PHP HOW−TO

Table of Contents

PHP HOW−TO

Al Dev (Alavoor Vasudevan) alavoor@yahoo.com

v5.0, 14 May 2000

This document tells you howto develop PHP programs and also to migrate all the Windows 95 GUI applications to powerful PHP + HTML + DHTML + XML + Java applets + Javascript. The information in this document applies to all the operating sytems where PHP is ported that is − Linux, Windows 95/NT, OS/2, all flavors of Unix like Solaris, HPUX, AIX, SCO, Sinix, BSD, etc..

20.Appendix J phpDBTest.php3 Example

1.Introduction

PHP stands for 'Hypertext Pre−Processor' and is a server side HTML scripting/programming language. PHP is a tool that lets you create dynamic web pages. PHP−enabled web pages are treated just like regular HTML pages and you can create and edit them the same way you normally create regular HTML pages.

PHP was kept the **"top secret and strictly confidential"** computer language by many companies in the world, but now had become the most well−known and most widely used scripting language for web, internet, e−commerce and business−to−business projects. Even today many competing companies keep PHP language as a highly confidential matter not disclosing to outsiders (competitors).

PHP will storm the entire world and will take the IT industry by surprise!! The power of PHP is that it is **cross−platform and runs everywhere!!** It runs on Linux, Windows 95/98/NT, Windows 2000, Solaris, HPUX and all flavors of unix. PHP is write once and deploy anywhere and everywhere. It runs on many web−servers like Apache, Microsoft IIS, etc..

PHP runs 5 to 20 times faster than Java!! It is extremely easy to use and you can develop very complex web/e−commerce applications very rapidly in a very short period of time.

It has object oriented features and takes the best features from Java, C++, PERL and "C" langauges. PHP language is a *marriage* of best features from Java, C++, PERL and C.

PHP is the **real gem** of all the scripting/programming languges and will soon become the "MECCA" for programmers world−wide!! PHP has a huge user base and a large developer base as it runs on both window95/NT and all flavors of unixes.

PHP can be compiled and optimized to make it run even faster by using the Zend Optimizer. Zend optimizer is integrated with PHP in PHP version 4.0.

You would normally use a combination of PHP (70% code) + HTML/DHTML/XML (25% code) + Javascript (5% code client side validations) for your e−commerce projects.

2.PHP Download

- PHP main site http://www.php.net
- PHP resources http://ils.unc.edu/web−db/php/links.html
- PHP Code Exchange − http://px.sklar.com

2.1 PHP Installation

See the installation guide and instructions at PHP main site http://www.php.net or INSTALL file in the downloaded package itself.

3.PHP Tutorial

In this tutorial we assume that your server has support for PHP activated and that all files ending in .php3 are handled by PHP.

Your first PHP−enabled page: Create a file named hello.php3 and in it put the following lines:

```
      <html>< head>< title >PHP Test< /title >< /head >
      < body>
      <?php echo "Hello World<P>"; ?>
      < /body>< /html>
```

Note that this is not like a CGI script. Think of it as a normal HTML file which happens to have a set of special tags available to you.

If you tried this example and it didn't output anything, chances are that the server you are on does not have PHP enabled. Ask your administrator to enable it for you.

The point of the example is to show the special PHP tag format. In this example we used < ?php to indicate the start of a PHP tag. Then we put the PHP statement and left PHP mode by adding the closing tag, ? > . You may jump in and out of PHP mode in an HTML file like this all you want.

We are going to check what sort of browser the person viewing the page is using. In order to do that we check the user agent string that the browser sends as part of its request. This information is stored in a variable. Variables always start with a dollar−sign in PHP. The variable we are interested in is $HTTP_USER_AGENT. To display this variable we can simply do:

```
      <?php echo $HTTP_USER_AGENT; ?>
```

For the browser that you are using right now to view this page, this displays:

Mozilla/4.0 (compatible; MSIE 4.01; Windows 98)

There are many other variables that are automatically set by your web server. You can get a complete list of them by creating a file that looks like this:

```
      <?php phpinfo()?>
```

Then load up this file in your browser and you will see a page full of information about PHP along with a list of all the variables available to you.

You can put multiple PHP statements inside a PHP tag and create little blocks of code that do more than just a single echo.

```
<?php
if(strstr($HTTP_USER_AGENT,"MSIE")) {
    echo "You are using Internet Explorer<br>";
}
?>
```

We can take this a step further and show how you can jump in and out of PHP mode even in the middle of a PHP block:

```
<?php
if(strstr($HTTP_USER_AGENT,"MSIE"))
{
        ?>
        < center>< b>You are using Internet Explorer< /b>< /center>
        <?
}
else
{
        ?>
        < center>< b>You are not using Internet Explorer< /b>< /center>
        <?
}
?>
```

Instead of using a PHP echo statement to output something, we jumped out of PHP mode and just sent straight HTML. The important and powerful point to note here is that the logical flow of the script remain intact. Only one of the HTML blocks will end up getting sent to the viewer. Running this script right now results in:

You are using Internet Explorer

Dealing with Forms

One of the most powerful features of PHP is the way it handles HTML forms. The basic concept that is important to understand is that any form element in a form will automatically result in a variable with the same name as the element being created on the target page. This probably sounds confusing, so here is a simple example. Assume you have a page with a form like this on it:

```
<form action="action.php3" method="POST">
Your name: <input type=text name=name>
You age: <input type=text name=age>
<input type=submit>
< /form>
```

There is nothing special about this form. It is a straight HTML form with no special tags of any kind. When the user fills in this form and hits the submit button, the action.php3 page is called. In this file you would have something like this:

```
Hi <?php echo $name?>.  You are <?php echo $age?> years old.
```

Surprise!! The $name and $age variables are automatically set for you by PHP !!

4.IDE tools for PHP

Many HTML editors are supporting PHP :

- Blue Fish http://bluefish.linuxave.net
- Coffee cup http://www.coffeecup.com/linux
- Dreamweaver http://www.dreamweaver.com
- Amaya http://www.w3.org/Amaya
- Homesite http://www.homesite.com
- Hotdog http://www.hotdog.com
- Zend Optimizers http://www.zend.com
- Zend Compilers http://www.zend.com

In near future every HTML editors and XML editor will be supporting PHP "Rapid Application Development" tool.

5.ctags for PHP ! Surprise!!!

Tags are extremely valuable and are used for navigation of source code inside the editors like vi, emacs, CRiSP, NEdit etc... If you had programmed a lot in C, C++ or Java you might have used the **ctags** program to create tags. To see the online manual page, type 'man ctags' at linux/unix bash prompt.

The **ptags** program for PHP is given below, which you can use to create the tags for PHP source code. Your **productivity will improve 3 to 4 times** if you use **ptags**.

See also Vim color text editor for PHP, C, C++ at http://metalab.unc.edu/LDP/HOWTO/Vim−HOWTO.html

```
// Save this file as ptags.cpp and compile by
//              g++ −o ptags ptags.cpp
//*****************************************************************
// Copyright policy is GNU/GPL but additional request is
// that you include author's name and email on all copies
// Author : Al Dev Email: alavoor@yahoo.com
// Usage : ptags *.php3 *.inc
//                 This will generate a file called tags
//*****************************************************************
#include <iostream.h>
#include <fstream>
#include <stdio.h> // for sprintf
#include <stdlib.h> // for system
#include <string.h> // for memset
#include <ctype.h> // for isspace

#define BUFF_LEN  1024
#define LOCATION  9

char *ltrim(char *dd);
char *rtrim(char *ee);

main(int argc, char **argv)
{
        if (argc < 2)
        {
                cerr << "\nUsage: " << argv[0] << " file .... " << endl;
                exit(0);
        }

        char fname[100] = "tag_file.out";
        FILE    *fpout;
        ofstream    fout(fname);
        if (fout.fail())
        {
                cerr << "\nError opening file : " << fname << endl;
                exit(−1);
        }
        //fpout = fopen(fname, "w");

        for (int ii = 1; ii < argc; ii++)
        {
                /*
                char buff[2024];

                sprintf(buff, "\\rm −f %s; ls %s > %s 2>/dev/null", outfile, argv[1], outfile);
                cout << "\nbuff = " << buff << endl;

                system(buff);
                fclose(fp);
                */
                FILE *fpin = NULL;
                fpin = fopen(argv[ii], "r");
                if (fpin == NULL)
                {
                        cerr << "\nError opening file : " << argv[ii] << endl;
                        exit(−1);
                }
                char buff[BUFF_LEN + 100];
                memset(buff, 0, BUFF_LEN +10);
                for ( ; fgets(buff, BUFF_LEN, fpin) != NULL; )
                {
                        char aa[BUFF_LEN + 100];
                        memset(aa, 0, BUFF_LEN +10);
                        strcpy(aa, buff);
                        ltrim(aa);

                        // Remove the trailing new line..
                        {
                                int tmpii = strlen(aa);
                                if (aa[tmpii−1] == '\n')
                                        aa[tmpii−1] = 0;
                        }
                        //cout << "aa is : " << aa << endl;
                        if (strncmp(aa, "function ", LOCATION) != 0)
                                continue;
                        //cout << buff << endl;

                        // Example tags file output is like −
                        // al2  al.c    /^al2()$/;"     f
                        {
                                char bb[BUFF_LEN + 100];
                                memset(bb, 0, BUFF_LEN +10);
                                strcpy(bb, & aa[LOCATION]);
                                char *cc = bb;
                                while (cc != NULL && *cc != '(')
                                        *cc++;
                                *cc = 0;
                                cc = rtrim(bb);
                                //cout << "bb is : " << bb << endl;
                                //cout << cc << "\t" << argv[ii] << "\t" << "/^" << aa << "$/;\"\
                                fout << cc << "\t" << argv[ii] << "\t" << "/^" << aa << "$/;\"\tf
                                //fprintf(fpout, "%s\t%s\t/^%s$/;\"f\n", cc, argv[ii], aa );
                        }

                        memset(buff, 0, BUFF_LEN +10);
                }
                fclose(fpin);
        }
        fout.flush();
        fout.close();
        //fclose(fpout);

        // Sort and generate the tag file
        {
                char tmpaa[1024];
                sprintf(tmpaa, "sort %s > tags; \\rm −f %s", fname, fname);
                system(tmpaa);
        }
}

char *ltrim(char *dd)
{
    if (dd == NULL)
        return NULL;

    while (isspace(*dd))
        dd++;

        return dd;
}

char *rtrim(char *ee)
{
    if (ee == NULL)
        return NULL;

        int tmpii = strlen(ee) − 1;
        for (; tmpii >= 0 ; tmpii−−)
        {
                if (isspace(ee[tmpii]) )
                {
                        //cout << "\nis a space!!" << endl;
                        ee[tmpii] = 0;
                }
        }
        return ee;
}
```

