# Integrating SAP with PHP

by Jason Simmons

*SAP is an extremely large ERP system which contains a wealth of information relating to the business where it is installed. SAP is marketing a number of new web-based platforms to exploit this data in web applications. These can be expensive and too "large" for your requirements. This article highlights how PHP and the SAPRFC extension could provide a cost effective alternative to these offerings.*

SAP is one of the most successful Enterprise Resource Planning tools currently available. It enables the management of all areas of your business, such as logistics, financial information, human resources, manufacturing processes and inventory management. It has many different modules, which you select according to your industry and processes. Due to its widespread coverage, SAP can contain some very important information regarding the performance and operation of your business.

With the emergence of Internet based technologies such as the Web, Java, XML and browser-based applications in general, SAP has released a number of products under the new mySAP title to help companies reach the data stored deep within the recesses of SAP. These products can be expensive and, in some cases, too large for a company's actual requirements. A lot of these products also rely on the deployment of Java. What if you are a PHP developer and do not have the time to learn Java—or you just don't want to use Java for various reasons?

At the end of this article, you will have enough information to utilize SAP data inside PHP. Depending on your requirements, you may even be able to forgo the need to purchase some of the more expensive SAP tools to extract data.

Enough introductory mumbo-jumbo… let's get on and grab hold of that valuable SAP information!

## The PHP SAPRFC Extension

Eduard Koucky has developed a PHP extension to incorporate SAP RFCs within PHP. SAP RFC, as you can imagine, stands for SAP *Remote Function Calls*. Once this extension is installed, you will able to execute SAP functions using PHP Commands.

The extension is available from SourceForge at *http://saprfc.sourceforge.net*. The latest version is 1.3.2, which contains some new additions to the API, and support for PHP5. I have tested version 1.3 using PHP 4.3.8, and I recommend that you try the latest version. The installation instructions are quite clear and straightforward—and, most of all, easy to follow. For Windows machines, you will need to install the win32 binaries and configure your PHP.ini file to load the extension. You will also need the SAP-supplied DLL, `librfc.dll`. I have found that the easiest way to ensure

**REQUIREMENTS**

PHP: **4.3.8**
OS: **Linux or Windows**
Other: **Non-Unicode SAP RFCSDK 6.20, SAPRFC PHP Extension, SOAP**
Code Directory: **SAP**

that the correct DLLs are all loaded is to install the SAP GUI client.

If you are installing the extension under Linux, then things get a little trickier. One of the most important components that you need to obtain is the SAPSDK. The author indicates that you can download the SDK from *http://service.sap.com/swdc* under `patches`. You will need to be a SAP customer and be able to login. It can be difficult to find anything on the SAP website, so I looked elsewhere. After much hunting, I found the SAPSDK on the "Presentation CD2" which comes with the SAP software media pack. You will need to do some research here, as I'm not sure if the location of the SAPSDK is consistent across different versions or upgrades. For my part, I'm using version 4.6D of SAP.

Everything I'm outlining here relating to the SAPSDK is just guidelines. It's best not to try and follow these instructions verbatim. When mounting the CD, it is important to turn off the auto mapping of uppercase to lowercase characters, for example:

```
mount -o map=off /dev/cdrom /mnt/cdrom
```

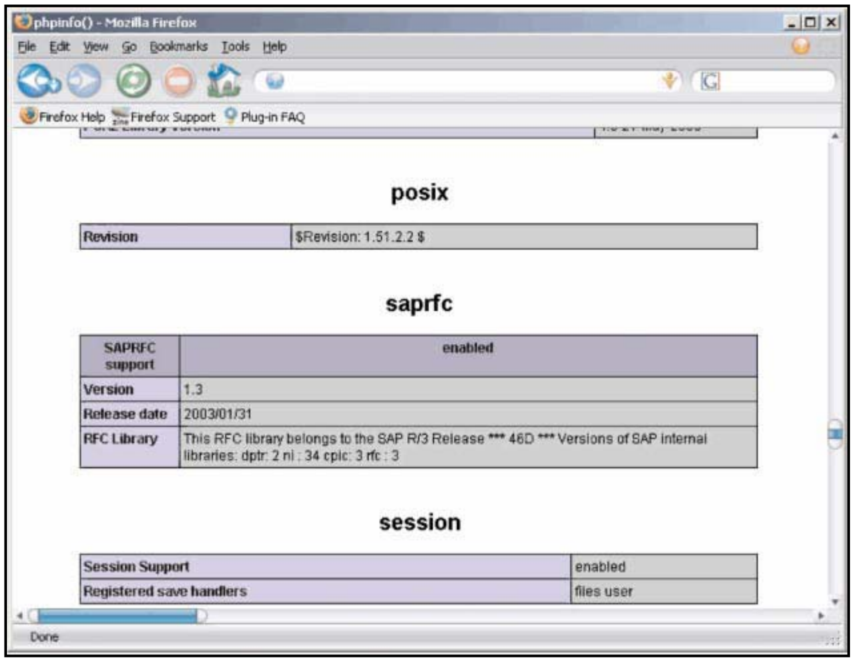
Locate the `Linux` or `Unix` subdirectory under the `SDK` directory. In there, you should find 2 files called `RFC.SAR` and `SAPCAR`. The former is an archive that contains the actual SDK. `SAPCAR`, on the other hand, is an executable that you have to use to extract the required files—it seems to be SAP's own version of the UNIX `tar` utility, and the syntax required to run both of them is very similar. To extract the files, enter the command

```
./SAPCAR -xvf RFC.CAR
```

