database that comes with this article point the database connection to the Northwind.mdb file. If you haven't copied this file into the RowColors directory you may want to do so now.

Alternatively, if you want to recreate the Northwind 'Products' Table using MySQL, you can run the script –

"northwind_products_MySQL.sql"

You now need to refer to the Formatting Row Colors Article in Issue #1 of CCS Developer Magazine.

Now You Can Generate The PHP Code…

Now, generate the code, publishing it to the DocumentManager directory on your web server.

Replacing The default_events.php file

After you have generated and published the code to the RowColors table, you will need to replace the default_events file with the one found in the zip file that accompanies this article.

This saves you the time of re-typing the code.

Generate and publish the default.ccs page again so that the changed default_events.php file is transferred to your web server.

Open your web browser and point it to the appropriate location on your web server.

Refer to the article located in Issue #1 for more information about dynamically formatting row colors.

As you'll discover, all of the real work occurs in the Products_Hidden1_BeforeShow() function.

Send Us Your Article

What are you waiting for?

Get your article published in the next issue of CCS Developer Magazine!

function Products_Hidden1_BeforeShow()

```
function Products_Hidden1_BeforeShow()
{
   $Products_Hidden1_BeforeShow = true;
//End Products_Hidden1_BeforeShow

//Custom Code @10-B8559C5D
// -------------------------
   global $Products;
        $sStyle = "OliveDataTD";  //Default Style

        $NumUnitsInStock = $Products->ds->f("UnitsInStock");

        //handle any null values
        if(strlen($NumUnitsInStock)==0)
                $NumUnitsInStock=0;

        if($NumUnitsInStock < 6)
                $sStyle = "OliveRedDataTD";
        elseif($NumUnitsInStock > 6 && $NumUnitsInStock < 21)
                $sStyle = "OliveOrangeDataTD";
        elseif($NumUnitsInStock > 20 && $NumUnitsInStock < 51)
                        $sStyle = "OliveYellowDataTD";


        $Products->Hidden1->SetValue($sStyle);
// -------------------------
//End Custom Code

//Close Products_Hidden1_BeforeShow @8-4DE10D30
   return $Products_Hidden1_BeforeShow;
}
//End Close Products_Hidden1_BeforeShow


?>
```

The CodeCharge-PHP Common Database Interface. Using It To Produce MS-EXCEL Reports

Language: .PHP
Database: Oracle / Sybase

Written By: Fernando Sibaja-Araya.
(fsibaja@gmail.com)

PHP provides a wide variety of database support. Everybody knows that its set of database extensions is one of the PHP features that make it so popular and accepted.

Although PHP's lack of database standardization is an obstacle to building database-independent applications.

Each database extension has its own set of functions and these are defined to fulfill the needs or conception of its engine.

As an example of this, we show two code fragments that do the same thing (just execute an SQL statement), but for two different databases: ORACLE and SYBASE.

Refer To Figure 1.

Figure 1 – Connection Strings

```
$link_ID=OCIplogon ("user","password","database")
 or die("Connection failed");
$query_ID = OCIParse($link_ID, "SELECT * FROM TABLE");
OCIExecute($query_ID);

$link = sybase_connect('server', "user", "password")
  or die("Connection failed");
sybase_select_db("db",$link)
$query_ID = sybase_query("SELECT * FROM TABLE", $link);
```

The disadvantage of PHP developing without any framework (well, one of them), is that if the developer doesn't use some kind of database class library, then porting the application to another database would be a complex and buggy work.

In the last example we can notice how the inherent differences between the databases engines make the connection process a whole different thing.

Fortunately we use a framework, we use CodeCharge Studio, and the people from YesSoftware Inc. have done their homework very well - designing a common database library class.

The advantages of the CCPCDC(Codecharge-PHP common database class) are the following.

- It uses the native extension functions provided by PHP, so there is no need of additional software installation.
- You have access to its implementation, so you can truly understand how it works.
- You have a visual administration tool to manage your connections as objects from the IDE, so change across databases wouldn't be easier.