6.Debugging PHP

To debug PHP programs create a file "debug2.inc" having the following functions :

```
<?php

/* define this variable, to prevent double declaration. */
if (!defined("_DEBUG2_DEFINED_"))
{
        define("_DEBUG2_DEFINED_", 1 );
}
else
        return; // if this file is already included then return

# file name : debug2.inc
# Functions for debuging the PHP source code
#*****************************************************************
# Copyright policy is GNU/GPL but additional request is
# that you include author's name and email on all copies
# Author : Al Dev Email: alavoor@yahoo.com
#*****************************************************************

# Usage of this functions −
# In your source code put something like −
# debug2_(__FILE__, __LINE__, "f_somevariable", $f_somevariable);
# And this will generate output in debug.out file.

//function debug2_($fname, $lname, $debug_var, $debug_value=0) {}

// Give read, exec for all on directory /debug2_logs
// chmod a+rwx /debug2_logs
// But here you need to open the file in append mode.
$fp_debug2 = fopen("/debug2_logs/debug.out", "a");
if ($fp_debug2 == false)
{
        print "<b>File open failed − global.var.inc<b>";
        exit;
}

function debug2_($fname, $lname, $debug_var, $debug_value=0)
{
        global $fp_debug2;

        //print "<br> debug_value is : $debug_value <br>";
        if (!$debug_value)
        {
                fwrite($fp_debug2, "\n ". $fname ."  ". $lname .": $debug_var");
        }
        else
        {
                fwrite($fp_debug2, "\n ". $fname . " ". $lname .": $debug_var = $debug_value");
        }
        //print "<br> f_cookie is : $f_cookie <br>";
}

// In your first page, which is generally index.php3
// truncate the debug2_logs file in beginning of code
function init_debug_file()
{
        global $fp_debug2;
        $fp_debug2 = fopen("/debug2_logs/debug.out", "w");
        if ($fp_debug2 == false)
        {
                print "<b>File open failed − global.var.inc<b>";
                exit;
        }
        system("chmod a+rwx /debug2_logs/debug.out");
}

?>
```

In your PHP source code initial page which is generally index.php3, put a line like

```
<?php
        include ("debug2.inc");

        init_debug_file();
        // all other commands follows here ...
        // ...........
?>
```

To output debug values, in your PHP source code files, put debug2_() calls as illustrated below:

```
<?php
include ("debug2.inc");
debug2_(__FILE__, __LINE__, "f_somevariable", $f_somevariable);

function aa()
{
        $aa = 8;
        debug2_(__FILE__, __LINE__, "aa", $aa);
}
?>
```

When you run the PHP program the output will be traced in the file called debug.out giving the filename, linenumber, variable name and it's value.

Use the debug2_() generously in your code. The usage of debug2_() calls in your program will **NOT** have any impact on the final production code and also has no impact on the performance because they will be filtered out as described below. You can use copy and paste to save time of typing debug2() calls or use the 'yank to buffer' feature of Vi editor and paste.

When you are done development and testing and when you are ready to deploy on the production server, filter out the debug2_ calls from your source code. At unix prompt −

```
bash$ mkdir production
bash$ grep −v debug2_  filea.php3 > production/filea.php3
```

For a large group of files −

```
bash$ mkdir production
bash$ ls *.php3 | while read ans
do
        grep −v  debug2_ $ans > production/$ans
```

```
done
```

And now copy the files from production to the deployment area.

7.Limitations of PHP

Everything has limitations or disadvantages and PHP is no exception. The following are the limitations of PHP (so be **WARNED !!**)

1. PHP is NOT 100 % pure Object Oriented scripting language. PHP is good if your PHP code size does not exceed 3,00,000 lines. Maintainence of PHP code greater than 1,00,000 lines becomes more difficult.
2. PHP will NOT give the performance of "C" or "C++" language. Because it is scripting language and is interpreted it will be a bit slower than the optimized "C++" programs. For top performance, you should use "C++" and fast−CGI with database/webserver connection pooling and use C++ compiler optimizer "−O3" options. Zend optimizer in PHP 4 will speed up the performance of PHP to certain extent.

On the other hand, PHP has lot of advantages and it's advantages outweigh it's limitations −

1. You can very rapidly develop web applications in PHP as compile and link is eliminated in PHP scripting language.
2. PHP applications are very stable and do not depend on the browser technologies unlike Javascript applications which depend on browsers. PHP will give you the freedom to select any server platform and browser does not know that the HTML page is generated by PHP!!
3. PHP has excellent database conectivity to all SQL database servers.
4. PHP has partial support for Object oriented features
5. PHP has C++, Perl, Javascript like syntax features and has programs like 'ptags/ctags' to navigate the source code
6. PHP has Zend optimizer which speeds up the performance
7. PHP runs on all unixes, linux, Windows 95/NT/2000 and is more powerful than ASP, JSP and others.
8. PHP has a very large user base and developer base.

WARNING: If you want 100% pure Object Oriented scripting language than you MUST consider **Python**. The 'Python' is a object oriented scripting language from ground up. You would be using the Python Web Application server called 'Zope' which is available at − http://www.zope.org and python is at

8.Related URLs

Visit following locators which are related to C, C++ −

- Vim color text editor for C++, C http://metalab.unc.edu/LDP/HOWTO/Vim−HOWTO.html
- SQL database server for PHP PostgreSQL http://metalab.unc.edu/LDP/HOWTO/PostgreSQL−HOWTO.html
- Source code control system CVS HOWTO for C++ programs

http://metalab.unc.edu/LDP/HOWTO/CVS−HOWTO.html
- Linux goodies main site http://www.aldev.8m.com
- Linux goodies mirror site http://aldev.webjump.com

9.Other Formats of this Document

This document is published in 11 different formats namely − DVI, Postscript, Latex, Adobe Acrobat PDF, LyX, GNU−info, HTML, RTF(Rich Text Format), Plain−text, Unix man pages and SGML.

- You can get this HOWTO document as a single file tar ball in HTML, DVI, Postscript or SGML formats from − ftp://metalab.unc.edu/pub/Linux/docs/HOWTO/other−formats/ or ftp://metalab.unc.edu/pub/Linux/docs/HOWTO/other−formats/
- Plain text format is in: ftp://metalab.unc.edu/pub/Linux/docs/HOWTO or ftp://metalab.unc.edu/pub/Linux/docs/HOWTO
- Translations to other languages like French, German, Spanish, Chinese, Japanese are in ftp://metalab.unc.edu/pub/Linux/docs/HOWTO or ftp://metalab.unc.edu/pub/Linux/docs/HOWTO Any help from you to translate to other languages is welcome.

The document is written using a tool called "SGML tool" which can be got from − http://www.xs4all.nl/~cg/sgmltools/ Compiling the source you will get the following commands like

- sgml2html PHP−HOWTO.sgml (to generate html file)
- sgml2rtf PHP−HOWTO.sgml (to generate RTF file)
- sgml2latex PHP−HOWTO.sgml (to generate latex file)

This document is located at −

- http://metalab.unc.edu/LDP/HOWTO/PHP−HOWTO.html

Also you can find this document at the following mirrors sites −

- http://www.caldera.com/LDP/HOWTO/PHP−HOWTO.html
- http://www.WGS.com/LDP/HOWTO/PHP−HOWTO.html
- http://www.cc.gatech.edu/linux/LDP/HOWTO/PHP−HOWTO.html
- http://www.redhat.com/linux−info/ldp/HOWTO/PHP−HOWTO.html
- Other mirror sites near you (network−address−wise) can be found at http://metalab.unc.edu/LDP/hmirrors.html select a site and go to directory /LDP/HOWTO/PHP−HOWTO.html

In order to view the document in dvi format, use the xdvi program. The xdvi program is located in tetex−xdvi*.rpm package in Redhat Linux which can be located through ControlPanel | Applications | Publishing | TeX menu buttons.

```
        To read dvi document give the command −
                xdvi −geometry 80x90 howto.dvi
        And resize the window with mouse. See man page on xdvi.
```

```
                To navigate use Arrow keys, Page Up, Page Down keys, also
                you can use 'f', 'd', 'u', 'c', 'l', 'r', 'p', 'n' letter
                keys to move up, down, center, next page, previous page etc.
                To turn off expert menu press 'x'.
```

You can read postscript file using the program 'gv' (ghostview) or 'ghostscript'. The ghostscript program is in ghostscript*.rpm package and gv program is in gv*.rpm package in Redhat Linux which can be located through ControlPanel | Applications | Graphics menu buttons. The gv program is much more user friendly than ghostscript. Ghostscript and gv are also available on other platforms like OS/2, Windows 95 and NT.

- Get ghostscript for Windows 95, OS/2, and for all OSes from http://www.cs.wisc.edu/~ghost

```
                To read postscript document give the command −
                        gv howto.ps

                To use ghostscript give −
                        ghostscript howto.ps
```

You can read HTML format document using Netscape Navigator, Microsoft Internet explorer, Redhat Baron Web browser or any other web browsers.

You can read the latex, LyX output using LyX a "X−Windows" front end to latex.

10.Copyright

Copyright policy is GNU/GPL as per LDP (Linux Documentation project). LDP is a GNU/GPL project. Additional requests are − Please retain the author's name, email address and this copyright notice on all the copies. If you make any changes or additions to this document then you please intimate all the authors of this document.

