

Craig S. Cmehil

The SAP® Developer's Guide to PHP

- ▶ Detailed guidance on integrating PHP applications with SAP systems
- ▶ Expert instructions to install and master the SAPRFC extension module
- ▶ A practical, sample application and comprehensive coding comments help ensure your success

The SAP Developer's Guide to PHP

Craig S. Cmehil

Contents

Preface	3	5 Foundation and Connection	29
1 Introduction to PHP	5	5.1 HTML and JavaScript	29
1.1 The History	5	5.2 CSS	32
1.2 Why PHP?	6	5.3 PHP	33
1.3 How PHP Works	6	6 Beta Testing and Enhancements	47
1.4 The Basics of Getting Started	8	6.1 Dynamic Login	47
2 SAPRFC and Environment	9	Cookies	47
2.1 A Merging of Technologies	9	Sessions	48
2.2 The Environment	10	6.2 Action Log	53
2.3 Setting Up and Installing		Better Display	53
the Environment	10	Modifying	53
HTTP Server	10	Deleting and Printing	56
Installing PHP	11	6.3 Modify Data	57
Manual Configuration of HTTP Server ..	13	6.4 "Bells & Whistles"	62
2.4 The First PHP webpage	15	Hiding Our Log	62
2.5 SAPRFC	16	Controlling the Display of the Log	65
3 Connecting PHP to SAP	19	CSS Styles	66
Using the Classes	20	A Appendix	71
Making the Connection	20	A.1 Chapter 5 Code	71
Calling the Function	21	A.2 Chapter 6 Code	80
Reading the Result	22	A.3 CSS Styles Used in Our Example	
Logging Off	23	Application	93
4 Example Application	25	A.4 Calendar Configuration Script	95
4.1 The Problem	25	Index	97
4.2 The Background	27		
4.3 The System	27		

Preface

Due to the growing interest in the development of solutions in a corporate environment at reduced costs, PHP is becoming increasingly more popular. The need for quick and simple guides to help developers and their employers get started, however, is problematic. PHP has many such guides that are available and easy to locate; for those of you working within the SAP environment, the information you need is readily accessible from multiple sources as well. What is lacking is what happens when two such technologies come together.

You have two distinct groups of people, developers in this case, who have little or no connection with one another. And here, as in other cases, connections are created out of patience and understanding. Whether you're a seasoned SAP developer or a developer just getting started, with the proper understanding and a few easy examples, you can build this connection.

Our goal in this book is to try to remove some of the guesswork in order to strengthen the ties between ABAP and PHP, until we finally have a solid connection.

We do not intend to teach you how to program or how to work with ABAP; rather, we hope to show you how you can use PHP to read, display, and access the ABAP side of your SAP system.

Note Some of the numerous listings in this book have been laid out in two-column format for printing reasons. To indicate where a line break is necessary because of the layout and so as not to be included in the actual code, we have inserted the "↵" character. Please pay particular attention to potential blanks that may appear before this character; rebreaking the following lines of code is only made for visual aspects.

1 Introduction to PHP

1.1 The History

In the Fall of 1994, Rasmus Lerdorf created what would later become known as the PHP programming language. Used almost entirely by Lerdorf for the tracking of information on his own home page, PHP (known then as "Personal Home Page Tools") was very limited in its capabilities, which were based primarily on Perl¹ scripts. As Lerdorf required more capabilities, he moved on to create PHP/FI or Personal Home Page Tools/Form Interpreter, which was a C implementation that allowed for database interaction and dynamic web page building. In 1995, Lerdorf provided the source code of PHP/FI to the world for code improvements and bug fixes.

PHP's future really began to solidify with the release of PHP/FI 2.0 in November of 1997. With an estimated following of thousands of users worldwide and approximately 1% of the domains on the Internet having it installed, PHP/FI 2.0 took the next big step—it was upgraded to PHP 3.0.