This will extract the files required under a directory called `rfcsdk`. Make note of the location of these files, as you will need this later when installing the PHP extension. The PHP Extension installation guide suggests placing the files in the `/usr/sap/rfcsdk` directory, or alternatively, in `/usr/local/rfcsdk` or `/opt/rfcsdk`, depending on your installation.

From this point forwards, you can just follow the instructions given in the installation guide for the extension and you should be able to get everything running without much in the way of problems. I thought it'd be a good idea to spend some time explaining how to get hold of the SDK, due to the fact that the installation guide does not provide much information on how to locate it. The guide will, however, explain how to compile and install the PHP extension, so please do follow the instructions closely, as small oversights can cause the compilation to fail. Throughout the remainder of this article, I will

**Figure 1**


phpinfo() - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Firefox Help
Firefox Support
Plug-in FAQ
Done

posix

| | |
|---|---|
| Revision | $Revision: 1.51.2.2 $ |

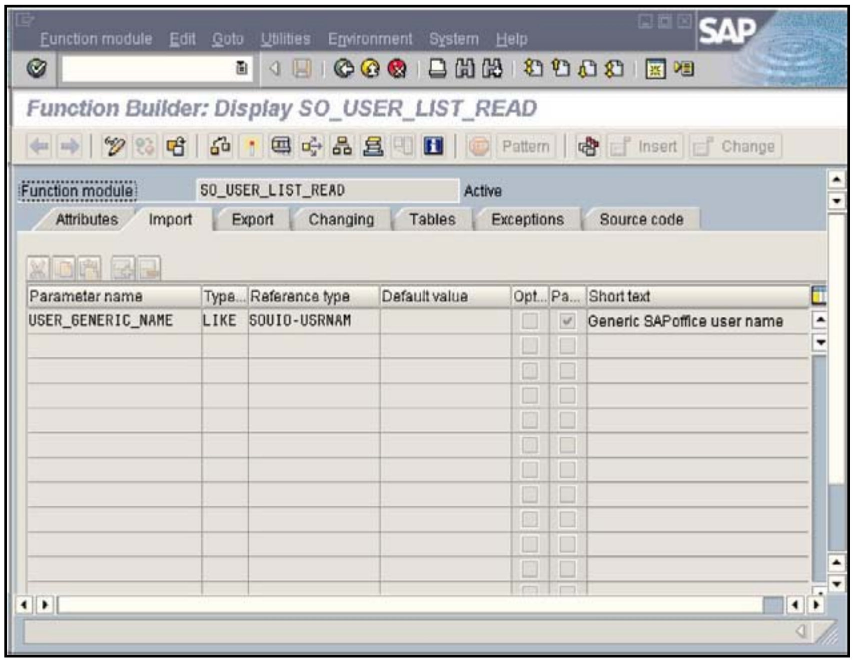
saprfc

| | |
|---|---|
| SAPRFC support | enabled |
| Version | 1.3 |
| Release date | 2003/01/31 |
| RFC Library | This RFC library belongs to the SAP R/3 Release *** 46D *** Versions of SAP internal libraries: dptr: 2 ni : 34 cpic: 3 rfc : 3 |

session

| | |
|---|---|
| Session Support | enabled |
| Registered save handlers | files user |

**Figure 2**



Function Builder: Display SO_USER_LIST_READ

Function module SO_USER_LIST_READ Active

| Parameter name | Type... | Reference type | Default value | Opt... | Pa... | Short text |
|---|---|---|---|---|---|---|
| USER_GENERIC_NAME | LIKE | SOUIO-USRNAM | | ☐ | ✓ | Generic SAPoffice user name |

assume that you have successfully compiled the extension (which should up in the output from a call to phpinfo(), as shown in Figure 1)—otherwise, you won't be able to run any of the example scripts.