11.Appendix A Database Wrapper Example

Submitted by: Barton Greg greg@createtech.com To get this file, in the web−browser, save this file as 'Text' type as pgsql.lib

---

```
This  is  a  database   wrapper  for  PostgreSQL,  but
can  be  simply  modified  for  any  other   database  type.


<?php

if ($dbObjDefined  !=  1)
{
        $dbObjDefined  =  1;

        // Wrapper  class  for  database  calls
```

```
class  dbObj
{
        // Connection  handle  to  database
        var  $conn;

        // Default  connection  parameters
        var  $host  =  "YourSite.com";
        var  $user  =  "johndoe";
        var  $password  =  "pwd";
        var  $port  =  "5432";
        var  $dbname  =  "MyDB";

        // Open  initial  connection.  $params  is
        // an  associative  array  holding
        // parameters  to  the  pg_Connect  function.
        function  init($params)
        {
                if(isset($parame[host]))
                        $host  =  $parame[host];
                else
                        $host  =  $this−>host;

                if(isset($parame[user]))
                        $user  =  $parame[user];
                else
                        $user  =  $this−>user;

                if(isset($parame[password]))
                        $password  =  $parame[password];
                else
                        $password  =  $this−>password;

                if(isset($parame[port]))
                        $port  =  $parame[port];
                else
                        $port  =  $this−>port;

                if(isset($parame[dbname]))
                        $dbname  =  $parame[dbname];
                else
                        $dbname  =  $this−>dbname;

                $this−>conn  =  pg_Connect ( "  host=$host  user=$user  password=$passwor
        }

        // Send  SQL  to  database  connection.
        // Return  recordset  object  on  success.
        // Return  0  on  failure.
        function  exec($SQL)
        {
                $this−>resultset  =  pg_Exec($this−>conn,  $SQL);

                if  ($this−>resultset)
                {
                        $recset  =  new  recordset;
                        $recset−>init($this−>resultset);
                        return  $recset;
                }
                else
                {
                        return  0;
                }
        }

        function valid()
        {
                return $this−>resultset;
        }

        // Close  connection  to  database
        function  free()
        {
                pg_close($this−>conn);
        }
};

/*
** This  is  a  simple  recordset  class  which  can  be
** traversed  using  next(),  prev(),  and  current()  methods.
** It  is  initialized  from  a  resultset  returned  from  the
** function  "pg_Exec"  or  can  be  generated  by  a  call  to  the
** exec  method  from  the  dbObj class given above.
** Below "Tuples" means rows.
*/
class  recordset
{
        var  $resultset;
        var  $index;
        var  $numFields;
        var  $numTuples;

        function  init($newResultset)
        {
                $this−>resultset  =  $newResultset;
                $this−>index  =  0;
                $this−>numFields  =  pg_NumFields($this−>resultset);
                $this−>numTuples  =  pg_NumRows($this−>resultset);
        }

        // Used in display() below
        function valid()
        {
                return $this−>resultset;
        }

        // Get  a  value  by  row  number  and  either
        // column  name  or  column  number
        function  getVal($row,  $col)
        {
                return  pg_Result($this−>resultset,  $row,  $col);
        }

        // Return  an  array  of  field  names
        function  getFields()
        {
                for ($i=0;  $i < $this−>numFields;  $i++)
                        $retArray[] = pg_FieldName($this−>resultset,  $i);
                return  $retArray;
        }

        // Get  number  of  columns  in  resultset
        function  getNumFields()
        {
                return  $this−>numFields;
```

```
}

// Get  a  tuple  (associative  array  of
// column  values)  by  row  number
function  getTupleDirect($row)
{
        for ($i=0;  $i < $this−>numFields;  $i++)
        {
                $retArray[pg_FieldName($this−>resultset,  $i)]  =
                        pg_Result($this−>resultset,  $row,  $i);
        }
        return  $retArray;
}

// Get  an  array  filled  with  all  values  in  a  column
// (using  either  column  name  or  column  number)
function  getColumn($col)
{
        for ($i=0; $i < $this−>numTuples; $i++)
                $retArray[]  =  pg_Result($this−>resultset,  $i,  $col);
        return  $retArray;
}

// Return  the  number  of  records  in  the  recordset
function  getNumTuples()
{
        return  $this−>numTuples;
}

// Get  tuple  pointed  to  by  the  current  index
function  getTuple()
{
        if ($this−>index >= 0  &&  $this−>index  <  $this−>numTuples)
                return  $this−>getTupleDirect($this−>index);
        else
                return  0;
}

function valueof($col)
{
        if ($col < $this−>numFields)
        {
                return pg_Result($this−>resultset,  $this−>index,  $col);
        }
        else
        {
                return "";
        }
}

// Reached last row − end of rows ? Used in display() below
function eof()
{
        return $this−>index == $this−>numTuples;
}

// Return  1  if  index  is  within  bounds  of  the  recordset
function  current()
{
        if ($this−>index >= 0  &&  $this−>index  <  $this−>numTuples)
                return  1;
        else
```

```
                return  0;
}

// Increment  index. Used in display() below
function  next()
{
        if ($this−>index < $this−>numTuples)
        {
                $this−>index++;
                return  1;
        }
        else
        {
                return  0;
        }
}

// Decrement  index
function  prev()
{
        if ($this−>index  >=  0)
        {
                $this−>index−−;
                return  1;
        }
        else
        {
                return  0;
        }
}

// Reset  index  to  0 − See also first()
function  reset()
{
        $this−>index  =  0;
}

// See also reset(). Used in display() below
function  first()
{
        $this−>index  =  0;
}

function last()
{
        $this−>index = $this−>numTuples −1 ;
}

// Used in display() below
function showheader($col, $fmt = "")
{
        printf("\t< th %s>%s< /th >\n", $fmt,
        is_string($col) ? $col : pg_fieldname($this−>resultset, $col));
}

// Used in display() below
function showvalue($col, $fmt =  "", $def =  " ")
{
        $v = $this−>valueof($col);
        printf( "\t< td %s>%s< /td>\n", $fmt, $v ==  "" ? $def : $v);
}
```

```
                function showurl($col, $fmt =  "")
                {
                        $v = $this−>valueof($col);
                        if ( $v !=  "" )
                        {
                                printf("\t< td %s> < /td>\n", $fmt);
                        }
                        else
                        {
                                printf( "\t< td %s>< a href=%s>%s< /a>< /td>\n", $fmt, $v, $v);
                        }
                }

                function display()
                {
                        if (!$this−>valid() )
                        {
                                return;
                        }

                        printf( "<table cellspacing=1 cellpadding=1 border=1>\n");

                        printf( "<tr>\n");
                        for ($c = 0; $c < $this−>cols; $c++ )
                        {
                                $this−>showheader($c);
                        }
                        printf( "< /tr>\n");

                        $this−>first();
                        while (!$this−>eof())
                        {
                                printf( "<tr>\n");

                                for ($c = 0; $c < $this−>cols; $c++)
                                {
                                        $this−>showvalue($c);
                                }

                                printf( "< /tr>\n");
                                $this−>next();
                        }
                        printf("< /table\n");
                }

                // Free  memory  allocated  to  recordset.
                function  free()
                {
                        pg_Freeresult($this−>resultset);
                }
        };
}

?>
```

12.Appendix B SQL abstraction Example

Submitted by: Gianugo Rabellino nemorino@opera.it To get this file, in the web−browser, save this file as 'Text' type as sqlabst.lib

```
PX: PHP Code Exchange
<?php

/*
*    SAL − SQL Abstraction Library
*           version 0.01
*/

/*
** Set the variable $dbtype to any of the following
** values: MySQL, mSQL, Postgres, ODBC before including this library
*/
// $dbtype = "MySQL";
// $dbtype = "mSQL";
// $dbtype = "PostgreSQL";
// $dbtype = "ODBC";

// SQL_connect($host, $user, $password, $db)
// returns the connection ID

function SQL_connect($host, $user, $password, $db)
{
        global $dbtype;

        switch ($dbtype)
        {
                case  "MySQL":
                        $conn=mysql_pconnect($host, $user, $password);
                        mysql_select_db($db);
                        return $conn;
                        break;;

                case  "mSQL":
                        $conn=msql_pconnect($host);
                        msql_select_db($db);
                        return $conn;
                        break;;

                case  "PostgreSQL":
                        $conn=pg_pconnect($host, "5432", "",$db);
                        return $conn;
                        break;;

                case  "ODBC":
                        $conn=odbc_pconnect($db,$user,$password);
                        return $conn;
                        break;;

                default:
                        $conn=mysql_pconnect($host, $user, $password);
                        mysql_select_db($db);
                        return $conn;
                        break;;
        }
```

```
}

// SQL_query($host, $user, $password, $db)
// executes an SQL statement, returns a result identifier
function SQL_query($query, $id)
{
        global $dbtype;
        switch ($dbtype)
        {
                case  "MySQL":
                        $res=mysql_query($query, $id);
                        return $res;
                        break;;

                case  "mSQL":
                        $res=msql_query($query, $id);
                        return $res;
                        break;;

                case  "PostgreSQL":
                        $res=pg_exec($id,$query);
                        return $res;
                        break;;

                case  "ODBC":
                        $rid=odbc_prepare($id,$query);
                        $res=odbc_execute($rid);
                        return $res;
                        break;;

                default:
                        $res=mysql_query($query, $id);
                        return $res;
                        break;;
        }
}

// SQL_num_rows($host, $user, $password, $db)
// given a result identifier, returns the number of affected rows
function SQL_num_rows($res)
{
        global $dbtype;

        switch ($dbtype)
        {
                case  "MySQL":
                        $rows=mysql_num_rows($res);
                        return $rows;
                        break;;

                case  "mSQL":
                        $rows=msql_num_rows($res);
                        return $rows;
                        break;;

                case  "PostgreSQL":
                        $rows=pg_numrows($res);
                        return $rows;
                        break;;

                case  "ODBC":
                        $rows=odbc_num_rows($res);
                        return $rows;
                        break;;

                default:
                        $rows=mysql_num_rows($res);
                        return $rows;
                        break;;
        }
}


// SQL_fetchrow($res,$row)
// given a result identifier, returns an array with the resulting row
// Needs also a row number for compatibility with PostgreSQL
function SQL_fetch_row($res, $nr)
{
        global $dbtype;

        switch ($dbtype)
        {
                case  "MySQL":
                        $row = array();
                        $row = mysql_fetch_row($res);
                        return $row;
                        break;;

                case  "mSQL":
                        $row = array();
                        $row = msql_fetch_row($res);
                        return $row;
                        break;;

                case  "PostgreSQL":
                        $row = array();
                        $row = pg_fetch_row($res,$nr);
                        return $row;
                        break;;

                case  "ODBC":
                        $row = array();
                        $cols = odbc_fetch_into($res, $nr, &$row);
                        return $row;
                        break;;

                default:
                        $row = array();
                        $row = mysql_fetch_row($res);
                        return $row;
                        break;;
        }
}

// SQL_fetch_array($res,$row)
// given a result identifier, returns an associative array
// with the resulting row using field names as keys.
// Needs also a row number for compatibility with PostgreSQL.
function SQL_fetch_array($res, $nr)
{
        global $dbtype;

        switch ($dbtype)
        {
                case  "MySQL":
                        $row = array();
                        $row = mysql_fetch_array($res);
                        return $row;
                        break;;

                case  "mSQL":
                        $row = array();
                        $row = msql_fetch_array($res);
                        return $row;
                        break;;

                case  "PostgreSQL":
                        $row = array();
                        $row = pg_fetch_array($res,$nr);
                        return $row;
                        break;;

                /*
                * ODBC doesn't have a native _fetch_array(), so we have to
                * use a trick. Beware: this might cause HUGE loads!
                */

                case  "ODBC":
                        $row = array();
                        $result = array();
                        $result = odbc_fetch_row($res, $nr);
                        $nf = count($result)+2;  /* Field numbering starts at 1 */
                        for ($count=1; $count < $nf; $count++)
                        {
                                $field_name = odbc_field_name($res, $count);
                                $field_value = odbc_result($res, $field_name);
                                $row[$field_name] = $field_value;
                        }
                        return $row;
                        break;;
        }
}
```

