Discover how you can interact with Lotus Domino databases from Web applications created in the PHP programming language. Learn how to access Domino applications from PHP pages using a COM object, the Lotus Notes API, and XML.

In this article, you learn how to interact with IBM Lotus Domino databases from Web applications created with the PHP programming language. You also learn how to access Domino applications from PHP pages using a COM object, the IBM Lotus Notes application programming interface (API), and XML. Code examples for each method are included in the Downloads section. The XML method also has a downloadable example application with a simple Web interface to an IBM Lotus Domino mail database (see figure 1).

Figure 1. Example of a PHP user interface to a mail database

PHP is a powerful embedded scripting language. It is free to use and popular for developing dynamic Web applications. PHP code isn't compiled; rather, it's interpreted at run time. Much of its syntax is borrowed from the C, Java, and Perl programming languages. You can add PHP as a Common Gateway Interface (CGI) engine to most Web servers. It even works in the Microsoft Windows and Linux operating systems. In addition, more than 50 percent of Web hosting companies have PHP-enabled servers. With such widespread PHP technology, you simply cannot afford to remain unfamiliar with it.

Working with PHP

You include PHP code in Web pages with the help of special tags. All PHP code begins with the <php tag and ends with the ?> tag:
The Web server interprets this simple example and translates it to following HTML output:

```html
<html>
<body>
  Hello World!
</body>
</html>
```

PHP Web pages can interact with Domino databases in several ways. The choice of integration method depends on which operating system you use to run PHP, the security restrictions in place between PHP and your Domino servers, and what you are allowed to install on the PHP server. The methods described in this article are:

- COM objects
- The Notes API
- XML through the Web

**Running PHP code offline**

Although PHP is primarily intended to run on a Web server and is not for standalone applications, you can run it from the command line as a shell scripting tool similar to Microsoft Visual Basic Scripting Edition (VBScript). This feature can be useful for offline testing before putting the files on the Web server.

**Accessing Lotus Domino**

Three methods are available for accessing a Domino database: through COM objects, through the Notes API, and through XML.

**Access Lotus Domino using Domino COM objects**

The easiest and most fast-forward method of accessing information stored in Domino databases is through Domino COM objects. Using COM from PHP isn't much more difficult than doing so from LotusScript. The following code example shows how to print the field values of all the documents in a view.

```php
<?php
//Initiate Lotus Notes session
$session = new COM( "Lotus.NotesSession" );
$session->Initialize();

//Show the name of the current Notes user
print "Current user: " . $session->CommonUserName . "\n"

//Get database handle
$db = $session->getDatabase( "", "mailtest.nsf" );

//Get view handle using previously received database handle.
//Note that the reserved character in the view name must be \-escaped
$view = $db->getView( "\$Drafts" );
```
//Get first document in view using previously received view handle
$doc = $view->getFirstDocument();

//Loop until all documents in view are processed
while (is_object($doc)) {

//Get handle to a field called "Subject"
$field=$doc->GetFirstItem("Subject");

//Get text value of the field
$fieldvalue=$field->text;

//Show the value of the field
print "Subject: " . $fieldvalue . "\n";

//Get next document in the view
$doc = $view->getNextDocument($doc);
}

//Release the session object
$session = null;
?>

Figure 2 shows the output from running this COM object from the command line.

**Figure 2. Result of running COM object code at a command prompt**

Using COM objects is easy, but there are some disadvantages that make COM rather limited. The main disadvantage is that you must have Lotus Notes client software installed on the PHP server. If you’re using a Web hosting company for your PHP applications, there isn’t much you can do to convince the hosting company to install a Notes client just for purposes of your PHP Web site. If you have your own PHP server or perhaps PHP and Domino servers on the same computer, you should have no problems setting up COM access to the Domino server.

Another issue that limits COM use is that COM objects are supported only on the Windows platform. Considering that most PHP servers run on the Linux platform, this requirement considerably reduces the number of available servers that can host your COM solution, even if you have full administrative control for those servers.