## SAP Functions

SAP Functions are written in a programming language called ABAP. To utilize these functions, you will need SAP expertise, or access to a SAP ABAP developer. SAP functions have **IMPORT** parameters and also **EXPORT** parameters. **IMPORT** parameters need to be supplied when invoking an RFC, and the resultant data will be provided via the **EXPORT** parameters or a specified results table. Remember that these functions will be running on a remote server. Therefore, the context of **IMPORT** and **EXPORT** is from the remote SAP server's point of view. The SAP server expects you to supply **IMPORT** parameters, and it will supply you with **EXPORT** parameters. You will also need a username and password to invoke these SAP functions. I would suggest setting up a "non-dialog" user for use with PHP.

As a test, we are going to call a standard RFC that comes with SAP to provide a list configured SAP users. The standard SAP function to do this is **SO_USER_LIST_READ**. This function has one **IMPORT** parameter called **USER_GENERIC_NAME**, as you can see in Figure 2. You can supply this parameter if you wish to select a particular user.

This function places the resultant data into a table called **USER_DISPLAY_TAB** (shown in Figure 3). It is also important to ensure that the function is configured to allow remote invocation, as shown in Figure 4.

### Listing 1

```
 1 <?php
 2 $LOGIN = array (
 3 "ASHOST"=>"PRODUCTION01",  // application server host name
 4 "SYSNR"=>"00",             // system number
 5 "CLIENT"=>"400",           // client
 6 "USER"=>"RFCUSERNAME",     // user
 7 "PASSWD"=>"PLSCHANGEME",   // password
 8 "CODEPAGE"=>"1100");       // codepage
 9
10 //----- Set the name of the function
11 $rfcfunction  = "SO_USER_LIST_READ";
12 $resultstable = "USER_DISPLAY_TAB";
13
14 //----  Make a connection to the SAP server
15 $rfc = saprfc_open($LOGIN);
16
17 if(!$rfc) {
18     // We have failed to connect to the SAP server
19     echo "Failed to connect to the SAP server".saprfc_error();
20     exit(1);
21 }
22
23 //---- Locate the function and discover the interface
24 $rfchandle = saprfc_function_discover($rfc, $rfcfunction);
25
26 if(!$rfchandle){
27     // We have failed to discover the function
28     echo "We have failed to discover the
function".saprfc_error($rfc);
29     exit(1);
30 }
31
32 //---- Setup the results handle
33 saprfc_table_init($rfchandle,$resultstable);
34
35 //--- Call the function and check for errors
36 $rfcresults = saprfc_call_and_receive($rfchandle);
37
38 if ($rfcresults != SAPRC_OK){
39
40     if ($rfcresults == SAP_EXCEPTION){
41         $error = ("Exception raised:".saprfc_exception($rfchan-
dle));
42     }else{
43         $error = ("Call error:".saprfc_error($rfchandle));
44     }
45     echo $error;
46     exit();
47 }
48
49 $results =array();
50 $numrows = saprfc_table_rows($rfchandle,$resultstable);
51
52 for ($i=1; $i <= $numrows; $i++){
53
54     $results[$i] = saprfc_table_read($rfchandle,$result-
stable,$i);
55 }
56
57 saprfc_function_free($rfchandle);
58 saprfc_close($rfc);
59 var_dump($results);
60 ?>
```

## Calling SAP Functions

With that behind us, let's look at some PHP code that can be used to invoke this function. As you can see in Listing 1, the **$LOGIN** array defines the connection details to the SAP server where we wish to run **SO_USER_LIST_READ**. With this taken care of, we open a connection to the SAP server using saprfc_open() on line 15. Line 24 creates our object within the SAP server using saprfc_function_discover(). We then prepare our results table within this object on line 33 with saprfc_table_init() On line 36, the function is then executed and the result is captured. As long as the result is **SAP_OK**, we can move on to extracting our data from the SAP object. Lines 52 to 55 loop through the results, extracting them from the table using saprfc_table_read() and placing them in an array. Finally, the code on lines 57 and 58 releases the object on the SAP server and closes the connection. It is always good manners to tidy up and close the door behind you. If all went well, you should see a var_dump() of your results array—this should contain a list of users configured on your system.