13.Appendix C PostgreSQL large object Example

Submitted by: PHP code exchange px@sklar.com To get this file, in the web−browser, save this file as 'Text' type as pgsql_largeobj.lib

```
PX: PHP Code Exchange − PostgreSQL large object access

<?
        $database = pg_Connect ( "",  "",  "",  "",  "jacarta");
        pg_exec ($database,  "BEGIN");
        $oid = pg_locreate ($database);
        echo ( "$oid\n");
        $handle = pg_loopen ($database, $oid,  "w");
        echo ( "$handle\n");
        pg_lowrite ($handle,  "foo");
        pg_loclose ($handle);
        pg_exec ($database,  "COMMIT");
        pg_close ($database);
?>
```

14.Appendix D User authentication Example

To get this file, in the web−browser, save this file as 'Text' type as user_pw.lib

From the PHP 3 Manual: Works only if PHP is an Apache module. Instead of simply printing out the $PHP_AUTH_USER and $PHP_AUTH_PW, you would probably want to check the username and password for validity. Perhaps by sending a query to a database, or by looking up the user in a dbm file.

```
<?php
        if (!$PHP_AUTH_USER)
        {
                Header("WWW-authenticate: basic realm=\"My Realm\"");
                Header("HTTP/1.0 401 Unauthorized");
                echo "Text to send if user hits Cancel button\n";
                exit;
        }
        else
        {
                echo "Hello $PHP_AUTH_USER.<P>";
                echo "You entered $PHP_AUTH_PW as your password.<P>";
        }
?>
```

15.Appendix E Network admin Example

To get this file, in the web−browser, save this file as 'Text' type as network.lib

PHP: network adminstrator's best friend from http://www.phpWizard.net

As a web−developer, you're probably used to such lovely tools as ping, whois, nslookup etc. But what when you need one of those utilities at a client's office and have no access to telnet? Good guess. Time to look up the functions in the "Network" section of the PHP manual.

Socket operations:

The most important function there is fsockopen(). Using this function, you can connect to any open port on a server and establish a socket connection with it. The function's syntax is as following:

```
        int fsockopen(string hostname, int port, int [errno], string [errstr]);
```

The first two arguments are obvious, the next two are optional and used for error handling. The "errno" and "errstr" should be passed by reference. "Passing by reference" means that the original variable will get modified. Normally, the content of a variable passed to a function wouldn't be modified.

So, you could use this function to open a connection to a webserver and print out the headers:

```
function get_headers($host, $path = "/")
{
        $fp = fsockopen ("$host", 80, &$errnr, &$errstr) or die("$errno: $errstr");
        fputs($fp,"GET $path HTTP/1.0\n\n");
        while (!$end)
        {
                $line = fgets($fp, 2048);
                if (trim($line) == "")
                        $end = true;
                else
                        echo $line;
        }
        fclose($fp);
}
```

In this example you see that you can apply any file operations (fread, fwrite etc) to the the pointer you got using the fsockopen() call. Note that the example realizes a HTTP/1.0 client − it won't work with name−based virtual hosts.

Finger: Naturally, you can also open connections to other ports. Writing a small finger client with PHP is trivial therefore. Let's change the example from above to query a finger daemon:

```
function finger ($host, $user)
{
        $fp = fsockopen($host, 79, &$errno, &$errstr) or die("$errno: $errstr");
        fputs($fp, "$user\n");
        while (!feof($fp))
                echo fgets($fp, 128);
        fclose($fp);
}
```

Whois: Querying a whois server uses the same concept:

```
// domain is like "phpwizard.net"
function whois($domain, $server="whois.internic.net")
{
        $fp = fsockopen ($server, 43, &$errnr, &$errstr) or die("$errno: $errstr");
        fputs($fp, "$domain\n");
        while (!feof($fp))
                echo fgets($fp, 2048);
        fclose($fp);
}
```

Blocking and non−blocking operations: But there's a problem with all those functions. They work fine if

1. You have a conenction with low latency and
2. If the server you're connecting to is up and running. If not, your script will be busy until it times out. The reason for this is that default socket connections are blocking and don't time out. You can avoid these "hanging scripts" by switching to non−blocking socket operations. The function set_socket_blocking() does just that: it set all operations on a socket (first parameter: socket pointer) to either blocking (second parameter: true) or false (second parameter: false). Using non−blocking

operations, the finger function would like like this:

```
        $fp = fsockopen($host, 79, &$errno, &$errstr) or die("$errno: [ ] $errstr");
        set_socket_blocking($fp, 0);
        fputs($fp, "$user\n");

        $stop = time() + $timeout;
        while (!feof($fp) && time() < $stop )
                echo fgets($fp, 128);
        fclose($fp);
```

Modifying these 3 functions to use non−blocking socket calls is left as an exercise for you.

16.Appendix F PostgreSQL Database Wrapper Examples

Submitted by: Joe Thong darkjoe@softhome.net Site URL: http://phpdb.linuxbox.com

Description: A PHP database wrapper for various database servers with a powerful Recordset for result data manipulation. Database results are flushed automatically by phpDB.