CodeCharge-PHP Common database interface API.

Each implementation has different properties depending of the database engine, we will show just the common stuff.

Since PHP4 isn't a really object oriented language and doesn't have access modifiers we'll just show the methods that should be public, but the classes have many more.

Properties:

DBHost
The database server name or IP.

Some databases like ORACLE don't make use of this.

DBDatabase
The database name.

DBUser
The user name or schema.

DBPassword
The password.

Construction:

The constructor name depends of the name that you use for your database connection, and has the form <prefix>DatabaseName, where prefix is "clsDB".

For example if you call your connection "myDatabase" then the class name will be clsDBmyDatabase.

The assignation of the database properties (DBHost, DBDatabase, DBUser, DBPassword) is made by extending the generic class, so you just have to take care of these values using the design view.

Methods:

query(String queryString)
Executes the query specified in queryString.

boolean next_record()
Fetches the next query's row. Return true if a new row is retrieved, false if not. If you want to know if a specific query is empty then you'll have to call this method at least once.

f(String index), f(int index)
Returns the value of the column indicated by index.

You must call to next_record() before. you can use the string identifier used in the query or the ordinal number of the field.

close()
Releases the resources used by the connection.

Be sure to close the connections that you use after every query.

Believe it or not, these four methods are all you need to know to perform all the database tasks that you will have to.

For example, refer to Figure 2.

As we all know (and if you don't then you should), the key idea behind of CodeCharge Studio developing is to avoid as much as possible - the custom code - and try to do all the tasks through the IDE.

Although some kind of functionalities like excel or pdf's reporst have to be managed with a couple of tricks.

Now that we have explained the Common Database Interface we will use this knowledge to create a reusable excel report library.

The excel reports

Ok, let's face it.

From the receptionist to the executive manager everybody uses excel, it has become a very common information interchange media.

A web application that performs excel reports would allow your customers to have the information directly in a format that they can manipulate and

The CodeCharge-PHP Common Database Interface. Using It To Produce MS-EXCEL Reports

Language: .PHP
Database: Any

Figure 2 – Example

```
/*
If you already have used a grid or a recordSet with a datasource in the
page, a connection instance will be already created in the page and can
by accessed using de global keyword , is better to use this instead of
creating a new one. If you do need to create a new connection then you
just have to do it this way:
$dbmyDatabase = new clsDBmyDatabase();
 */
global $dbmyDatabase;

$dbmyDatabase->query("SELECT COL1, COL2 FROM MY_TABLE");
echo "Printing query results...\n";
while($dbmyDatabase->next_record()){
  echo "col1: ". $dbmyDatabase->f("COL1") . "col2: " .
$dbmyDatabase->f(1),"\n";
}
$dbmyDatabase->close();
```

Figure 3 – BiffWriter Class Methods

```
$Biff->xlsWriteText //writes text in a cell of the
matrix
 $Biff->xlsWriteNumber //writes a number in a cell
of the matrix
 $Biff->xlsSetFont //sets a
 $Biff->xlsAddFormat(
 $Biff->xlsSetPrintGridLines();
 $Biff->xlsSetPrintHeaders();
 $Biff->xlsHeader
```

How To Obtain Biffwriter

The Biffwriter, developed by Chirstian Novak. (www.cnovak.com)

manage, that would make them happy (or at least they would smile more often).

In this article we will use one libraries that do the excel work writing directly in the binary format (not html tables that the program will have to convert later).

Refer to Figure 3 (How To Obtain Biffwriter).

Please check the site and the license issues.

We will provide you a library that will interact with these two products, but you will have to download the Biffwriter from its correspondent site.

First we will use the biffwriter from Christian Novak.

The biffwriter provides a class that produces the excel spread sheet. We will use the following methods in our library:

Refer To Figure 3.

Teaching how to use the Biffwriter class is beyond the scope of this article, I just will describe briefly the methods while we use it in conjunction with the common database library.

You don't even have to understand it in order to use the functions that we'll provide you.