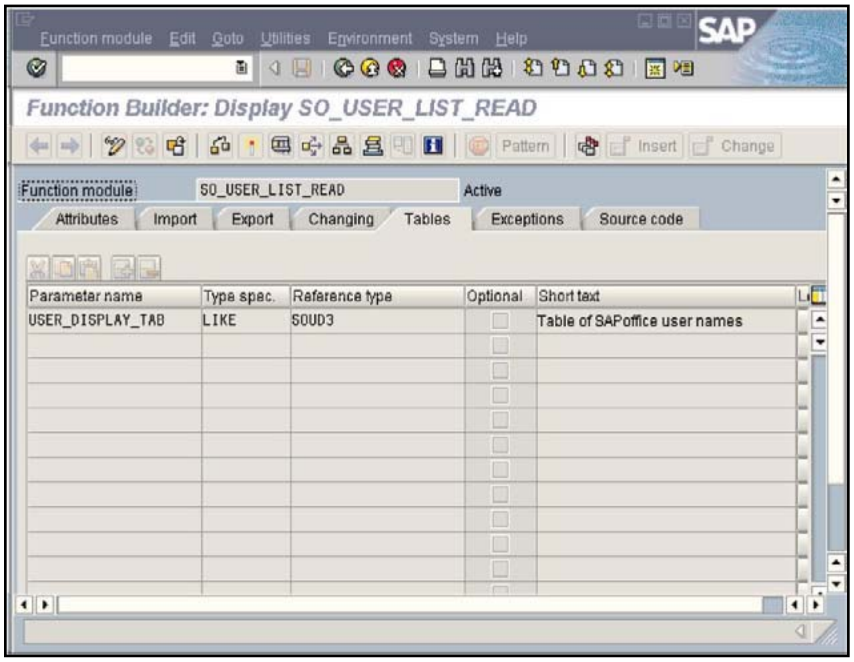
As you can see, it is fairly straightforward to invoke SAP functions from within PHP. If you look at Listing 1, you can see that, after every SAP operation, we do some error checking to make sure we have received valid results back from the server. Also, every time we call a SAP function we go through the same process of logging in, setting up the function object and executing the call. The author has supplied a PHP class to make these activities slightly easier to code, but even that could still do with some improvements—such as, for example, a nice PHP5 class implementation with some solid exception handling.

One very important point to mention at this stage is security. If you look at our example user-listing script, we have minimal security in the form of our PHP script providing logon information to the SAP servers.

However, everyone who can access the PHP script now has access to the SAP function SO_USER_LIST_READ. This punches a hole through SAP's security measures. Therefore, we must provide a method to ensure that only the right people have access to this script. We could ask the user to provide SAP logon information, which the PHP script then uses to invoke the RFC. If the user does not have the correct privileges, the script will fail. Another option is to provide some authentication and access control within the PHP script itself (perhaps using PHPGACL, which can be found at *http://phpgacl.sourceforge.net/*). It is also important to safeguard the actual script if it contains a hard-coded username and password set for your SAP systems. The implications of a security breach in your SAP systems are quite serious; therefore, security should be at the forefront of your application design.

As mentioned earlier, the SAP function SO_USER_LIST_READ is built-in. SAP comes with hundreds of predefined functions, often referred to as BAPIs (Business Application Programming Interfaces). Using our newly installed PHP extension, we now have the ability to invoke any of the RFC-enabled BAPIs to meet our requirements.

**Figure 3**



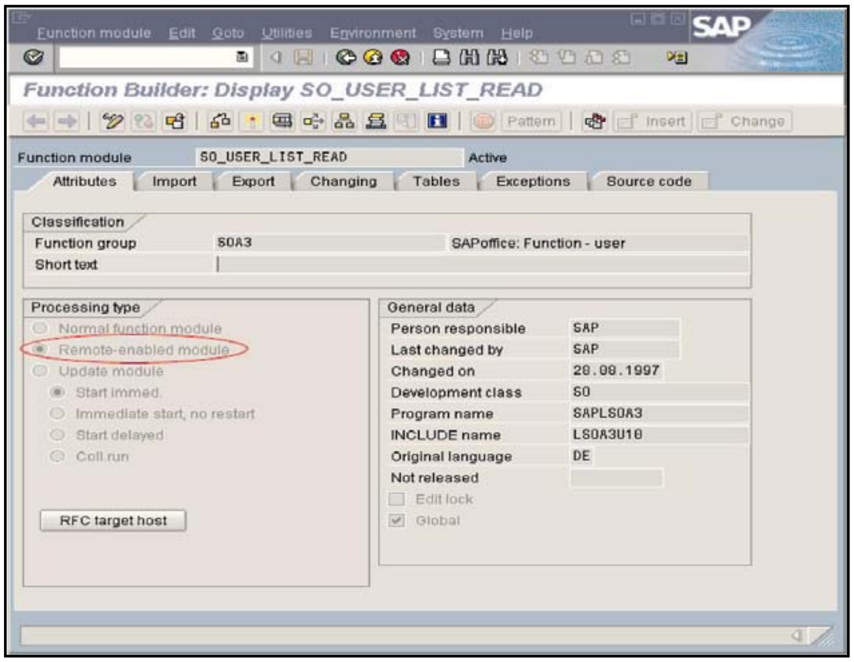
Function Builder: Display SO_USER_LIST_READ