While working on an eCommerce University project, Andi Gutmans and Zeev Suraski found PHP/FI 2.0 to be

lacking and decided to rewrite the entire package. To build on the existing user base of PHP/FI and in a show of cooperation, Suraski and Gutmans joined Lerdorf and announced PHP 3.0 to be the next official release, thus halting the development of PHP/FI 2.0.

With public testing of PHP 3.0 underway since its launch in June of 1998, Suraski and Gutmans began to rewrite the core of PHP. This rewrite produced the Zend Engine, and the two moved on to found Zend Technologies.² Zend has been present now for ever major advancement of PHP, the latest being PHP 5.0, released on the Zend Engine II on July 13, 2004. PHP is known today as *PHP: Hypertext Preprocessor*, which means that it handles data before it becomes HTML (*Hypertext Markup Language*).

According to Netcraft's April 2002 survey, PHP is now the most deployed server-side scripting language, running on approximately 9 million of the 37 million domains in their survey. This is confirmed by PHP's own figures, which show PHP usage (measured on a per-domain basis) growing at roughly 5% per month. In May 2003,

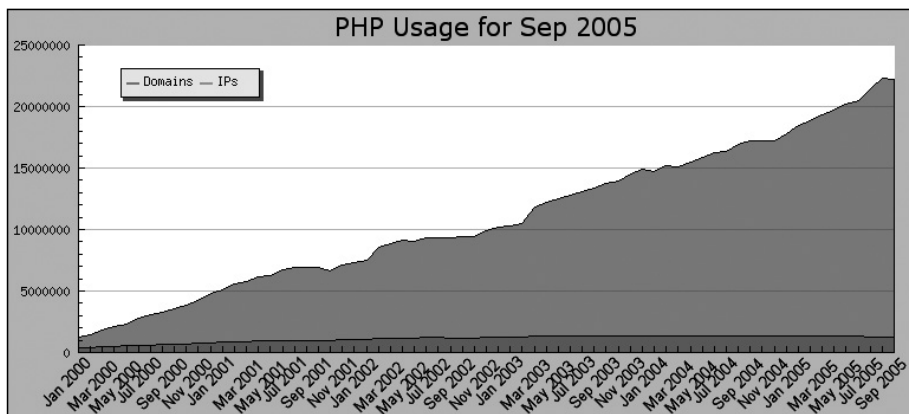


Figure 11 PHP Usage Statistics for September 2005

1 Perl (also known as the Practical Extraction and Report Language) is an interpreted programming language designed by Larry Wall.

2 Zend Technologies was found by Zeev Suraski and Andi Gutmans in Ramat Gan, Israel.

almost 13 million domains were using PHP, based on the same source.³ Today, the number of domains using PHP is even greater (see Figure 1.1).

PHP is also considered to be one of the few truly cross-platform programming languages, or to be more precise, one of the few truly cross-platform "scripting languages,"⁴ because it can run on almost any operating system and can be moved from each operating system with little or no changes to the coding.

1.2 Why PHP?

Today PHP is one of the most popular server-side scripting languages; however, is that enough to convince someone that this is the optimal platform for development? Some of the reasons that account for PHP's popularity include the following:

1. It's a loose language, a dynamically typed language, and therefore, the rules aren't as strict with variables—they don't have to be declared, and they can hold any type of object, thereby making PHP an ideal environment for rapid prototyping and development of not only small but large applications (see Table 1.1).
2. PHP applications can also be both—based as well as desktop style applications via the command line compilers.

Why do you use PHP?	Number of Responses	Response ratio
Easy to develop with	3093	89%
Web application focus	2328	67%
Affordable	2311	66%
Flexibility	2287	66%
Apache integration	2109	61%
Performance	1997	57%
Preference for non-Microsoft technology	1829	53%
Multiple platform support	1785	51%
Availability of the source code (open source)	1404	40%
UNIX/Linux investment	1304	37%

³ See <http://www.php.net/usage.php>.

⁴ The term "scripting language" usually refers to a language that is designed to do something after an event occurs.