AutoXlsReport

We have written a small library called autoXlsReport. We defined two main functions in it:
- autoXlsReport_simple
- autoXlsReport_fliped

We will describe the implementation for the first one, and for the other we'll just tell you how to use it.

AutoXlsReport_simple function.

Refer to Figure four (4).

Figure four (4) shows the code for the autoXlsReport_simple function.

This function receives the following parameters:

$usrDB ->
This is a database instance. It could be any kind of supported database. It must come already initialized.

$report_name ->
A string with a meaningful description of the report.

$query_string->
The SQL query for the report. Must be a simple SELECT query.

$column_names_array->
An array containing the header of the columns, this parameter is optional.

If you don't provide it, then the function will use the column's name.

$array_data ->
This is list of the columns from the SELECT statement that will appear in the report.

This parameter is also optional.

It's necessary if you use database functions in your query.

For example, if you use ORACLE and you have a query like this;

SELECT NAME, TO_CHAR(BIRTH_DATE, 'MM/DD/YYYY HH:MI') AS BIRTH
FROM CUSTOMER.

Then you'll have to provide an array like this: array("NAME", "BIRTH").

(We didn't develop a complete SQL parser, we made a simple library. So this is the simplest and easiest way to manage these cases).

Lines from 153 to 158

Lines from 153 to 158 do a little parse work taking the columns names from the query.

If the user provides an array then the function will use it instead of the column list in the SELECT statement.

Lines from 164 to 170

Lines from 164 to 170 create a new biffwriter and give it a default style.

Line 172

Line 172 executes the query.

Line 175

Line 175 prints the headers for the columns.

Line 178

Line 178 executes a while loop that prints each column from the query into the report.

The CodeCharge-PHP Common Database Interface. Using It To Produce MS-EXCEL Reports

Language: .PHP
Database: Any

Figure 4 – autoXLSReport_Simple Function

```
150 function autoXlsReport_simple( $usrDB, $report_name, $query_string, $column_names_array = "", $data_array="")
151 {
152
153  list($select, $from) = spliti(' from ', $query_string);
154  list($anything, $select) = spliti('select', $select);//we split the column names from the rest of the query
155
156  if($data_array == "")//explicit choosed columns
157   $select_array2 = split(',', $select);//we set the choosed columns directly from the query
158  else $select_array2 = $data_array;
159
160  if($column_names_array == "")
161     $column_names_array = $select_array2;
162
163   //Excel generator
164  $Biff = new BiffWriter();
165  autoStyle(&$Biff);
166
167  $Biff->xlsSetPrintGridLines();
168  $Biff->xlsSetPrintHeaders();
169
170  $Biff->xlsHeader("Company name");
171  //we use the query method from Common database class, the connection must come already initialized
172  $usrDB->query($query_string);
173
174   //print headers
175  write_row( &$Biff, $column_names_array, 5, 0, COLUMNA);
176
177  $Row = 6;//starting row
178  while($usrDB->next_record())//loops the data set and print each row
179      write_db_row(&$Biff, &$usrDB, $select_array2, $Row++);
180  //Set a nice header
181  autoHeader(&$Biff, $report_name);
182  $Biff->xlsParse();
183 }
```

Line 182

Line 182 finishes the work and sends the report to the client.

AutoXlsReport_fliped

Sometimes, your customers could want to have their reports in a different way than it's naturally generated.

It happens very often with a master-detail report.

Let's say that you have two tables with a master-detail relationship:

EMPLOYEE and ACTIVE.

Let's say that all the employees have at least one employee assigned but no more than twenty.

If you do a simple select statement like that shown in Figure 5 - then you will have a report similar to that shown in figure six (6).