Function module SO_USER_LIST_READ Active

Attributes | Import | Export | Changing | Tables | Exceptions | Source code

| Parameter name | Type spec. | Reference type | Optional | Short text |
|---|---|---|---|---|
| USER_DISPLAY_TAB | LIKE | SOUD3 | ☐ | Table of SAPoffice user names |

**Figure 4**



Function Builder: Display SO_USER_LIST_READ

Function module SO_USER_LIST_READ Active

Attributes | Import | Export | Changing | Tables | Exceptions | Source code

Classification

| | | |
|---|---|---|
| Function group | SOA3 | SAPoffice: Function - user |
| Short text | | |

Processing type

- Normal function module
- [✓ Remote-enabled module]
- Update module
  - [✓ Start immed.]
  - Immediate start, no restart
  - Start delayed
  - Coll.run

General data

| | |
|---|---|
| Person responsible | SAP |
| Last changed by | SAP |
| Changed on | 20.08.1997 |
| Development class | SO |
| Program name | SAPLSOA3 |
| INCLUDE name | LSOA3U10 |
| Original language | DE |
| Not released | |
| Edit lock | ☐ |
| Global | ✓ |

You can view the BAPIs defined in your installation of SAP by using the SAP Function Builder SE37 (See Figure 5) or the Object Navigator SE80. You will see that BAPIs exist for all areas of business. If you look at the screenshot in Figure 5, you can see some of the business areas covered.

For example, suppose that we are to produce an electronic purchase requisition system with authorization workflow in PHP. To raise the order in SAP, PHP could call the appropriate BAPI in the MM-PUR function modules. In this case, we would most likely use the BAPI BAPI_PO_CREATE to place the order within SAP.

Still, as extensive as SAP is, there are times where you will need to create your own functions. If you have SAP systems installed, you are most likely to have some expertise that will create particular functions for you at hand—be it yourself or some other resource within your organization. However, if you're not versed in ABAP programming, it would probably be of benefit if you were to get at least a basic understanding of its capabilities and mechanics. As a first step, SAP comes with a couple of development tools, SE38 ABAP Editor and SE80 Object Navigator, that can help you build your very own functions.

## PHP Power and SAP

The scale of what we have just enabled becomes apparent when we start to combine other technologies into the mix.

As we all know. PHP can communicate with many different technologies, such as Macromedia Flash, PDF generators, Oracle databases and GSM phones—just to name a few. What if we took some of these technologies and combined them with our new SAP Extension? We could create some extremely powerful business applications.

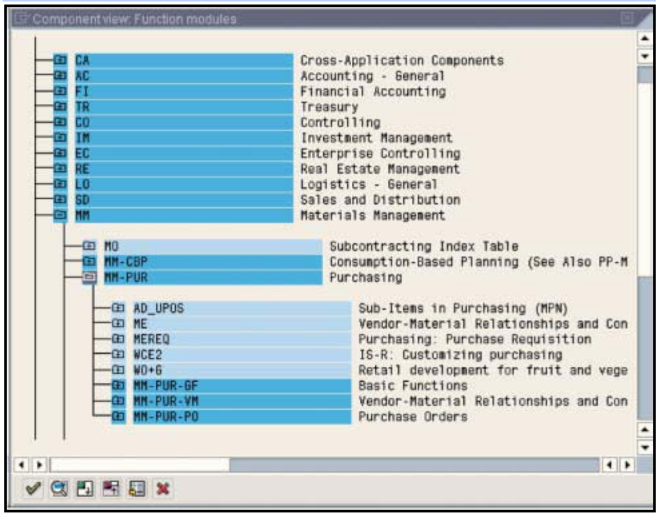
For example, I have already touched on the idea of a Web-based electronic purchasing requisition system. What about an alerting system that sends notifications via SMS/EMAIL (Material scrap rate has reached 90%, Purchase order XYZ is 5 days late) or a reporting tool that generates dynamic PDF reports based on SAP data?

To illustrate my point, I would like you to dig out your July 2004 copy of *php|architect*. In this issue, Jason Sweat takes us through data mining with the JpGraph graphing class (the data being mined is the PHP bug database). His article shows some impressive graphs based on data retrieved from the database. If we were to replace the data retrieval functions with SAP RFC functions, we would have created a powerful SAP data-mining tool with little difficulty.