To get this file, in the web−browser, save this file as 'Text' type as phpDB−postgresql.lib

```
<?php
/*
Name: phpDB PostgreSQL module
Version: 1.02bR6
Description: A PHP database wrapper for various database
        servers with a powerful recordset for result data
        manipulation.  Database results are flushed
        automatically by phpDB.
*/

/* define this module, to prevent double class declaration.     */
if (!defined("_PHPDB_ABSTRACT_LAYER")) {
        define("_PHPDB_ABSTRACT_LAYER", 1 );
}
else
        return;

//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
         Class Name: phpDB
//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
class phpDB
{
        /*      public variables        */
        var $version = '1.02bR6'; // Version number of phpDB
        // This variable keeps what database type is going to
        // be used.  Current supported database server are
        // MySQL, MSQL, SQL Server, and Sybase
        var $databaseType = '';
        // Specifies which database is going to be used
        var $databaseName = '';
        // The hostname of the database server, port
        // number is optional.  e.g: "db.devNation.com"
        var $hostname = '';
        var $username = ''; // used to connect to the database server
        var $password = ''; // Password for the username

        // Private variables −−−−−− starts with underscore
        // An array of executed querys. For results cleanup purposes.
        var $_queryIDList = array();
        // The returned link identifier whenever a
        // successful database connection is made
        var $_connectionID      = −1;
        // A variable which was used to keep the returned
        // last error message.  The value will then returned
        // by the errorMsg() function
        var $_errorMsg = '';

        // This variable keeps the last created result
        // link identifier
        var $_queryID = −1;
        // A boolean variable to state whether its a persistent
        // connection or normal connection
        var     $_isPersistentConnection = false;
        // Holds the newly created result object,
        // returned via the execute() method
        var $_tempResultObj = '';

        // A constructor function for the phpDB object.
        // When initializing, specify the dbType i.e: "mysql",
        // "msql", "postgresql", "mssql", and "sybase"
        function phpDB($dbType = "postgresql")
        {
                switch ($dbType) {
                        case "mysql":
                        case "msql":
                        case "postgresql":
                        case "mssql":
                        case "sybase":
                        case "informix":
                                $this−>databaseType = $dbType;
                                break;
                        default:
                                return false;
                }
        }

        // Returns: A positive link identifier on success, or
        // false on error.      Connect to the server with the provided
        // arguments. The connection to the server will be closed
        // when the script terminates, unless close() function is
        // called beforehand
        function connect($argHostname = "", $argUsername = "",
                $argPassword = "", $argDatabaseName = "")
        {
                $connString = "";
                $hostPieces = array();
                /*      Must specify the database argument      */
                if (!$argDatabaseName) {
                        return false;
                }
                if ($argHostname != "") {
                        $this−>hostname = $argHostname;
                }
                if ($argUsername != "") {
                        $this−>username = $argUsername;
                }
                if ($argPassword != "") {
                        $this−>password = $argPassword;
                }
                if ($argDatabaseName != "") {
                        $this−>databaseName = $argDatabaseName;
                }
                $hostPieces = split(":", $this−>hostname);
                if ($hostPieces[0]) {
                        $connString .= "host=$hostPieces[0]";
                        if (isset($hostPieces[1])) {
                                $connString .= " port=$hostPieces[1]";
                        }
                }
            if ($this−>username) {
                        $connString .= " user=$this−>username";
                }
            if ($this−>password) {
                        $connString .= " password=$this−>password";
                }
                $connString .= " dbname=$this−>databaseName";

                $this−>_connectionID = @pg_Connect($connString);
                return $this−>_connectionID;
        }

        // Returns: A positive link identifier on success, or
        // false on error.  Connect to the server with the
        // provided arguments. The connection to the server will
        // not be closed when the script terminates. Instead it
        // will be kept for later future use
        function pconnect($argHostname = "", $argUsername = "",
                $argPassword = "", $argDatabaseName = "")
        {
                $connString = "";
                $hostPieces = array();
                /*      Must specify the database argument      */
                if (!$argDatabaseName) {
                        return false;
                }
                if ($argHostname != "") {
                        $this−>hostname = $argHostname;
                }
                if ($argUsername != "") {
                        $this−>username = $argUsername;
                }
                if ($argPassword != "") {
                        $this−>password = $argPassword;
                }
                if ($argDatabaseName != "") {
                        $this−>databaseName = $argDatabaseName;
                }
                $hostPieces = split(":", $this−>hostname);
                if ($hostPieces[0]) {
                        $connString .= "host=$hostPieces[0]";
                        if (isset($hostPieces[1])) {
                                $connString .= " port=$hostPieces[1]";
                        }
                }
            if ($this−>username) {
                        $connString .= " user=$this−>username";
                }
            if ($this−>password) {
                        $connString .= " password=$this−>password";
                }
                $connString .= " dbname=$this−>databaseName";

                $this−>_connectionID = @pg_pConnect($connString);
                if ($this−>_connectionID) {
                        $this−>_isPersistentConnection = true;
                }
                return $this−>_connectionID;
        }

        // Returns: true on success, false on error Select
        // the database name to be used PostgreSQL
        // Note:        function Not available
        function selectDB($dbName) {
                        return false;
        }

        // Returns: the Recordset object disregard success
        // or failure Send the sql statement to the database server
        function execute($sql = "") {
                // Instantiate an object without considering whether
                // the query return any results or not
                $this−>_queryID = @pg_Exec($this−>_connectionID, $sql);
                $this−>_tempResultObj = new Recordset($this−>_queryID);
                $this−>_insertQuery($this−>_queryID);
                return $this−>_tempResultObj;
        }

        // Returns: the last error message from previous
        // database operation
        function errorMsg() {
                $this−>_errorMsg = @pg_errormessage($this−>_connectionID);
            return $this−>_errorMsg;
        }

        // Returns: true on success, false on failure
        // Close the database connection
        function close() {
                if ($this−>_queryIDList && sizeof($this−>_queryIDList > 0)) {
                        while(list($_key, $_resultID) = each($this−>_queryIDList)) {
                                @pg_freeresult($_resultID);
                        }
                }
                // If its not a persistent connection, then
                // only the connection needs to be closed
                if ($this−>_isPersistentConnection != true) {
                        return @pg_close($this−>_connectionID);
                }
                else {
                        return true;
                }
        }

        // A PRIVATE function used by the constructor function
        // of the query object.  insert the successful returned
        // query id to the query id list.  Used for later results
        // cleanup.  A private function that's never meant to
        // be used directly
```

```
        function _insertQuery($query_id) {
                $this−>_queryIDList[] = $query_id;
        }
}

//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
         Class Name: Recordset
//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
class Recordset
{
        /*      public variables        */
        var $fields;
        // indicates that the current record position is before
        // the first record in a Recordset object
        var $BOF = null;

        // indicates that the current record position is after
        // the last record in a Recordset object
        var $EOF = null;

        /*      private variables       */
        var $_numOfRows = −1; // NEVER change the value! READ−ONLY!
        var $_numOfFields = −1; // NEVER change the value! READ−ONLY!
        // Holds anything that was returned from the database specific functions
        var $_tempResult = '';
        // This variable keeps the result link identifier
        var $_queryID = −1;
        // This variable keeps the current row in the Recordset
        var $_currentRow = −1;

        // Returns: query id on success and false if
        // failed Constructor function
        function Recordset($queryID)
        {
                $this−>_queryID = $queryID;
                if ($queryID) {
                        $this−>_numOfRows = @pg_numrows($this−>_queryID);
                        /*      pg_numrows() returns −1 on error        */
                        if ($this−>_numOfRows == −1) {
                                $this−>_numOfRows = 0;
                        }
                        $this−>_numOfFields = @pg_numfields($this−>_queryID);
                        /*      pg_numfields() returns −1 on error      */
                        if ($this−>_numOfFields == −1) {
                                $this−>_numOfFields = 0;
                        }
                }
                else {
                        $this−>_numOfRows = 0;
                        $this−>_numOfFields = 0;
                }
                /*      If result set contains rows     */
                if ($this−>_numOfRows > 0 && $this−>_currentRow == −1) {
                        $this−>_currentRow = 0;
                        $this−>fields = @pg_fetch_array($this−>_queryID, $this−>_currentRow);
                        $this−>EOF = false;
                        $this−>BOF = false;
                }
                return $this−>_queryID;
        }

        // Returns: true if successful, false if fail Set the Recordset
        // pointer to a specified field offset. If the next call to
        // fetchField() won't include a field offset, this field would
        // be returned.  PostgreSQL Note:       function Not available
        function fieldSeek($fieldOffset = −1) {
                $this−>_tempResult = false;
                return $this−>_tempResult;
        }

        // Returns: an object containing field information.  Get column
        // information in the Recordset object. fetchField() can be used
        // in order to obtain information about fields in a certain query
        // result. If the field offset isn't specified, the next field
        // that wasn't yet retrieved by fetchField() is retrieved.
        // PostgreSQL Note:     function Not available
        function fetchField($fieldOffset = −1) {
                $this−>_tempResult = false;
                return $this−>_tempResult;
        }

        // Returns: true if there still rows available, or false if there
        // are no more rows.  Moves to the next row in a specified Recordset
        // object and makes that record the current row and the data
        // corresponding to the row will be retrieved into the fields
        // collection.  Note: Unlike the moveRow() method, when _currentRow
        // is getNumOfRows() − 1, EOF will immediately be true.  If row number
        // is not provided, the function will point to the
        // first row automatically
        function nextRow() {
                if ($this−>getNumOfRows() > 0) {
                        $this−>fields = array();
                        $this−>_currentRow++;
                        $this−>fields = @pg_fetch_array($this−>_queryID, $this−>_currentRow);
                        /*      This is not working.  True all the time */
                        if ($this−>fields) {
                                $this−>_checkAndChangeEOF($this−>_currentRow − 1);
                                return true;
                        }
                }
                $this−>EOF = true;
                return false;
        }

        // Returns: true on success, false on failure moveRow() moves
        // the internal row pointer of the Recordset object to point
        // to the specified row number and the data corresponding to
        // the row will be retrieved into the fields collection.  If
        // row number is not provided, the function will point to
        // the first row automatically
        function moveRow($rowNumber = 0) {
                if ($rowNumber == 0) {
                        return $this−>firstRow();
                }
                else if ($rowNumber == ($this−>getNumOfRows() − 1)) {
                        return $this−>lastRow();
                }
                if ($this−>getNumOfRows() > 0 && $rowNumber < $this−>getNumOfRows()) {
                        $this−>fields = null;
                        $this−>_currentRow = $rowNumber;
                        $this−>fields = @pg_fetch_array($this−>_queryID, $this−>_currentRow);
                        /*      This is not working.  True all the time */
                        if ($this−>fields) {
                                // No need to call _checkAndChangeEOF() because
                                // the possibility of moving to the last row
                                // has been handled by the above code
                                $this−>EOF = false;
                                return true;
                        }
                }
                $this−>EOF = true;
                return false;
        }

        // Returns: true on success, false on failure firstRow()
        // moves the internal row pointer of the Recordset object
        // to the first row and the data corresponding to the row
        // will be retrieved into the fields collection
        function firstRow() {
                if ($this−>getNumOfRows() > 0) {
                        $this−>fields = array();
                        $this−>_currentRow = 0;
                        $this−>fields = @pg_fetch_array($this−>_queryID, $this−>_currentRow);
                        $this−>EOF = true;
                        /*      This is not working.  True all the time */
                        if ($this−>fields) {
                                return true;
                        }
                }
                $this−>EOF = true;
                return false;
        }

        // Returns: true on success, false on failure lastRow()
        // moves the internal row pointer of the Recordset object
        // to the last row and the data corresponding to the row
        // will be retrieved into the fields collection
        function lastRow() {
                if ($this−>getNumOfRows() > 0) {
                        $this−>fields = array();
                        $num_of_rows = $this−>getNumOfRows();
                        /*      $num_of_rows decemented at above        */
                        $this−>_currentRow = −−$num_of_rows;
                        $this−>fields = @pg_fetch_array($this−>_queryID, $this−>_currentRow);
                        /*      This is not working.  True all the time */
                        if ($this−>fields) {
                                /*      Special case for making EOF false.      */
                                $this−>EOF = false;
                                return true;
                        }
                }
                $this−>EOF = true;
                return false;
        }

        // close() only needs to be called if you are worried about
        // using too much memory while your script is running. All
        // associated result memory for the specified result identifier
        // will automatically be freed
        function close() {
                $this−>_tempResult = @pg_freeresult($this−>_queryID);
                return $this−>_tempResult;
        }

        // Returns: the number of rows in a result set.
        // Get number of rows in result
        function getNumOfRows() {
                return $this->_numOfRows;
        }

        // Returns: the number of fields in a result set.
        // Get number of fields in result
        function getNumOfFields() {
                return $this->_numOfFields;
        }

        /*      Check and change the status of EOF.     */
        function _checkAndChangeEOF($currentRow) {
                if ($currentRow >= ($this->_numOfRows - 1)) {
                        $this->EOF = true;
                }
                else {
                        $this->EOF = false;
                }
        }
}
?>
```

17.Appendix G Microsoft SQL Server DB Wrapper Example

Submitted by: Joe Thong darkjoe@softhome.net Site URL: http://phpdb.linuxbox.com

Description: A PHP database wrapper for various database servers with a powerful Recordset for result data manipulation. Database results are flushed automatically by phpDB.