But your costumers could want it in a more readable form, similar to that

Figure 5 – Example SQL Statement

```
SELECT
   EMPLOYEE.EMPLOYEE_ID,
   EMPLOYEE.NAME,
   ACTIVE.CODE,
   ACTIVE.DESCRIPTION
FROM
   EMPLOYEE, ACTIVE
WHERE
   ACTIVE.EMPLOYEE_ID = EMPLOYEE.EMPLOYEE_ID
ORDER BY
   EMPLOYEE.EMPLOYEE.ID, ACTIVE.CODE
```

Figure 6 – Sample Report

EMPLOYEE_ID	NAME	CODE	DESCRIPTION
001	James S.	4154	PC
001	James S.	4157	Desktop
001	James S.	4500	Palm computer
002	Claudia W.	5000	Desk
002	Claudia W.	5002	Chair

Figure 7 – Sample Report #2

EMPLOYEE_ID	NAME	ACTIVE1	ACTIVE2	... ACTIVE20
001	James S.	4154 PC	4157 Desktop	...
002	Claudia W.	5000 Desk	5002 Chair	

The CodeCharge-PHP Common Database Interface. Using It To Produce MS-EXCEL Reports

Language: .PHP
Database: Any

shown in figure seven (7).

Actually it's the same report, it looks different but it's just a matter of view.

The autoXlsReport_simple takes this master-detail report and flips it into a readable view.

It takes the following arguments:

$usrDB ->
The database instance

$report_name ->
A string with a meaningful description of the report.

$query_string ->
The SQL query.

$pk ->
It's the primary key column.

It's necessary in order to associate each detail to its master.

In the example above the value should be EMPLOYEE_ID

$index_fliped_data ->
The ordinal index from where the detail data starts.

It must be a numeric number.

In the example above it should be the number 2 (starting from the active's code)

$column_names_array , $data_array ->
The same meanings than in the simple report.

Integrating the library with your CodeCharge Project

We'll just add one page to an existing project.

Also, for this example we want to show how you could combine the use of the Common Database interface with our AutoXlsReport library and some data dictionary stuff (Oracle).

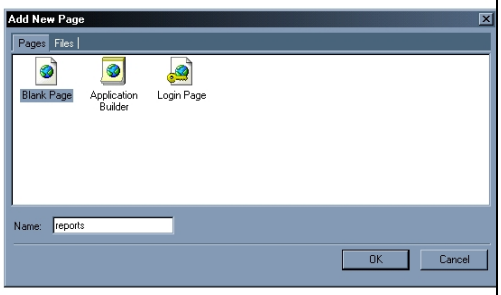
This is just an example to guide you and show you what's possible.

You can rearrange everything to fulfill your needs.

First we create a new page called "reports" – Figure eight (8).

We create a simple recordSet, with

Figure 8 Creating The Reports Page



just one button and a listbox called tableName.

We set its datasource to a simple DataDictionary query:

"SELECT TABLE_NAME FROM USER_TABLES"

(Yes, you guessed it! It returns all of the tables from the current user), and we set its databound and datatext columns to TABLE_NAME.

We also add a little text to the form definition for the record:

```
<form method="post"
action="{Action}&REPORT=YES"
name="{HTMLFormName}">
```

On the afterInitialize event of the page (just after the security stuff) we add the code shown in Figure ten (10) to the AfterInitialize event of the page.

The tableReport function takes the tableName and creates the report.

Refer to Figure eleven (11) to see the code that needs to be plugged into the record_events.php page.

Explanation

First we make the properly include of the autoXlsReport file.

Then we declare the global $DBTEST (your database instance) that's already created for this page.

Then we construct the columns array with a fast query to the data dictionary.

After that we declare a simple query: SELECT * FROM $tableName", we just call the autoXlsReport function with the database instance, the title, the query, and the columns array.

Don't forget to place the biffwriter files and the autoXlsReport.php file in your publishing directory in order to get page working.

This example allows the user to obtain all of the data for any Table on the SCHEMA.

This is a simple example, maybe not a real life case as is, but it show the possibilities and the general template that you should follow to create fast and simple report pages.