## A SAP Information Portal

Let's now examine a real-world example. Imagine that

### Figure 5



```
CA          Cross-Application Components
AC          Accounting - General
FI          Financial Accounting
TR          Treasury
CO          Controlling
IM          Investment Management
EC          Enterprise Controlling
RE          Real Estate Management
LO          Logistics - General
SD          Sales and Distribution
MM          Materials Management
  MO          Subcontracting Index Table
  MM-CBP      Consumption-Based Planning (See Also PP-M
  MM-PUR      Purchasing
    AD_UPOS     Sub-Items in Purchasing (MPN)
    ME          Vendor-Material Relationships and Con
    MEREQ       Purchasing: Purchase Requisition
    WCE2        IS-R: Customizing purchasing
    WO+G        Retail development for fruit and vege
    MM-PUR-GF   Basic Functions
    MM-PUR-VM   Vendor-Material Relationships and Con
    MM-PUR-PO   Purchase Orders
```

### Figure 6

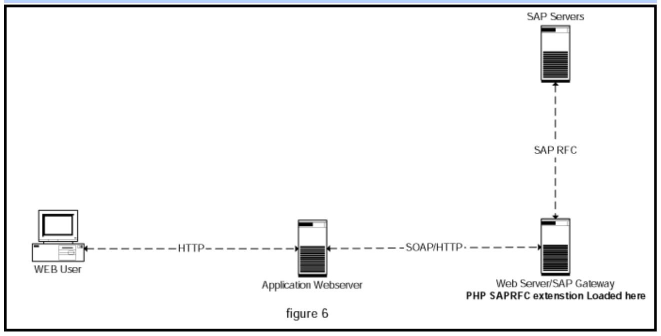


figure 6

### Listing 2

```php
<?php
require_once('nusoap.php');
// ---stcokservice.php ----------------------------------
// getwbstock()
//
// Stock allocation by area as a percentage
// fills an array with the results
//      EXPORTING
//            VALUE(SYSTEM) LIKE  SY-SYSID
//            VALUE(TRDIR) LIKE  TRDIR STRUCTURE  TRDIR (ARRAY)
//      TABLES
//              JTAB ( ARRAY)
//  SAP FUNCTION /USR/COSMIO0228A
// ------------------------------------------

function getwbstock(){
    global $LOGIN;
    // Identify custom ABAP RFC function that we need to run on the SAP server
    $function_name = "/USR/COSMIO0228A";

    $rfc = saprfc_open ($LOGIN); // login to the SAP Server then check for errors
    if (!$rfc )
    {
        // Need some better error handing in here
        $error = "RFC connection failed with error:".saprfc_error();
        return new soap_fault("Client","",$error);
    }

    // Make sure that the SAP function we wish to run can be located.
    $fce = saprfc_function_discover($rfc, $function_name);
    if (! $fce )
    {
        $error = "Discovering interface of function module $function_name failed";
        return new soap_fault("Client","",$error);
    }


    // Setup the results table
    saprfc_table_init ($fce,"JTAB");

    // Call the function and check for errors
    $rfc_rc = saprfc_call_and_receive ($fce);
    if ($rfc_rc != SAPRFC_OK)
    {
        if ($rfc == SAPRFC_EXCEPTION )
        $error = ("Exception raised: ".saprfc_exception($fce));
        else
        $error = ("Call error: ".saprfc_error($fce));
        return new soap_fault("Client","",$error);
    }

    // Read the results into variables
    $SYSTEM = saprfc_export ($fce,"SYSTEM");
    $TRDIR = saprfc_export ($fce,"TRDIR");
    ;

    // Init our variables to hold the totals in each stock location
    $indexcount  =0;
    $totalstock  =0;
    $totalpurch  =0;
    $totalassemb =0;
    $totalfinish =0;
    $totalspare  =0;
    $totalredund =0;
    $totalobsole =0;
    $JTAB = array();

    // Loop through the number of results returned by SAP
    // Then store the totals for each location so we can cal-culate our percentages
    $rows = saprfc_table_rows ($fce,"JTAB");
    for ($i=1; $i<= $rows ; $i++)
    {
        $temp = saprfc_table_read ($fce,"JTAB",$i);
        $flag=false;
        foreach ($temp as $key => $value){
            if((int)$value ) $flag=true;
        }
        if($flag) {

            $totalstock   += $temp["TOTAL"];
```

*Continued on page 47*

you have been tasked with creating a management information portal for the business managers in your company. The key requirement is that it must to be as “live” as possible, and they should have the choice as to what information they can see once they have logged in. Most of the information required is within the company’s SAP system, and the users want to process the data according to their own requirements without having it “disconnected” from it.

The best way to fulfill these requirements is to break down the information required into objects that can then be reused around different areas of the portal and other applications. For example, we could have an object that outputs data to a PDF file, Excel spreadsheet, SMS/Email alert system and even graphs—managers and marketing types, after all, just love graphs.

In this section, we are going to create a graphing object that will show the allocation of stock by location as a percentage. The key components to our object will be the SAP RFC Extension, NuSOAP class, and JpGraph graphing tool.

Figure 6 provides an overview of how our object will operate. At the core of the system is our web server, configured with the PHP SAP RFC extension. It also hosts the SOAP service, which will transport the data retrieved from the SAP servers. The second web server hosts the Information portal application. This is where our business managers will come to request data. This web server will be using SOAP clients to request SAP based data. It is not a requirement that two web servers be used in this manner—you can place the Application Server and the SAP Gateway on the same machine. However, you may find this to be a less scalable solution.