Why do you use PHP?	Number of Responses	Response ratio
Multiple web server support	853	25%
Time to market	606	17%
Inherited an existing PHP system	281	8%
Other, please specify	300	9%

Table 1.1 Zend Technologies PHP Survey Results June 2003

In short, PHP is easier, faster, better, and more flexible to learn than most other languages, as well as being a very adaptable and "forgiving" language in terms of syntax and coding.

Often, it is because of these reasons that PHP is such an optimal tool. Within SAP, a company with so many possibilities for development, there is one area that affects every aspect of a project and that is time. You can use PHP to transform a concept or idea—rapidly develop, deploy, and present it—in a far more visual way than is possible with a presentation. Here, the idea is not to replace defined development methods, but to enhance the conceptual phase of the project, thereby allowing for more freedom in the initial stages. Once the project is well defined, you can begin working through the first stages of development. By using PHP, the value of the project in the initial stages can be quickly, cheaply, and easily determined, saving valuable time and resources. All of this depends on having the ability to work with PHP, which is something anyone who has the desire and the interest can do and learn well enough for these purposes.

1.3 How PHP Works

How PHP works is actually quite simple and easy to understand: The client machine will send the request to the server, usually in the form of a URL (*Uniform Resource Locator*) that will then be taken by the server and sent to the appropriate compiler, which will, in return, send the result back to the client machine. Figure 1.2 shows the process from the client request via a URL to the server that processes the PHP and returns HTML to the client.

```

<?php
$image = "http://static.php.net/www.php.net/images/php.gif";
?>

<html>
<head>
  <title>image.php example</title>
</head>

<body>
  <? echo "<img src=\"\$image\" border=\"0\"> "; ?>
</body>
</html>

```

Listing 1.1 Sample image.php Page

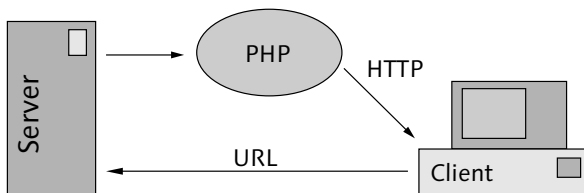


Figure 1.2 PHP Process

This is a standard server-side processing model. Next, the result that the client machine receives is interpreted by the browser and the HTML code is displayed appropriately in the browser window.

For example, let's take this example:

http://localhost:8080/image.php.

This is the URL that our browser will call. The server will locate this URL and verify that the extension has been defined as a PHP extension and therefore route the request to the PHP compiler.

The compiler will now interpret this request and load the *image.php* file (see Listing 1.1), read each line of the file, and determine what actions to take.

As you can see in Listing 1.1, there's a mix of HTML and PHP on the page. The PHP compiler compiles the items on this page that must be compiled, that is, the items between the "<?>" and "<?>" that indicate embedded PHP code. Of course, this is just a simplified example to show you how PHP works. What we have done here is taken a variable called *\$image* and assigned it the URL location of a graphic file that we want to display. This is considered a "simplified example," because you could

have just as easily provided the location instead of the *\$image* and therefore saved a processing step.

After the server has compiled the PHP code, the results are returned to the client in the form of static HTML (see Figure 1.3).



Figure 1.3 Result of Compiled PHP Code and HTML Code

If you now reviewed the source code of this page, you would no longer see any of our PHP coding, only the compiled and static HTML (see Figure 1.4).



Figure 1.4 Source Code of New Compiled Page

1.4 The Basics of Getting Started

From a developer's point of view, you'll need two basic items to get started—the desire to work with PHP and the means to do so. The latter can be something as basic as a simple text editor such as Notepad or WordPad that comes with the Windows operating system, or one of the various editors that come with the different flavors of Linux.

Or, you can try one of the more powerful development environments such as Eclipse with the various PHP plugins, or even a high-end package like the *Zend Studio* from Zend Technologies.

Whichever editor or environment you choose should be fine. Working with PHP is similar to working with HTML, or any other programming language for that matter, and the tool is most often not critical. What matters is that you employ whatever you need to get the job done.