Figure 9

The CodeCharge-PHP Common Database Interface. Using It To Produce MS-EXCEL Reports

Language: .PHP
Database: Any

Figure 10 – Page After Initialize Event

```
//Custom Code @8-A43F7AC2
// -------------------------
    global $reports;

    if(CCGetFromGet("REPORT")=="YES"  && CCGetFromPost("TABLE_NAME")){

            tableReport(CCGetFromPost("TABLE_NAME"));
            exit();

           }

// -------------------------
//End Custom Code
```

Figure 11 – Report_Events Code

```
//put this code in your report_events.php page
include("autoxlsreport.php");
function tableReport($tableName){

 global $DBTEST; //already created

 $queryCols = "SELECT COLUMN_NAME ".
           "FROM USER_TAB_COLUMNS ".
           "WHERE TABLE_NAME= '$tableName'";

 $DBTEST->query($queryCols);

 $columns = array();

 while($DBTEST->next_record()){
  $columns[] = $DBTEST->f("COLUMN_NAME");
 }
 $DBTEST->close();

 $query= "SELECT * FROM $tableName";
 $title = "Report of $tableName";

 autoXlsReport_simple($DBTEST, $title,  $query, $columns, $columns);


}
```

Figure 12 – Retrieving Document.xls

About Fernando Sibaja-Araya

Mr. Sibaja is a graduate computer engineer from ITCR (Instituto Techologico de Costa Rica), and develops data driven web applications using CodeCharge Studio, PHP, oracle and ProgressSQL.

Please send your comments or suggestions to fsibaja@gmail.com

Costa Rica

Building A Document Management System

Language: .ASP

FEATURED ARTICLE: Part 3 of a three part series…

In our last issue, we continued building our document management system.

We worked through some fairly difficult segments of code, and in this issue – we increase that complexity by adding additional capabilities.

In building such applications you face ever-increasing degrees of difficulty at virtually every level. If your design is right, the difficulties – though still there, are less difficult than they would have been if you didn't have a good design.

It's difficult to say when you have a good design. But, when you usually know it when you go to implement additional features and they fit like they were planned.

Actually, they were planned. We told you that from the beginning - remember?

So let's get on with it, shall we?

What we need to do next…

We have implemented the ability to upload and download files. But we still need to create the ability for a user to check out a file, check in a file and perhaps, un-do a check out on a file.

By combining these additional capabilities and adding a bit of logic to it, we can begin to see a semblance of what the application is capable of.

Before We Begin

We've added a field to the Users table called:
"DocumentManager_CanAddFile"

This is a yes/no/bit field and you will need to add this field to your users table for the code to work.

We added this field because within organizations – not everyone is going to be given access to the application.

Therefore, by adding this field to the users record and setting it to True, we're effectively telling the application that this user has DocumentManager privileges.

Add this field and set the value to True for the appropriate users before continuing…

Also, we are assuming that you had all of the connection string issue sorted out since #2 and therefore, assume that your connection string is properly set up.

Adding Check-Out Capability

If you recall, we mentioned in an earlier article that the ability to download a file is different from the ability to "Check-Out" a file.

Non-CCS-File
Using FrontPage or your favorite HTML editor, open and refer also to the file "checkoutdownload.asp".

When you download a file, you're doing just that. Downloading it. But when you're Checking Out a file, you're telling everybody else that has access to the file that you have downloaded it for editing.

Therefore, when you check out a document – no one else should be able to Check Out the document for editing.

The routines are virtually the same as that for downloading the file, but in order to track who has the file, we need to track additional information such as, when they checked the file out, who checked out the file, etc.

In order to do this we need to create another table which we will call:
"DocumentManager_CheckInOut"

Refer To Figure 1.

This table has a number of additional fields that appear (in some cases) to resemble the structure of DocumentManager_Upload.

Non-CCS-File
Using FrontPage or your favorite HTML editor, refer to: Function `BackUpDocument()` in the file "checkoutdownload.asp".

Why some of the 'Upload' fields duplicated?

Indeed, some of the fields do resemble those found in table DocumentManager_Upload. The reason for this has more to do with the logic of the application than not.