For our object to work, we need to locate the relevant data held within SAP. For this example, I used an RFC that was written specifically to provide this information for my PHP script. I consulted with the ABAP developer as to what **IMPORT** and **EXPORT** parameters are required. In Listing 2, the function **getwbstock()** invokes the SAP based function **/USR/COSMIO0228A**. This function will retrieve our stock information and place the results in a table called **JTAB**. The **JTAB** table on the SAP server is

### Listing 2: *Continued from page 46*

```
            $totalpurch   += $temp[“PURCH”];
            $totalfinish  += $temp[“FINISH”];
            $totalassemb  += $temp[“ASSEMB”];
            $totalspare   += $temp[“SPARES”];
            $totalredund  += $temp[“REDUND”];
            $totalobsole  += $temp[“OBSOLE”];
            $indexcount++;
        }
    }


    $JTAB[“PURCH”]  = (int) ($totalpurch / ($totalstock/100));
    $JTAB[“FINISH”] = (int) ($totalfinish / ($totalstock/100));
    $JTAB[“ASSEMB”] = (int) ($totalassemb / ($totalstock/100));
    $JTAB[“SPARES”] = (int) ($totalspare / ($totalstock/100));
    $JTAB[“REDUND”] = (int) ($totalredund / ($totalstock/100));
    $JTAB[“OBSOLE”] = (int) ($totalobsole / ($totalstock/100));

    // Clean up and return our results back to the client
    saprfc_function_free($fce);
    saprfc_close($rfc);
    return $JTAB;

}// -------------- End of Function getwbstock





$server = new soap_server;
$LOGIN = array (
“ASHOST”=>”production01”,            // application server host name
“SYSNR”=>”00”,                       // system number
“CLIENT”=>”400”,                     // client
“USER”=>”rfcfunc”,                  // user
“PASSWD”=>”MYPASSWORD”,              // password
“CODEPAGE”=>”1100”);                 // codepage

$server->register(“getwbstock”);  // Register the Webservice wbstock
$server->service($HTTP_RAW_POST_DATA);
?>
```

### Listing 3

```
<?PHP
// ---SAPGraphs.class.php ---------------------------------
//  Graph stock data obtained from SOAP service getwbstock()
// ------------------------------------------
Class SAPStockGraphClass {


    var $stockdata;

    function SAPStockGraphClass(){

        include (“/var/www/jpgraph-1.15/src/jpgraph.php”);
        include (“/var/www/jpgraph-1.15/src/jpgraph_pie.php”);
        include (“/var/www/jpgraph-1.15/src/jpgraph_pie3d.php”);
        $this->stockdata = array();

    }


    function GetStockData(){

        require_once (“/usr/local/lib/php/nusoap.php”);
        // Create new saop object to the stock service on the appplication server
        $soapserver = new soapclient(“http://webapps/sapdemo/stockservice.php”);
        $this->stockdata    = $soapserver->call(“getwbstock”);
    }

    function PlotGraph(){

        $data = array_values($this->stockdata);
        $labels = array_keys($this->stockdata);

        $graph = new PieGraph(400,200,”auto”);
        $graph->SetShadow();

        $graph->title->Set(“Current Stock allocation”);
        $graph->title->SetFont(FF_FONT1,FS_BOLD);

        $p1 = new PiePlot3D($data);

        $p1->SetSize(0.5);
        $p1->SetCenter(0.45);
        $p1->SetLegends($labels);
        $graph->legend->SetAbsPos(300,10,’left’,’top’);
        $graph->Add($p1);
        $graph->Stroke();


    }

}
?>
```

then read row by row. Lines 53 and 54 provide an example of reading information from the `EXPORT` fields by reading system status information. `EXPORT` parameters are read by using `saprfc_export()`.

Calculations are performed on the data and the results are placed in an array named `$JTAB`. The `$JTAB` array is then returned, ready to be plotted by JpGraph. Incidentally, we do not have to call the array `$JTAB` here. I named it `$JTAB` just for continuity and ease of reference.

"... we are going to create a graphing object that will show the allocation of stock by location as a percentage."

This function is then exposed as a SOAP service. In lines 110-120, the login information is defined in the `$LOGIN` array and passed to the SAP server. The SAP RFC will run using the permissions of the user defined here. By way of a reminder, I would recommend including a privilege mechanism within this service to ensure that the correct people have access to the data that this function provides. As this is the SOAP server, Listing 2 will reside on the central Web server /SAP Gateway with the SAP RFC Extension Loaded as previously illustrated in Figure 6.