To get this file, in the web−browser, save this file as 'Text' type as phpDB−mssql.lib

---

```
<?php
/*
Name: phpDB Microsoft SQL Server module
Version: 1.02bR6
Description: A PHP database wrapper for various
        database servers with a powerful
        Recordset for result data manipulation.  Database
        results are flushed automatically by phpDB.
*/
// Define this module, to prevent double class declaration
if (!defined("_PHPDB_ABSTRACT_LAYER"))
{
        define("_PHPDB_ABSTRACT_LAYER", 1 );
}
else
        return;

//-------------------------------
          Class Name: phpDB
//-------------------------------
class phpDB
{
        // public variables
        var $version = '1.02bR6'; // Version number of phpDB
        // This variable keeps what database type is going
        // to be used.  Current supported database server
        // are MySQL, MSQL, SQL Server, PostgreSQL and Sybase
        var $databaseType = '';
        var $databaseName = ''; // Specifies which database is going to be used

        // The hostname of the database server, port
        // number is optional.  e.g: "db.devNation.com"
        var $hostname = '';

        var $username = ''; // to connect to the database server
        var $password = '';     // Password for the username

        // Private variables −−−−− starts with underscore

        // An array of executed querys. For results cleanup purposes
        var $_queryIDList = array();

        // The returned link identifier whenever a
        // successful database connection is made
        var $_connectionID      = −1;

        // A variable which was used to keep the returned last
        // error message.  The value will then returned
        // by the errorMsg() function
        var $_errorMsg = '';
        // This variable keeps the last created result link identifier
        var $_queryID = −1;

        // A boolean variable to state whether its a
        // persistent connection or normal connection
        var     $_isPersistentConnection = false;

        // Holds the newly created result object, returned
        // via the execute() method
        var $_tempResultObj = '';

        //      A constructor function for the phpDB object.
        // When initializing, specify the dbType i.e: "mysql",
        // "msql", "postgresql", "mssql", and "sybase"
        function phpDB($dbType = "mssql")
        {
                switch ($dbType)
                {
                        case "mysql":
                        case "msql":
                        case "postgresql":
                        case "mssql":
                        case "sybase":
                        case "informix":
                                $this−>databaseType = $dbType;
                                break;
                        default:
                                return false;
                }
        }

        // Returns: A positive link identifier on success,
        // or false on error.   Connect to the server with
        // the provided arguments. The connection to the server
        // will be closed when the script terminates, unless
        // close() function is called beforehand.
        function connect($argHostname = "", $argUsername = "",
                $argPassword = "", $argDatabaseName = "")
        {
                $boolDBSelected;
                if ($argHostname != "") {
                        $this->hostname = $argHostname;
                }
                if ($argUsername != "") {
                        $this->username = $argUsername;
                }
                if ($argPassword != "") {
                        $this->password = $argPassword;
                }
                if ($argDatabaseName != "") {
                        $this->databaseName = $argDatabaseName;
                }

                $this->_connectionID = @mssql_connect($this->hostname, $this->username, $this->pa

                if ($this->databaseName && $this->_connectionID) {
                        $boolDBSelected = @mssql_select_db($this->databaseName);
                        if(!$boolDBSelected) {  /*      If DB selection fails   */
                                @mssql_close($this->_connectionID);     /*      Close the current
                                return false;
                        }
                }
                return $this->_connectionID;
        }

        // Returns: A positive link identifier on success, or
        // false on error Connect to the server with the provided
        // arguments. The connection to the server will not be closed
        // when the script terminates. Instead it will be kept for
        // later future use
        function pconnect($argHostname = "", $argUsername = "",
                $argPassword = "", $argDatabaseName = "")
        {
                $boolDBSelected;
                if ($argHostname != "") {
                        $this->hostname = $argHostname;
                }
                if ($argUsername != "") {
                        $this->username = $argUsername;
                }
                if ($argPassword != "") {
                        $this->password = $argPassword;
                }
                if ($argDatabaseName != "") {
                        $this->databaseName = $argDatabaseName;
                }

                $this->_connectionID = @mssql_pconnect($this->hostname, $this->username, $this->p
                if ($this->_connectionID) {
                        $this->_isPersistentConnection = true;
                }

                if ($this->databaseName && $this->_connectionID) {
                        $boolDBSelected = @mssql_select_db($this->databaseName);
                        if(!$boolDBSelected) {  /*      if DB selection fails   */
                                return false;   /*      Persistent connection can't be closed   *
                        }
                }
                return $this−>_connectionID;
        }

        //      Returns: true on success, false on error Select the
        // database name to be used
        function selectDB($dbName)
        {
                $this−>databaseName = $dbName;
                if ($this−>_connectionID) {
                        return @mssql_select_db($dbName);
                }
                else {
                        /*      No database selected    */
                        return false;
                }
        }

        // Returns: the Recordset object disregard success or
        // failure Send the sql statement to the database server
        function execute($sql = "")
        {
                $this−>_queryID = @mssql_query($sql, $this−>_connectionID);
                // Instantiate an object without considering whether
                // the query return any results or not
                $this−>_tempResultObj = new Recordset($this−>_queryID);
                $this−>_insertQuery($this−>_queryID);
                return $this−>_tempResultObj;
        }

        // Returns: the last error message from previous database
        // operation Note: This function is NOT available for
        // Microsoft SQL Server
        function errorMsg()
        {
                $this−>_errorMsg = "errorMsg() is not available for Microsoft SQL Server";
            return $this−>_errorMsg;
        }

        /*      Returns: true on success, false on failure
                Close the database connection.  */
        function close() {
                if ($this−>_queryIDList && sizeof($this−>_queryIDList > 0)) {
                        while(list($_key, $_resultID) = each($this−>_queryIDList)) {
                                @mssql_free_result($_resultID);
                        }
                }
                // If its not a persistent connection, then
                // only the connection needs to be closed
                if ($this−>_isPersistentConnection != true) {
                        return @mssql_close($this−>_connectionID);
                }
                else {
                        return true;
                }
        }

        // A PRIVATE function used by the constructor function of
        // the query object.  insert the successful returned
        // query id to the query id list.  Used for later results
        // cleanup.  A private function that's never meant to be
        // used directly
        function _insertQuery($query_id) {
                $this−>_queryIDList[] = $query_id;
        }
}

//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
         Class Name: Recordset
//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
class Recordset
{
        /*      public variables        */
        var $fields;
        // indicates that the current record position is
        // before the first record in a Recordset object
        var $BOF = null;
        // indicates that the current record position is
        // after the last record in a Recordset object
        var $EOF = null;

        // Private variables
        var $_numOfRows = −1; // NEVER change the value!  READ−ONLY!
        var $_numOfFields = −1; // NEVER change the value!  READ−ONLY!

        // Holds anything that was returned from the
        // database specific functions
        var $_tempResult = '';
        // This variable keeps the result link identifier
        var $_queryID = −1;
        // This variable keeps the current row in the Recordset
        var $_currentRow = −1;

        // Returns: query id on success and false if
        // failed Constructor function
        function Recordset($queryID)
        {
                $this−>_queryID = $queryID;
                if ($queryID) {
                        $this−>_numOfRows = @mssql_num_rows($this−>_queryID);
                        $this−>_numOfFields = @mssql_num_fields($this−>_queryID);
                }
                else {
                        $this−>_numOfRows = 0;
                        $this−>_numOfFields = 0;
                }
                // If result set contains rows
                if ($this−>_numOfRows > 0 && $this−>_currentRow == −1) {
                        $this−>_currentRow = 0;
                        $this−>fields = @mssql_fetch_array($this−>_queryID);
                        $this−>EOF = false;
                        $this−>BOF = false;
                }
                return $this−>_queryID;
        }

        // Returns: true if successful, false if fail Set
        // the Recordset pointer to a specified field offset.
        // If the next call to fetchField() won't include a
        // field offset, this field would be returned
        function fieldSeek($fieldOffset = −1) {
                $this−>_tempResult = @mssql_field_seek($this−>_queryID, $fieldOffset);
                return $this−>_tempResult;
        }
        // Returns: an object containing field information.
        // Get column information in the Recordset object.
        // fetchField() can be used in order to obtain information
        // about fields in a certain query result. If the field
        // offset isn't specified, the next field that wasn't yet
        // retrieved by fetchField() is retrieved
        function fetchField($fieldOffset = −1) {
                if ($fieldOffset != −1) {
                        $this−>_tempResult = @mssql_fetch_field($this−>_queryID, $fieldOffset);
                }
                // The $fieldOffset argument is not provided thus its −1
                else if ($fieldOffset == −1) {
                        $this−>_tempResult = @mssql_fetch_field($this−>_queryID);
                }
                return $this−>_tempResult;
        }

        // Returns: true if there still rows available, or false
        // if there are no more rows.  Moves to the next row in a
        // specified Recordset object and makes that record the current
        // row and the data corresponding to the row will be retrieved
        // into the fields collection.  Note: Unlike the moveRow() method,
        // when _currentRow is getNumOfRows() − 1, EOF will immediately be
        // true.  If row number is not provided, the function will point
        // to the first row automatically
        function nextRow()
        {
                if ($this−>getNumOfRows() > 0) {
                        $this−>fields = array();
                        $this−>_currentRow++;
                        $this−>fields = @mssql_fetch_array($this−>_queryID);
                        // This is not working.  True all the time
                        if ($this−>fields) {
                                $this−>_checkAndChangeEOF($this−>_currentRow − 1);
                                return true;
                        }
                }
                $this−>EOF = true;
                return false;
        }

        // Returns: true on success, false on failure moveRow()
        // moves the internal row pointer of the Recordset object
        // to point to the specified row number and the data
        // corresponding to the row will be retrieved into the fields
        // collection.  If row number is not provided, the function will
        // point to the first row automatically
        function moveRow($rowNumber = 0)
        {
                if ($rowNumber == 0) {
                        return $this−>firstRow();
                }
                else if ($rowNumber == ($this−>getNumOfRows() − 1)) {
                        return $this−>lastRow();
                }
                if ($this−>getNumOfRows() > 0 && $rowNumber < $this−>getNumOfRows()) {
                        $this−>fields = null;
                        $this−>_currentRow = $rowNumber;
                        if(@mssql_data_seek($this−>_queryID, $this−>_currentRow)) {
                                $this−>fields = @mssql_fetch_array($this−>_queryID);
                                /*      This is not working.  True all the time */
                                if ($this−>fields) {
```

```
                                // No need to call _checkAndChangeEOF() because
                                // the possibility of moving to the last row has
                                // been handled by the above code
                                $this−>EOF = false;
                                return true;
                        }
                }
        }
        $this−>EOF = true;
        return false;
}

// Returns: true on success, false on failure firstRow() moves
// the internal row pointer of the Recordset object to the first
// row and the data corresponding to the row will be retrieved
// into the fields collection
function firstRow()
{
        if ($this−>getNumOfRows() > 0) {
                $this−>fields = array();
                $this−>_currentRow = 0;
                if (@mssql_data_seek($this−>_queryID, $this−>_currentRow)) {
                        $this−>fields = @mssql_fetch_array($this−>_queryID);
                        $this−>EOF = false;
                        /*      This is not working.  True all the time */
                        if ($this−>fields) {
                                return true;
                        }
                }
        }
        $this−>EOF = true;
        return false;
}

// Returns: true on success, false on failure lastRow() moves
// the internal row pointer of the Recordset object to the last
// row and the data corresponding to the row will be retrieved
// into the fields collection
function lastRow()
{
        if ($this−>getNumOfRows() > 0) {
                $this−>fields = array();
                $num_of_rows = $this−>getNumOfRows();
                $this−>_tempResult = @mssql_data_seek($this−>_queryID, −−$num_of_rows);
                if ($this−>_tempResult) {
                        /*      $num_of_rows decemented at above        */
                        $this−>_currentRow = $num_of_rows;
                        $this−>fields = @mssql_fetch_array($this−>_queryID);
                        /*      This is not working.  True all the time */
                        if ($this−>fields) {
                                /*      Special case for making EOF false.      */
                                $this−>EOF = false;
                                return true;
                        }
                }
        }
        $this−>EOF = true;
        return false;
}

// close() only needs to be called if you are worried about using
// too much memory while your script is running. All associated
```

```
        // result memory for the specified result identifier will
        // automatically be freed
        function close() {
                $this−>_tempResult = @mssql_free_result($this−>_queryID);
                return $this−>_tempResult;
        }

        // Returns: the number of rows in a result set.  Get
        // number of rows in result
        function getNumOfRows() {
                return $this−>_numOfRows;
        }

        /*      Returns: the number of fields in a result set.
        Get number of fields in result. */
        function getNumOfFields() {
                return $this−>_numOfFields;
        }

        /*      Check and change the status of EOF.     */
        function _checkAndChangeEOF($currentRow) {
                if ($currentRow >= ($this−>_numOfRows − 1)) {
                        $this−>EOF = true;
                }
                else {
                        $this−>EOF = false;
                }
        }
}
?>
```