For SAP developers, if you have already deployed and started using the SAP NetWeaver Developer Studio, I would highly recommend using "PHPEclipse,"⁵ which is a plugin for the Eclipse IDE. This plugin is easily integrated into Eclipse and quite handy when working with PHP.

⁵ See <https://sourceforge.net/projects/phpeclipse>.

```

.userBody {
  font-size: 95%;
  font-weight: bold;
  background-color: white;
  color: white;
  border-collapse: collapse;
  border: 1px solid #aaa;
  padding: 0 .8em .3em .5em;
}
#menu {
  font-size: 95%;
  font-weight: bold;
  line-height: 1.95em;
  background-color: white;
  color: black;
  border-collapse: collapse;
  border: 0px solid #aaa;
  padding: 0 .8em .3em .5em;
}
#clock {
  font-size: 95%;
  font-weight: bold;
  background-color: white;
  color: black;
  border-collapse: collapse;
  border: 0px solid #aaa;
  padding: 0 .8em .3em .5em;
}

```

Listing 5.5 Initial CSS Styles for Our Foundation

standard elements and custom elements and, in turn, defines each element's attributes from positioning to appearance. As a developer, this could prove very useful, especially if you need to add only the functionality but leave the design to someone else.

The rest of the information will be modified by simply adding the dynamic PHP. We'll also continue to add, alter, and adjust our HTML, JavaScript, and CSS.

5.3 PHP

We'll begin the implementation of PHP with the user list, which was the example that came with SAPRFC. We've

already seen that it works, so we simply need to take some of the coding and place it in our new file.

1. Copy the *saprfc.php* file from the *SAPRFC* directory to our *su01* directory.

2. Open *index.php* and find the "USER LIST" placeholder. Then, replace it with Listing 5.6. This will show additional cell definitions and CSS style assignments and a call to a PHP function `UserList($sap);`.

```

<td valign="top" width="120px">
  <table>
  <tr>
    <td class="tb-header">ID</td>
    <td class="tb-header">Status</td>
    <td class="tb-header">Valid</td>

```

```

</tr>
<?php echo UserList($sap); ?>
</table>
</td>

```

Listing 5.6 Replacement of the "USER LIST" placeholder

3. Create a new PHP file in your *su01* directory called *su01.php*, which we'll use to hold all of our functions for our page.
4. Open *su01.php* with your text editor and insert Listing 5.7.

You should pay particular attention to the following code segment in Listing 5.7. These are standard HTML methods, which enable us to use CSS styles and create the fol-

lowing effect: our rows will be highlighted as we move over them with our mouse.

```

onMouseOver="\this.className='highlight'↵
\" onMouseOut="\this.className='normal'\"

```

1. Before we can retrieve a list from the system, we must first log into the system. For the purposes of this example, I made a manual login. Later, we will make this login dynamic, but for now manual will suffice. Open your *index.php* file again and insert Listing 5.8 above the `<html>` tag. It includes the call to SAPRFC functions, SU01 functions as well as logging into the system.

```

<?php
    // SAPRFC class library and
    // custom library SU01
    require_once("saprfc.php");

```

```

<?php
function UserList($sap) {
    // Call function
    $result=$sap->callFunction("SO_USER_LIST_READ",
        array( array("IMPORT","USER_GENERIC_NAME","*"),
            array("TABLE","USER_DISPLAY_TAB",array())
        ));
    // Call successful?
    if ($sap->getStatus() == SAPRFC_OK) {
        // Yes, print out the user list
        foreach ($result["USER_DISPLAY_TAB"] as $user) {
            $user_status = GetStatus($sap,$user["SAPNAM"]);
            $user_valid = GetValid($sap,$user["SAPNAM"]);
            $listing .= "<tr onMouseOver=\"this.className='highlight'\"↵
                onMouseOut=\"this.className='normal'\">td class=\"tb-data\">↵
                <a href=\"index.php?user=\".$user[\"SAPNAM\"].\"\">\".$user↵
                [\"SAPNAM\"].\"</a></td><td class=\"tb-data\">\".$user_status.\"↵
                </td><td class=\"tb-data\">\".$user_valid.\"</td></tr>";
        }
    } else {
        // No, print long version of last error
        $sap->printStatus();
    }
    // Now return the list
    return $listing;
}
?>