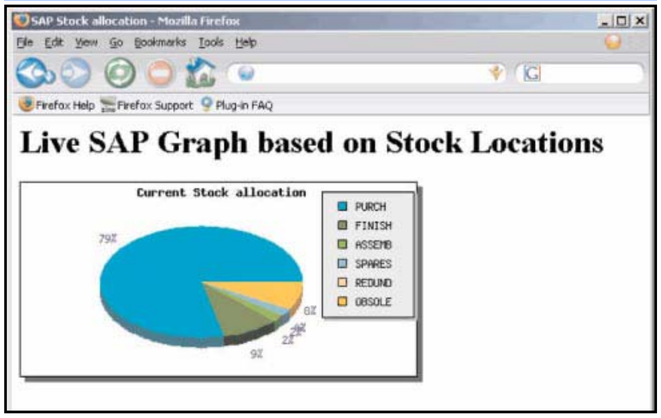
The next element of our object is the client SOAP service that will invoke our SAP RFC. Listing 3 shows our client based class. This is where our SAP RFC Data is retrieved and consumed via a JpGraph Object. The method `GetStockData()`, starting at line 20, invokes the SOAP service `getwbstock()` residing on the SAP gateway server. The data is then consumed by the method `plotGraph()` starting at line 28. This method utilizes a JpGraph object to plot the data in a pie graph. For want of some error checking and validation, our object is complete. To create our stock graph, we need to create a script to drive our new client based class. If we refer back to Figure 6, our new SOAP client will reside on the Application web server, as well as the following code snippet, which instantiates our SOAP client.

```
<?php

// Stockgraph.php
// Create a PNG stream graph image based on SAP
Stock data
// Now include our SOAP client (listing3.php)
 require_once("SAPGraphs.class.php");
 $stockobject = new SAPStockGraphClass();
 $stockobject->GetStockData();
 $stockobject->PlotGraph();

?>
```

Listing 3 shows how this object can be invoked with just a few lines of code. This graph can now be treated directly as an image in HTML documents. If you were using `IMPORT` parameters, you could even use CGI variables to communicate with our graphing object to narrow down your selection. For example, take a look at the HTML code in Listing 4, where our script `Stockgraph.php` is being used to create a PNG stream to produce our graph, which is shown in Figure 7.

As with all JpGraph images, Care must be taken to capture all error messages and direct them to a log file or generate a graphical error message, rather than allow PHP to output text directly to the browser. This will ensure that your image stream, which JpGraph will generate, will not be corrupted with string data in the

**Figure 7**


SAP Stock allocation - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Firefox Help Firefox Support Plug-in FAQ
Current Stock allocation
79%
8%
2%
9%
PURCH
FINISH
ASSEMB
SPARES
REDUND
OBSOLE

Live SAP Graph based on Stock Locations

## Useful Links

http://saprfc.sourceforge.net/
http://www.sap.com/solutions/netweaver/index.aspx
http://www.sapgenie.com/its/index.htm
http://www.sap.com/solutions/netweaver/webappserver/index.aspx

form of error messages.

Our code can now be utilized in any application that needs to display this data. One point to remember is that, each time this graph is displayed, it will run the RFC on the SAP server. This could produce undue load on your SAP system, so it might be best to develop some sort of caching mechanism to minimize the load on the SAP server. However, it might be a good idea to produce a timestamp on the graph to indicate how old the data is.

## Conclusion

I hope you are as excited by this PHP extension as I was when I first came across it. It's exciting because it places a whole new world of power into the hands of the PHP developer. PHP is a strong and flexible language and, unlike some of its competitors, the batteries *are* included.

Despite PHP often being seen as a non-enterprise language and overlooked for important applications, now that it has the ability to interface with a major enterprise application such as SAP it can be a real choice even where the heavier weight-lifting is left to other platforms.

SAP have produced a somewhat confusing array of web integration tools, among them Internet Transaction Server (ITS), Web Application Server (WAS), SAP Enterprise Portal and SAP Netweaver. These products focus on the "mainstream" enterprise technologies, such as Java and .NET. However, with this extension you can at least create a stepping-stone before these tools are required. It also provides an opportunity to focus on the problem at hand, rather than having to focus on how far you can deploy your application due to licensing restrictions.

Eduard Koucky has done a fantastic job in the creation of this extension. Hopefully, you are now fully charged to go forward and create high value information portals and dynamic reporting tools. Have fun!

**About the Author** ?>

*Jason Simmons is an IT Solutions Consultant who provides IT systems to support business processes. He believes strongly that Open Source technologies will be an ever-increasing business-enabler. Jason does not class himself as a programmer, but he knows enough to get the results required. You can contact him at* Jason@Jsimmons.co.uk

**To Discuss this article:**
**http://forums.phparch.com/190**