**Access Lotus Domino through an API**

An alternative to COM access is to use the Lotus Notes/Domino C API. Theoretically, you can use this solution on operating systems other than Windows. We recommend that only developers who have a lot of experience in C/C++ programming use it (if you have a reason for not using a COM interface or XML). The functionality provided in the sample API solution is limited, and you must basically write your own code in the C programming language to accomplish most tasks.
The sample library provides a good example for creating PHP extensions. It comes only as source code, and you must have the Visual C compiler and the Domino C API toolkit to compile the code. For your convenience, we've compiled the source code into a DLL file and included it in the Downloads section. You can activate the Notes extension by adding the following line to the php.ini configuration file:

```
extension=php5_notes.dll
```

The following code example shows how to use this technique after you produce the required DLL.

```php
$dbpath="mailtest.nsf";
$searchword="demo";

//Perform full text search in the specified database. //The result is an array of Note IDs
$search=notes_search ( $dbpath, $searchword );

If($search[0]==NULL){
die("No results were found for search word '$searchword'");
}

//Show number of found documents
print "Found " .count($search) . " results for search word '$searchword'
";
?>
```

Figure 3 shows the output from running the API code from the command line.

**Figure 3. Output of the API code**

![Figure 3](image)

**Access Lotus Domino using XML**

Using XML to interact with Domino databases is the most compatible solution. It doesn't require any special adjustments or installations on the PHP Web server, and it works on all operating systems. In our opinion, it's the best approach in most cases.

In the XML solution in this article, you use techniques that are most likely to work in the majority of configurations. As an example, you create a PHP interface to a standard Domino mail database. The only modification made in the Domino environment is a LotusScript agent used to view fields from the mail documents. You use the same agent to parse the XML and to create and send mail.
To view the Inbox folder from the mail database, complete these steps:

1. Ask the user for his or her Notes user name and password.
2. Send a login request to the Domino database using the user name and password provided (see figure 4).

**Figure 4. Login screen in the PHP sample application**

3. From the login response, get a session cookie. (You need this cookie for subsequent requests to the Domino database.)
4. Make a request to the Domino server for the XML source of the Inbox folder, using the ReadViewEntries URL command. This command returns the content of the view presented in XML form instead of returning HTML. You tell Domino your identity by appending a session cookie to the request headers.
5. On the received XML-formatted view content, use PHP's XML parser to find documents and fields.
6. When the XML parser has processed the whole document, construct the HTML code, and print to the browser. To make a mail message a clickable link, use the UNID attribute of each document node.

To access Domino databases, you need functionality both to get a Web page from the Domino server and to parse the XML. You must also assume the following for purposes of the demo application:

- The Domino server is version 6 or later; otherwise, you can't use LotusScript's XML-parsing functionality. (This LotusScript agent is discussed in more detail later.)
- The PHP Web server allows outgoing Web requests.
- The Domino server is accessible from the PHP Web server by either the Internet or an intranet.

**Note:** If you can't access the Domino server from the Internet, you may need to adjust your firewall configuration to allow traffic from the PHP server.

**PHP functions for reading Web-based Domino files**

PHP has several functions for retrieving Web-based content. These functions were included in PHP from the inception, but administrators haven't always activated them. The following functions can retrieve a file from the Web:
• **file_get_contents()**. This function is easy to use, and it supports both the GET and POST request types. With `file_get_contents()`, you get the whole Web file at once as a string.

• **fopen()**. This method reads the Web file in chunks.

• **file()**. This method reads the whole Web file and separates the file's content line by line. It's a kind of cross-over between `file_get_contents()` and `fopen()`.

• **fsockopen()**. This function has the greatest chance of being activated on the hosting Web server. The main difference between this and other functions is that with `fsockopen()`, you must create the whole Web request by yourself instead of setting different parameters.

• **CURL**. The Client URL Library (CURL) has the most advanced functionality. Using CURL, you can connect to and communicate with many different types of servers with many different types of protocols. CURL currently supports HTTP, HTTPS, FTP, Gopher, Telnet, DICT, File, and LDAP. It also supports proxies, cookies, and authentication. Unfortunately, CURL is often disabled by Web hosting companies.

**Note:** In the example application in this article, you use the `file_get_contents()` function exclusively.

Obtaining the Domino session cookie is an important part of the process, and you can use the same code in other Domino-related applications. For example, you can retrieve a cookie for logging in to the IBM Lotus Sametime server using STLinks.

You get the session cookie by sending a POST request to the Domino server. In that request, you include your user name and password. The response from the server contains a cookie that you use in subsequent requests to the Domino server. By including a session cookie, the Domino server thinks that your script is actually a human user who recently logged in and is now accessing the database.