Let's Consider The Logic

When a user checks out a document, the application automatically makes a copy of the file being "Checked Out" and saves it to the DocumentManager_CheckInOut table.

Alternatively, you could cause the application to perform the same action when the file is checked in.

We have chosen to perform this action at the time the document is checked out for reasons that will be explained below.

But, going back to our example - there are now two (2) copies of the same document in the database. One (1) copy is the one seen in the grid, and the 2nd copy is the one located in the DocumentManager_CheckInOut table.

You might think that strange, but – there's a reason to this madness.

By making a copy of the file when the file is being checked out, we've effectively maintained an 'audit trail' of who edited the document. Should anything happen to the document being shown in the Grid, we still have a copy that can be 'rolled back'.

Let's say for example, that you did not do this… make a copy, that is.

A user checks out "MainDocument1.doc" and makes modifications to it.

When they go to upload the file (Check-In the file), they accidentally upload "MainDocument3.doc" – and that document overwrites the original "MainDocument1.doc" in the record.

Oops!

This is easily corrected however, because all the user needs to do is Check Out the file again, and upload the correct file.

But what if the correct file ("MainDocument1.doc edited version) was corrupted at some point on the users hard disk or floppy disk.

Here's where our methodology pays off. Since the original uncorrupted version of "MainDocuement1.doc" is intact within the "DocumentManager_CheckInOut" table, this record can used to overwrite (and therefore correct the

Building A Document Management System

Part 1 of a three part series, continued…

problem in this case) the corrupted version 2 of the .doc file. Admittedly, it's the earliest good version and version 2 has to be re-keyed, but...

But you can see now why we've chosen this method. It's safe.

Adding Check-In Capability

Non-CCS-File
Using FrontPage or your favorite HTML editor, open and refer also to the file "checkinUpload.asp".

Providing the "Check-In" capability is similar to the performing the original upload routine.

In fact, since we caused the 'history' and a copy of the file to be saved at the time the file was checked out, the only thing happening here is a replication of the upload/overwrite routines.

Adding Un-Do Check-Out Capability

Non-CCS-File
Using FrontPage or your favorite HTML editor, open and refer also to the file "undoCheckin.asp".

The ability to "Un-Do" you Check Out is important.

Sometimes users will inadvertently check out a file only to realize moments later when they open the file, that it the wrong one.

Sometimes they intended to download the file and clicked the "Check-Out" link instead and on realizing this want to correct their action.

Since undoing a check out means that no change will occur to the current document, we simply need to delete the record located within table "DocumentManager_CheckInOut" containing the copy made earlier when the user inadvertently checked out the document.

Setting Document Level User Permissions

Now that we've added the ability to;

- Check-Out a file,
- Check-In a file and

Figure 1 - DocumentManager_CheckInOut

Field Name	Data Type
id	AutoNumber
check_out_date	Date/Time
check_out_time	Date/Time
checked_out_by	Number
check_in_date	Date/Time
check_in_time	Date/Time
uploadid	Number
Title	Text
DataSize	Number
ContentType	Text
SourceFileName	Text
user_id	Number
UploadDT	Date/Time
version_no	Number
times_downloaded	Number
data	OLE Object

Figure 2 – Table: DocumentManager_Upload_To Users

Field Name	Data Type
id	AutoNumber
upload_id	Number
user_id	Number
owner_id	Number
date_added	Date/Time
can_update	Yes/No
can_delete	Yes/No
can_download	Yes/No
can_check_in_out	Yes/No

- Un-Do a Check-Out on a file

We need to provide the application with the ability to discern what actions any given user can take against a given file.

Therefore, we need to modify the table DocumentManager_Upload_To_Users to account for the various actions that a given user can take when they are assigned access to the file.

Refer to Figure 2.

Feel free to include additional options as business requirements dictate.

These 'attributes' are important because they allow us to dynamically create or not create the links necessary to perform a given task.

In other words, they control the 'option' icons that the user sees to the left of the document/file record.