```

Listing 5.7 Function to Gather User List and Return HTML Table Rows

```
include("su01.php");

// Now login into SAP system
$sap = login("cmehcr1","xxxxxxx");
?>
```

Listing 5.8 Header Information

2. Now that our page can make a call to the login function, we need to add it. Open your *su01.php* file again and insert Listing 5.9 and Listing 5.10 before the `UserList($sap);` function. Again, this would be a dynamic login for any system in the ideal scenario. Logoff functions are recommended with any dynamic login. For our example, closing the page will terminate the login.

```
function login($user,$pwd) {
    // Create SAPRFC instance
    $sap = new saprfc(array(
        "logindata"=>array(
            "ASHOST"=>"2.2.2.183"
            ,"SYSNR"=>"00"
            ,"CLIENT"=>"000"
            ,"USER"=>$user
            ,"PASSWD"=>$pwd
        )
        ,"show_errors"=>false
        ,"debug"=>false));
    return $sap;
}
```

Listing 5.9 Login Function

```
function logoff($sap) {
    // Logoff/Close SAPRFC connection
    // LL/2001-08
    $sap->logoff();
}
```

Listing 5.10 Logoff Function

3. Before we can continue, we need to implement a few other functions that our `UserList` function calls. In your *su01.php* file, insert Listing 5.11 after the `LogOff($sap)` function. Here again, the images are from the standard SAP icons located within the system.

```
function GetStatus($sap,$uid) {
    $value = "";
    $input = GetStatusValue($sap,$uid);
    if ($input == "UnLocked" ) {
        $value = "<img src=\"images/¬
            s_S_LOOP.gif\" border=\"0\" ¬
            alt=\"\".$input.\">";
    } else {
        $value = "<img src=\"images/¬
            s_S_LOCL.gif\" border=\"0\" ¬
            alt=\"\".$input.\">";
    }

    return $value;
}
```

Listing 5.11 PHP Function That Returns an HTML Image Tag

4. As you can see in the Listing 5.11, this PHP function calls another function. So, after this function, insert Listing 5.12. This PHP function makes a call to the system and retrieves a value, which, in this case, is retrieved from a custom ABAP function module.

```
function GetStatusValue($sap,$uid) {
    $value = "";
    $uid = strtoupper($uid);
    $result=$sap->callFunction¬
        ("Z_GET_LOCKSTATUS",
            array( array("IMPORT",¬
                "USERNAME",$uid),
                array("EXPORT",¬
                "STATUS",array())
            ));
    // Call successful?
    if ($sap->getStatus() == SAPRFC_OK) {
        // Yes, then get value
        if ($result["STATUS"] == "0" ) {
            $value = "UnLocked";
        } else {
            $value = "Locked";
        }
    } else {
        // No, print long version of
        // last error
    }
}
```

```

        $sap->printStatus();
    }

    return $value;
}

```

Listing 5.12 PHP Function That Makes a Call to the System

5. Now, we must create our custom ABAP function module within the system. If, per chance, you know of another way to access the required data, please feel free to experiment. One way to access the required data that comes to mind, but is not released by SAP for customer use, is the ABAP function module RFC_READ_TABLE.
6. Log into your system and go to Transaction SE37.
7. Type the name of our custom function module "Z_GET_LOCKSTATUS" and choose the **Save** button (see Figure 5.3).