18.Appendix H Sybase SQL Server DB Wrapper Example

Submitted by: Joe Thong darkjoe@softhome.net Site URL: http://phpdb.linuxbox.com

Description: A PHP database wrapper for various database servers with a powerful Recordset for result data manipulation. Database results are flushed automatically by phpDB.

To get this file, in the web−browser, save this file as 'Text' type as phpDB−sybase.lib

---

```
<?php
/*
Name: phpDB Sybase module
Version: 1.02bR6
Description: A PHP database wrapper for various database
        servers with a powerful Recordset for result data
        manipulation.  Database results are flushed
        automatically by phpDB.
*/

// Define this module, to prevent double class declaration
if (!defined("_PHPDB_ABSTRACT_LAYER")) {
        define("_PHPDB_ABSTRACT_LAYER", 1 );
}
else
        return;
```

```
//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
         Class Name: phpDB
//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
class phpDB
{
        /*      public variables        */
        var $version = '1.02bR6'; // Version number of phpDB
        // This variable keeps what database type is going
        // to be used.  Current supported database server
        // are MySQL, MSQL, SQL Server, and Sybase
        var $databaseType = '';
        // Specifies which database is going to be used
        var $databaseName = '';
        // The hostname of the database server, port number
        // is optional.  e.g: "db.devNation.com"
        var $hostname = '';
        var $username = '';     // to connect to the database server
        var $password = ''; // Password for the username

        // Private variables −−− starts with underscore
        // An array of executed querys.  For results
        // cleanup purposes
        var $_queryIDList       = array();
        // The returned link identifier whenever a successful
        // database connection is made
        var $_connectionID      = −1;
        // A variable which was used to keep the returned last
        // error message.  The value will then returned by
        // the errorMsg() function
        var $_errorMsg = '';
        // This variable keeps the last created result
        // link identifier
        var $_queryID = −1;
        // A boolean variable to state whether its a
        // persistent connection or normal connection
        var     $_isPersistentConnection = false;
        // Holds the newly created result object, returned
        // via the execute() method
        var $_tempResultObj = '';

        // A constructor function for the phpDB object.  When
        // initializing, specify the dbType i.e: "mysql",
        // "msql", "postgresql", "mssql", and "sybase"
        function phpDB($dbType = "sybase")
        {
                switch ($dbType) {
                        case "mysql":
                        case "msql":
                        case "postgresql":
                        case "mssql":
                        case "sybase":
                        case "informix":
                                $this−>databaseType = $dbType;
                                break;
                        default:
                                return false;
                }
        }

        // Returns: A positive link identifier on success, or
        // false on error.      Connect to the server with the
        // provided arguments. The connection to the server will be
        // closed when the script terminates, unless close()
        // function is called beforehand
        function connect($argHostname = "", $argUsername = "",
                $argPassword = "", $argDatabaseName = "")
        {
                $boolDBSelected;
                if ($argHostname != "") {
                        $this−>hostname = $argHostname;
                }
                if ($argUsername != "") {
                        $this−>username = $argUsername;
                }
                if ($argPassword != "") {
                        $this−>password = $argPassword;
                }
                if ($argDatabaseName != "") {
                        $this−>databaseName = $argDatabaseName;
                }

                $this−>_connectionID = @sybase_connect($this−>hostname, $this−>username, $this−>p

                if ($this−>databaseName && $this−>_connectionID) {
                        $boolDBSelected = @sybase_select_db($this−>databaseName);
                        /*      If DB selection fails   */
                        if(!$boolDBSelected) {
                                /*      Close the current connection    */
                                @sybase_close($this−>_connectionID);
                                return false;
                        }
                }
                return $this−>_connectionID;
        }

        // Returns: A positive link identifier on success, or false
        // on error.  Connect to the server with the provided
        // arguments. The connection to the server will not be closed
        // when the script terminates. Instead it will be kept for later future use
        function pconnect($argHostname = "", $argUsername = "",
                $argPassword = "", $argDatabaseName = "")
        {
                $boolDBSelected;
                if ($argHostname != "") {
                        $this−>hostname = $argHostname;
                }
                if ($argUsername != "") {
                        $this−>username = $argUsername;
                }
                if ($argPassword != "") {
                        $this−>password = $argPassword;
                }
                if ($argDatabaseName != "") {
                        $this−>databaseName = $argDatabaseName;
                }

                $this−>_connectionID = @sybase_pconnect($this−>hostname, $this−>username, $this−>
                if ($this−>_connectionID) {
                        $this−>_isPersistentConnection = true;
                }

                if ($this−>databaseName && $this−>_connectionID) {
                        $boolDBSelected = @sybase_select_db($this−>databaseName);
                        /*      if DB selection fails   */
                        if(!$boolDBSelected) {
                                /*      Persistent connection can't be closed   */
                                return false;
                        }
                }
                return $this−>_connectionID;
        }

        /*      Returns: true on success, false on error
                Select the database name to be used     */
        function selectDB($dbName) {
                $this−>databaseName = $dbName;
                if ($this−>_connectionID) {
                        return @sybase_select_db($dbName);
                }
                else {
                        /*      No database selected    */
                        return false;
                }
        }

        /*      Returns: the Recordset object disregard success or failure
                Send the sql statement to the database server.  */
        function execute($sql = "") {
                $this−>_queryID = @sybase_query($sql, $this−>_connectionID);
                // Instantiate an object without considering whether
                // the query return any results or not
                $this−>_tempResultObj = new Recordset($this−>_queryID);
                $this−>_insertQuery($this−>_queryID);
                return $this−>_tempResultObj;
        }

        /*      Returns: the last error message from previous database operation
                Note: This function is NOT available for Sybase.        */

        function errorMsg() {
                $this−>_errorMsg = "errorMsg() is not available for Sybase";
            return $this−>_errorMsg;
        }

        /*      Returns: true on success, false on failure
                Close the database connection.  */

        function close() {
                if ($this−>_queryIDList && sizeof($this−>_queryIDList > 0)) {
                        while(list($_key, $_resultID) = each($this−>_queryIDList)) {
                                @sybase_free_result($_resultID);
                        }
                }
                // If its not a persistent connection, then
                // only the connection needs to be closed
                if ($this−>_isPersistentConnection != true) {
                        return @sybase_close($this−>_connectionID);
                }
                else {
                        return true;
                }
        }

        // A PRIVATE function used by the constructor function
        // of the query object.  insert the successful returned
```

```
        // query id to the query id list.  Used for later results
        // cleanup.  A private function that's never meant
        // to be used directly
        function _insertQuery($query_id) {
                $this−>_queryIDList[] = $query_id;
        }
}

//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
         Class Name: Recordset
//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
class Recordset
{
        /*      public variables        */
        var $fields;
        // indicates that the current record position is
        // before the first record in a Recordset object
        var $BOF = null;
        // indicates that the current record position
        // is after the last record in a Recordset object
        var $EOF = null;

        // Private variables − starts with underscore
        var $_numOfRows = −1; // NEVER change the value! READ−ONLY!
        var $_numOfFields = −1; // NEVER change the value! READ−ONLY!
        // Holds anything that was returned from
        // the database specific functions
        var $_tempResult = '';
        // This variable keeps the result link identifier
        var $_queryID = −1;
        // This variable keeps the current row in the Recordset
        var $_currentRow = −1;

        // Returns: query id on success and false if
        // failed Constructor function
        function Recordset($queryID) {
                $this−>_queryID = $queryID;
                if ($queryID) {
                        $this−>_numOfRows = @sybase_num_rows($this−>_queryID);
                        $this−>_numOfFields = @sybase_num_fields($this−>_queryID);
                }
                else {
                        $this−>_numOfRows = 0;
                        $this−>_numOfFields = 0;
                }
                /*      If result set contains rows     */
                if ($this−>_numOfRows > 0 && $this−>_currentRow == −1) {
                        $this−>_currentRow = 0;
                        $this−>fields = @sybase_fetch_array($this−>_queryID);
                        $this−>EOF = false;
                        $this−>BOF = false;