Controlling The Icons/Options For A Document

As mentioned earlier, the Icons representing the options a given user can take against a give file are defined by the bit flags and associated with the user via the DocumentManager_Upload_To_Users table.
Since we cannot dynamically alter the URL reference when a link has been defined as a "Form Link Control", we need to use a "Form label control" instead.

In order to understand this better, open the "DocumentManagerFiles_list.ccp" file and perform a right-click, show code on {lblCheckInOut}.

This code demonstrates one method that can be used to produce a hyperlink, dynamically. That is to say, display an icon or don't – for any given user/for any given file.

Ultimately, once you have the variables available to you, the rest is logic.

Look at the routine located in the "DocumentManagerUpload_lblCheckIn

Refer also to the other article in this issue "Expanding The CodeCharge Studio Portal – Part 2" that explains a method for creating links dynamically based on the name of the page that the user is viewing.

Building A Document Management System

Part 1 of a three part series, continued…

Out_BeforeShow()" event.

Notice that the icon either appears or doesn't depending on the bit flag associated with the user for this document.

Associating Content Type With An Icon

Open the CCS file "DocumentManagerFiles_List.asp" and refer to the function called; "DocumentManagerUpload_lblContent Image_BeforeShow()"

This function associates the content type with an icon. You will build up a list of icons over time, but these are common.

Feel free to add your own content type/icon associations to this routine. For instance; If your users are using a lot of AutoCad files, consider making an icon that consistently represents an AutoCad file..

Refreshing The Screen When The User Checks Out A File

You may have noticed that when a user Checks Out a file (you have to get to and click the 'Save' download button) that the screen automatically refreshes there-by updating the status indicators (icons) for the document.

Open / view Function DocumentManagerUpload_lblCheckIn Out_BeforeShow() located in DocumentMangerFiles_List.ccp.

To do this, we added a javascript call to the onClick event for the Check Out button…

onClick=""doLoad();

This calls the JavaScript function located in the HTML tab.

Refer to Figure 3.

This is an important script that you should place in your 'bag of tricks' for future use. You'll need to perform this type of action often, and this script appears to work in most browsers.

That's why the function is so long… it caters to a number of browsers.

Next Steps…

There you have it. The core functionality that is required to empower you with the minimum capabilities that any document management system requires.

Where you take this technology from here and how you apply it – is up to you.

The flow of the application may change according to the needs of your organization.

Figure 3 – Refreshing/Reloading The Screen

```
<noscript>
<!--  We have the "refresh" meta-tag in case the user's browser does not
correctly support JavaScript or has JavaScript disabled.  Notice that this is
nested within a "noscript" block.-->
<meta http-equiv="refresh" content="4">
</noscript>
<script language="JavaScript">
<!--
var sURL = unescape(window.location.pathname);
function doLoad(){
    // the timeout value should be the same as in the "refresh" meta-tag
    setTimeout( "refresh()", 4*1000 );
}
function refresh(){
    //  This version of the refresh function will cause a new
    //  entry in the visitor's history.  It is provided for
    //  those browsers that only support JavaScript 1.0.
    //
    window.location.href = sURL;
}
//-->
</script>
<script language="JavaScript1.1">
<!--
function refresh(){
    //  This version does NOT cause an entry in the browser's
    //  page view history.  Most browsers will always retrieve
    //  the document from the web-server whether it is already
    //  in the browsers page-cache or not.
    //
    window.location.replace( sURL );
}
//-->
</script>
<script language="JavaScript1.2">
<!--
function refresh(){
    //  This version of the refresh function will be invoked
    //  for browsers that support JavaScript version 1.2
    //

    //  The argument to the location.reload function determines
    //  if the browser should retrieve the document from the
    //  web-server.  In our example all we need to do is cause
    //  the JavaScript block in the document body to be
    //  re-evaluated.  If we needed to pull the document from
    //  the web-server again (such as where the document contents
    //  change dynamically) we would pass the argument as 'true'.
    //
    window.location.reload( false );
}
//-->
</script>
```