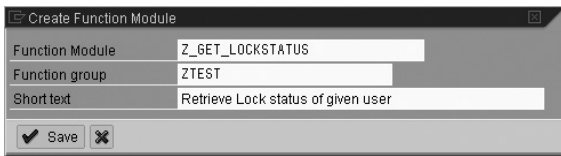


Figure 5.3 Creating Function Module in SE37

8. With the function module now created, we just need to set the **Remote-enabled module** attribute (see Figure 5.4) to ensure that we can use it from our PHP application.
9. Now we need to enter the **Import** parameters for the function module (see Figure 5.5). Our import parameter will be "USERNAME" and we will define as type "USR02-BNAME" which is the field "BNAME" from the SAP table USR02. BNAME is defined as **CHAR** with a length of 12. By checking the

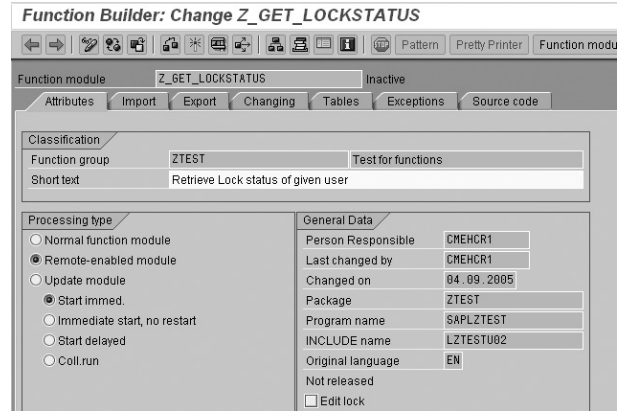


Figure 5.4 Attributes Tab

Pass Value, we're saying that the parameter contents are copied when the parameter is passed and when it is transferred back. Note that this is not recommended when the parameter is a table due to the toll it takes on performance.

10. Next, we'll set the **Export** parameters of the function (see Figure 5.6). Our export parameter will be "STATUS" and we will define as type "USR02-UFLAG," which is the field **UFLAG** from the SAP table USR02. **UFLAG** is defined as **INT1** with a length of 3.
11. Now switch over to the **Source Code** tab and enter Listing 5.13:

```

FUNCTION Z_GET_LOCKSTATUS.
*-----
*""Local interface:
*  IMPORTING
*    VALUE(USERNAME) TYPE  USR02-BNAME
*  EXPORTING
*    VALUE(STATUS) TYPE  USR02-UFLAG
*-----

```

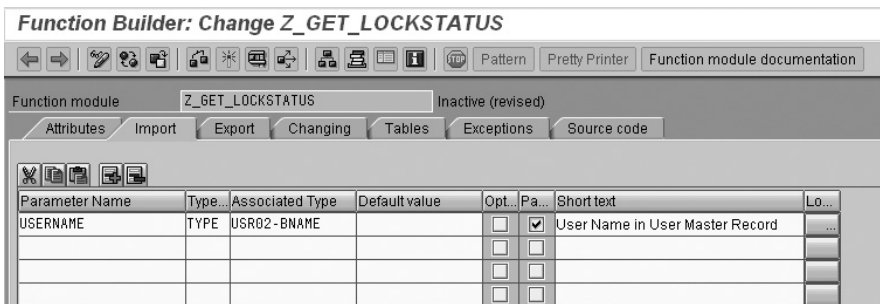


Figure 5.5 Import Tab of Our Function Module

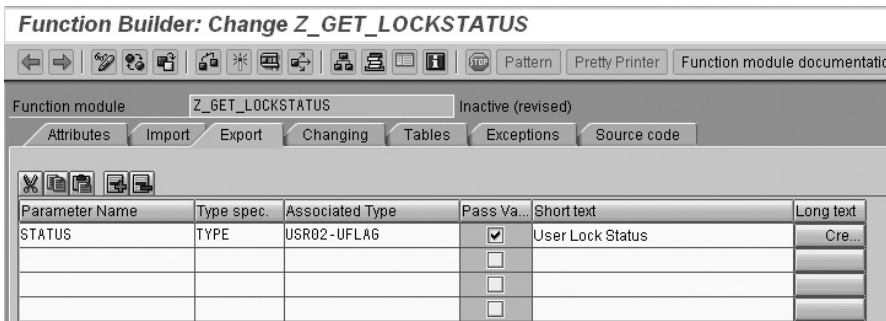


Figure 5.6 Export Tab of Our Function Module

```
SELECT uflag INTO status FROM usr02
WHERE bname = username.
ENDSELECT.
```

ENDFUNCTION.