                }
                return $this−>_queryID;
        }

        // Returns: true if successful, false if fail Set
        // the Recordset pointer to a specified field offset.
        // If the next call to fetchField() won't include a
        // field offset, this field would be returned
        function fieldSeek($fieldOffset = −1) {
                $this−>_tempResult = @sybase_field_seek($this−>_queryID, $fieldOffset);
                return $this−>_tempResult;
        }

        // Returns: an object containing field information.
        // Get column information in the Recordset object.
        // fetchField() can be used in order to obtain information
        // about fields in a certain query result. If the field
        // offset isn't specified, the next field that wasn't yet
        // retrieved by fetchField() is retrieved
        function fetchField($fieldOffset = −1) {
                if ($fieldOffset != −1) {
                        $this−>_tempResult = @sybase_fetch_field($this−>_queryID, $fieldOffset);
                }
                /*      The $fieldOffset argument is not provided thus its −1   */
                else if ($fieldOffset == −1) {
                        $this−>_tempResult = @sybase_fetch_field($this−>_queryID);
                }
                return $this−>_tempResult;
        }

        // Returns: true if there still rows available, or
        // false if there are no more rows.  Moves to the next
        // row in a specified Recordset object and makes that record
        // the current row and the data corresponding to the row will
        // be retrieved into the fields collection.  Note: Unlike
        // the moveRow() method, when _currentRow is getNumOfRows() − 1,
        // EOF will immediately be true.  If row number is not
        // provided, the function will point to the
        // first row automatically
        function nextRow() {
                if ($this−>getNumOfRows() > 0) {
                        $this−>fields = array();
                        $this−>_currentRow++;
                        $this−>fields = @sybase_fetch_array($this−>_queryID);
                        /*      This is not working.  True all the time */
                        if ($this−>fields) {
                                $this−>_checkAndChangeEOF($this−>_currentRow − 1);
                                return true;
                        }
                }
                $this−>EOF = true;
                return false;
        }

        // Returns: true on success, false on failure moveRow()
        // moves the internal row pointer of the Recordset object
        // to point to the specified row number and the data
        // corresponding to the row will be retrieved into the
        // fields collection.  If row number is not provided, the
        // function will point to the first row automatically
        function moveRow($rowNumber = 0) {
                if ($rowNumber == 0) {
                        return $this−>firstRow();
                }
                else if ($rowNumber == ($this−>getNumOfRows() − 1)) {
                        return $this−>lastRow();
                }
                if ($this−>getNumOfRows() > 0 && $rowNumber < $this−>getNumOfRows()) {
                        $this−>fields = null;
                        $this−>_currentRow = $rowNumber;
                        if(@sybase_data_seek($this−>_queryID, $this−>_currentRow)) {
                                $this−>fields = @sybase_fetch_array($this−>_queryID);
                                /*      This is not working.  True all the time */
                                if ($this->fields) {
                                        // No need to call _checkAndChangeEOF()
                                        // because the possibility of moving to the
                                        // last row has been handled by the above code
                                        $this->EOF = false;
                                        return true;
                                }
                        }
                }
                $this->EOF = true;
                return false;
        }

        // Returns: true on success, false on failure firstRow()
        // moves the internal row pointer of the Recordset object
        // to the first row and the data corresponding to the row
        // will be retrieved into the fields collection
        function firstRow() {
                if ($this->getNumOfRows() > 0) {
                        $this->fields = array();
                        $this->_currentRow = 0;
                        if (@sybase_data_seek($this->_queryID, $this->_currentRow)) {
                                $this->fields = @sybase_fetch_array($this->_queryID);
                                $this->EOF = false;
                                /*      This is not working.  True all the time */
                                if ($this->fields) {
                                        return true;
                                }
                        }
                }
                $this->EOF = true;
                return false;
        }

        // Returns: true on success, false on failure lastRow()
        // moves the internal row pointer of the Recordset object
        // to the last row and the data corresponding to the row
        // will be retrieved into the fields collection
        function lastRow() {
                if ($this->getNumOfRows() > 0) {
                        $this->fields = array();
                        $num_of_rows = $this->getNumOfRows();
                        $this->_tempResult = @sybase_data_seek($this->_queryID, --$num_of_rows);
                        if ($this->_tempResult) {
                                /*      $num_of_rows decemented at above        */
                                $this->_currentRow = $num_of_rows;
                                $this->fields = @sybase_fetch_array($this->_queryID);
                                /*      This is not working.  True all the time */
                                if ($this->fields) {
                                        /*      Special case for making EOF false.      */
                                        $this->EOF = false;
                                        return true;
                                }
                        }
                }
                $this->EOF = true;
                return false;
        }

        // close() only needs to be called if you are worried
        // about using too much memory while your script is
        // running. All associated result memory for the
```

```
        // specified result identifier will automatically be freed
        function close() {
                $this->_tempResult = @sybase_free_result($this->_queryID);
                return $this->_tempResult;
        }

        /*      Returns: the number of rows in a result set.
        Get number of rows in result.   */

        function getNumOfRows() {
                return $this->_numOfRows;
        }

        /*      Returns: the number of fields in a result set.
        Get number of fields in result. */

        function getNumOfFields() {
                return $this->_numOfFields;
        }

        /*      Check and change the status of EOF.     */
        function _checkAndChangeEOF($currentRow) {
                if ($currentRow >= ($this->_numOfRows - 1)) {
                        $this->EOF = true;
                }
                else {
                        $this->EOF = false;
                }
        }
}
?>
```

19.Appendix I phpDB.inc Example

Submitted by: Joe Thong darkjoe@softhome.net Site URL: http://phpdb.linuxbox.com

Description: A PHP database wrapper for various database servers with a powerful Recordset for result data manipulation. Database results are flushed automatically by phpDB.

To get this file, in the web−browser, save this file as 'Text' type as phpDB.inc

---

```
<?php
/*
Name: phpDB General module
Version: 1.02bR6
Description: A PHP database wrapper for various
        database servers.  Database results are flushed
        automatically by phpDB.  Supported database
        servers are MySQL, MSQL, PostgreSQL, Microsoft
        SQL Server and Sybase.
*/

        if (!defined("_PHPDB_GENERAL_LAYER")) {
          define("_PHPDB_GENERAL_LAYER", 1 );
        }
        else
```

```
                return;

        // Fill in the database server that you're
        // going to use.  Consult the phpDB Reference
        // Manual for more information
        $databaseType = '';
        // The phpDB module root path.  No trailing slash
        $phpDBRootPath = '.';

        function useDB($dbType = "")
        {
                GLOBAL $phpDBRootPath;
                switch (strtolower($dbType))
                {
                        case "mysql":
                        case "msql":
                        case "postgresql":
                        case "mssql":
                        case "sybase":
                        case "informix":
                                include("$phpDBRootPath". "/phpDB−" . "$dbType.lib");
                                break;
                        case "":
                                die("Please edit phpDB.inc in order to use phpDB");
                                return false;
                        default:
                                die("Invalid database selection");
                                return false;
                }
                return true;
        }

        useDB($databaseType);
?>
```

20.Appendix J phpDBTest.php3 Example

Submitted by: Joe Thong darkjoe@softhome.net Site URL: http://phpdb.linuxbox.com

Description: A PHP database wrapper for various database servers with a powerful Recordset for result data manipulation. Database results are flushed automatically by phpDB.

To get this file, in the web−browser, save this file as 'Text' type as phpDB−mssql.lib

```
<html>
< head>
        < title>Untitled< /title>
< /head>

< body>
<?php
        // Assumed this file is placed in the same directory with phpDB.inc
        include("phpDB.inc");
        $db = new phpDB();
        $db->pconnect("hostName", "userName", "passWord", "databaseName") or die ("Can't connect
        $rs = $db->execute("SELECT * FROM Items");
        $numOfRows = $rs−>getNumOfRows();
        echo "Number of Rows: $numOfRows";

        $rs−>firstRow(); // optional, but recommended
        while (!$rs−>EOF) {
                // Fields collection accessible as associative arrays too
                echo "<br>" . $rs−>fields[0];
                $rs−>nextRow(); // NOTE: nextRow() is placed at below
        }

        $rs−>close();
        $db−>close();   // optional
?>
< /body>
< /html>
```