Listing 5.13 ABAP Code for Our Custom Function Module

- Now you can save, check, and activate the function module. Feel free to try it out. It's very simple and direct, and should give you an idea of how easy it is to connect ABAP to PHP. You can also get an idea of how easy it would be to connect PHP to a rather complex ABAP function such as a SAP BAPI.
- Once we've completed those items, we can return to our PHP page. In *su01.php*, we have entered our first few functions to get the function `UserList` to

work. Next, we need to enter the remaining functions. Therefore, enter Listing 5.14 into the file.

- As you can see, this function calls `GetUserLogonDetails`. Therefore, we will now need to enter Listing 5.15 into our *su01.php* file. `GetUserLogonDetails` will return basic information about the user currently logged into the SAP system. For more information, refer to Transaction SE37 within your SAP system and the ABAP function module `BAPI_USER_GET_DETAIL`.
- Save the file in your editor. Then load your page in your browser as you have done previously and see if your list appears.

At this point, we should have a well-formatted list that appears when we open the page (see Figure 5.7).

```
function GetValid($sap,$uid) {
    $value = "";

    $result = GetUserLogonDetails($sap,$uid);
    $input = $result["GLTGB"];

    // Yes, then get value
    if ( strtotime("now") < strtotime($input) or $input == "00000000" ) {
        $value = "<img src=\"images/s_S_OKAY.gif\" border=\"0\" alt=\"\".$input.\">";
    } else {
        $value = "<img src=\"images/s_S_NONO.gif\" border=\"0\" alt=\"\".$input.\">";
    }

    return $value;
}
```

Listing 5.14 This Function Returns Whether or Not the User Is Still Valid in the System

```

function GetUserLogonDetails($sap,$uid) {
    $value = "";

    $uid = strtoupper($uid);
    $result=$sap->callFunction("BAPI_USER_GET_DETAIL",
        array( array("IMPORT","USERNAME",$uid),
            array("EXPORT","LOGONDATA",array())
        ));
    // Call successful?
    if ($sap->getStatus() == SAPRFC_OK) {
        // Yes, then get value
        $value = $result["LOGONDATA"];
    } else {
        // No, print long version of last error
        $sap->printStatus();
    }
    return $value;
}

```

Listing 5.15 Function to Retrieve User Logon Data

If not, you should review the code changes again to ensure that nothing was missed. If the list still doesn't show up, you will need to find the *dev_rfc.trc* file, which is most likely located in your *bin* directory of Apache HTTP Server. This file contains the developer trace of your RFC connection to SAP. If the connection to SAP is working, you'll need to log into the SAP system and check Transaction ST22 to determine whether your user has caused any runtime errors. If, at this point, you are still not further along, you'll need to contact your system administrator and ask him or her to run a trace (Transaction ST05) on your user for RFC. Together, with your system administrator, you should be able to determine the cause of the problem.







ID	Status	Valid
<u>BCUSER</u>		✓
<u>CMEHCR1</u>		✓
<u>DDIC</u>		✓
<u>TEST LOCK</u>		✓
<u>TEST VALID</u>		✗
<u>TMSADM</u>		✓

Figure 5.7 User List Outputted to the Screen with the Use of All Our PHP Coding

You'll probably notice that the list looks slightly different from yours. This is due to the CSS formatting. Therefore, we need to add the additional CSS styles now.

1. Open the *su01.css* file and add Listing 5.16 to complete our CSS enhancements for the list. These CSS changes will alter the appearance of the custom elements `tb-header` and `tb-data`, which have been applied to our HTML table in our page.

```

.tb-header {
    font-size: 95%;
    font-weight: bold;
    background-color: lightgrey;
    color: white;
    border-collapse: collapse;
    border: 1px solid #aaa;
    padding: 0 .8em .3em .8em;
}

.tb-data {
    align: center;
    border: 0px solid #aaa;
    padding: 0 .8em .3em .5em;
}

```

Listing 5.16 CSS Styles for the Table Header and Data Cells

Index

.NET 9

<div> tags 30

<textarea> 53, 65

A

ABAP 9, 35, 37

Action Log 42, 47, 53, 56, 58, 63, 65

Action Menu 43, 44

Action Parameter

 Capture 44

apache.exe 11, 14

Apache HTTP Server 9, 10, 12, 14, 30

 Commands 14

Attributes 21, 36

B

BAPI_USER_GET_DETAIL 27, 37

BAPI_USER_GETLIST 27

BAPI_USER_LOCK 27

BAPI_USER_UNLOCK 27

BEA Logic 9

Bells & Whistles 62

Beta Testing 47

Browser Usage 26

C

Calendar Scripts 60

Color scheme 66

Connection 29

Cookies 47

Corporate Identity 32

CSS 25, 29, 38

 New Enhancements 32

CSS Styles 38, 66

D

Data

 Modifying 57

Delete Action 56

Delete Function 56

Deleting 56

DHTML 25

DirectoryIndex 15

Display Resolutions 26

DocumentRoot 10, 15

Dynamic CSS 66

Dynamic Drive 30

Dynamic Login 47, 50

Dynamic PHP 33

Dynamic Styles 68

E

Eclipse 8, 9

Export parameters 36

F

File

 Write 43

Foundation 29

Function

 Calling 21

Function Modules 37

G

GetUserAddressDetails 39

GetUserLoginDetails 37, 39

H

Header Information 35

Hidden Form Field 69

Highlighting 39

HTML 5, 29

HTML checkbox 69

HTML table 30

httpd.conf 10, 11, 13, 15

HTTP Server 9

 Manual Configuration 13

I

IBM WebSphere 9

IF 53

Images 32

Import Parameters 22, 36

Internet search 48

J

Java 9

JavaScript 25, 26, 29, 32, 59, 66

 Usage 26

Java Connector 9, 19

L

Layout 27, 30

Listen 10

Login Client 21

Login Function 35

Login Page 52

Logoff Function 35

Log Size 66

M

Menu Area 53

Menu Options 55, 63

O

onMouseOver 34

P

Passwords 21
 PHP 5, 8, 11, 23, 29, 33, 37, 50
 Advantages 6
 Backup 12
 Compiler 7
 Environment 10
 Extensions 13
 File 57
 Hello World 15
 History 5
 Installing 11
 Integrating into Apache HTTP Server 13
 Processes 6
 Setup Program 12
 php.ini 14, 16
 PHPeclipse 8
 Printing 56

R

Radio Buttons 66
 Rasmus Lerdorf 5
 Remote-enabled module 21, 36
 RFC 44
 RFC_READ_TABLE 36
 RFCSDK 19

S

SAPRFC 9, 11, 16, 19, 20, 33, 57
 Class Library 20
 Functions 34

 Instance 21
 Log Off 23
 SAP Design Guild 66
 SAP GUI 27
 SAP NetWeaver Developer Studio 8
 SAP system 20, 25, 27
 Connect 19
 ServerRoot 10
 Sessions 47, 48
 Dormant 49
 Session Management 49
 Initialization 52
 ShowActionLog() 53
 SMTP 12
 Server Settings 12
 SO_USER_LIST_READ 21
 System
 Logoff 45

T

Table Parameters 22
 Transaction SE37 21, 27, 36, 37
 Transaction SU01 25, 27

U

ULUser function 44, 54
 URL 6
 User
 Lock 25
 Unlock 25
 USER_GENERIC_NAME 21
 UserList function 22, 25, 33

User Action 58
 User Details 41
 User Information 25
 User Name 21
 User Validity 25
 USR02-BNAME 36
 USR02-UFLAG 36

W

W3C 11, 26
 W3Schools 26
 Website
 PHP-based 25
 Web Application Server 20, 25
 Web browser
 Instances 49, 50
 WriteActionLog(\$value) 44

Z

Z_GET_LOCKSTATUS 36
 Zend Studio 8
 Zend Technologies 5