You use the POST method instead of the GET method because URLs of the GET method are logged in the Domino log, and anyone who accesses the log can see the user name and password in plain text. With the POST method, user names and passwords aren't part of the URL; therefore, they aren't visible in regular logs.

You can find the Domino session cookie in the Set-Cookie response header. The session cookie is called either `DomAuthSessId` or `LtpaToken`. You use the name `LtpaToken` when the Domino server is configured for single sign-on (SSO). You don't actually care what the cookie name is; you simply save the whole cookie string.

The following code example shows how to retrieve the cookie.

```php
$req="username=john+doe&password=john123";
$opts = array(
    "http"=>array(
        "method"=>"POST",
        "content" => $req,
        "header"=>"Accept-language: en\n"
    )
);
$context = stream_context_create($opts);
if (!($fp = fopen("http://server.com/maildb.nsf?login",
                "r", false, $context))) {
```
In the code line $req="username=john+doe&password=john123", you provide a valid user name and password to be used for logging in to the Domino server. Then, you set up additional options, such as the POST method type and other headers. After that, you apply your additional options to the outgoing HTTP request:

```
```

From the response you receive from the Domino server, you get all the headers using the stream_get_meta_data($fp) function and loop through the headers until you find the one that contains the Set-Cookie string. After that, you save the cookie by storing it inside a session variable:

```
$_SESSION["DominoCookie"] =$cookie
```

$_SESSION holds the cookie value until you close the Web browser.

Now, you apply the cookie to your next request, which retrieves the Inbox view/folder from a Domino mail database.

```
$opts = array(
    'http'=>array(
        'method'=>"GET",
        'header'=>"Accept-language: en\r\n".
        "User-Agent: Mozilla/4.0 (compatible;
        MSIE 6.0; Windows NT 5.1)\r\n"
        "Cookie: " . $_SESSION["DominoCookie"] . "\r\n"
    )
);
$context = stream_context_create($opts);
$xml = file_get_contents(
    "http://server.com/maildb.nsf/({$Inbox}?ReadViewEntries",
    false, $context);
```

This time, you use the GET method to download a Web page instead of the POST method. This method is specified with the 'method'=>"GET" line in the HTTP options array. Instead of reading only headers, you read the whole response using the file_get_contents(URL, false, context) function. The result of this operation is a long XML string containing all the columns from the Inbox view. You append the session cookie to the header with:

```
"Cookie: " . $_SESSION["DominoCookie"] . "\r\n"
```
After you receive the XML data, you can process it in the PHP XML parser. Then, you create your own user interface for the Inbox view.

**PHP functions for processing XML**

In PHP, just as in the LotusScript and Java programming languages, there are two ways of processing XML code: the Simple API for XML (SAX) and the Document Object Model (DOM). SAX is an event-based programming model, and it’s included and enabled by default in most PHP Web servers. The DOM extension must be activated by the Web server’s administrator. Although DOM is much easier to program with, in these examples, you use the SAX parser because you want to achieve maximum compatibility with most PHP configurations.

Techniques similar to processing Domino views can -- with small modifications -- be used for processing other XML sources, such as RSS feeds and even Web services. The following code example shows the XML source of the Inbox view containing one document.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<viewentries toplevelentries="6">
  <viewentry position="1" unid="38B16601EC8B42BAC12571440068335C" noteid="8FE" siblings="6">
    <entrydata columnnumber="0" name="$109">
      <text></text>
    </entrydata>
    <entrydata columnnumber="1" name="$86">
      <numberlist><number>0</number><number>0</number></numberlist>
    </entrydata>
    <entrydata columnnumber="2" name="$93">
      <text>Donald Duck</text>
    </entrydata>
    <entrydata columnnumber="3" name="$102">
      <number>178</number>
    </entrydata>
    <entrydata columnnumber="4" name="$70">
      <datetime dst="true">20060402T205929,52+02</datetime>
    </entrydata>
    <entrydata columnnumber="5" name="$99">
      <datetime dst="true">20060402T205929,52+02</datetime>
    </entrydata>
    <entrydata columnnumber="6" name="$106">
      <number>823</number>
    </entrydata>
    <entrydata columnnumber="7" name="$97">
      <number>0</number>
    </entrydata>
    <entrydata columnnumber="8" name="$73">
      <text>seen Mickey?</text>
    </entrydata>
  </viewentry>
</viewentries>
```

As you can see, documents in the XML source are presented with the `<viewentry>` tag, and columns (fields) are presented with `<entrydata>` tags. The document's Universal ID is available as an attribute of the `<viewentry>` node.
All you need to do is determine a way to programmatically find the values between those tags to get all the data you need. Sounds easy, and it is easy to accomplish, too. First, initiate the XML parser object using the `xml_parser_create` function:

```php
$xml_parser = xml_parser_create();
```

Then, set callback functions, which are responsible for handling the start and end of XML elements. These functions are identified to the processor through the use of the `xml_set_element_handler` and `xml_set_character_data_handler` functions. Each function accepts the parser handle and the names of the callback functions:

```php
xml_set_element_handler($xml_parser, "startElement", "endElement");
xm_set_character_data_handler($xml_parser, "textData");
```

You must add the `startElement`, `endElement`, and `textData` functions to your PHP code:

```php
function startElement($parser, $name, $attributes) {
    switch($name) {
    case "VIEWENTRY":
        $main = "VIEWENTRY";
        if($attributes["UNID"]!=""){
            $current_doc=$attributes["UNID"];
            $unids[$doc_counter]=$current_doc;
        } 
        break;
    case "ENTRYDATA":
        $main = "ENTRYDATA";
        if($attributes["COLUMNNUMBER"]!=""){
            $current_column=$attributes["COLUMNNUMBER"];
        } 
        break;
    case "DATETIME":
        $main = "DATETIME";
        break;
    default:
        break;
    }
}

function endElement($parser, $name) {
    if ($name == "VIEWENTRY") {
        $doc_counter++;  //increase document counter by 1
    } 
    $current_column="";
}

function textData($parser, $data) {
    switch($main) {
    case "ENTRYDATA":
        if (isset($doc_coll[$current_doc][$main])) {
            $doc_coll[$current_doc][$main][$current_column] .= "$data";
        } else {
            $doc_coll[$current_doc][$main][$current_column] = "$data";
        }
        break;
    case "DATETIME":
        if (isset($doc_coll[$current_doc][$main])) {
            $doc_coll[$current_doc][$main] .= "$data";
        } else {
            $doc_coll[$current_doc][$main] = "$data";
        }
    break;
    case "DATETIME":
    ```
So, let's see how it works. When the XML parser goes through the XML source and finds that a new element is started, it triggers the startElement function, which you specified as a callback function. In this case, such a starting element name is <ENTRYDATA>. Now you know that the current element being processed is a new column, so you can set the $current_column variable to the number of the column. In this case, columns represent fields:

```
case "ENTRYDATA":
    $main = "ENTRYDATA";
    if($attributes["COLUMNNUMBER"]!=""){
        $current_column=$attributes["COLUMNNUMBER"];
    } else {
        $current_column=$current_column;
    } break;
```

After that, the parser finds the text inside the <ENTRYDATA> node and sends it to the textData function that you specified as a callback function. The purpose of this function is to allow you to populate an array with values inside the <ENTRYDATA> node:

```
switch($main) {
    case "ENTRYDATA":
        if (isset($doc_coll[$current_doc][$main])) {
            $doc_coll[$current_doc][$main][$current_column] .= "$data";
        } else {
            $doc_coll[$current_doc][$main][$current_column] = "$data";
        }
        break;
}
```

Using the XML source example, the $doc_coll[$current_doc][$main][$current_column] = "$data" code line is:

```
$doc_coll["38B16601EC8B42BAC1257140068335C"]["ENTRYDATA"]["8"] = "seen Mickey?"
```

This means that the eighth column in the Inbox view for the document with Universal ID 38B16601EC8B42BAC1257140068335C is "seen Mickey?", which happens to be the subject of the mail message.

The XML parser is "smart" and knows that if an element is started, it must be finished somewhere. So, when it finds the end of the element, it signals you by triggering the endElement function (that's right: the same function name you specified as your second callback function). In that function, you reset the column number because it isn't needed anymore. You also check whether the document counter must be increased.

You use the $doc_counter variable as a counter for the array that stores Universal IDs. You must store Universal IDs mainly for the purpose of adding links to the mail documents. When you know that the VIEWENTRY element has ended, it means that a new Universal ID is coming next, and
you had better increase the current array number by 1 to keep the $doc_coll and $unids arrays intact:

```php
if ($name == "VIEWENTRY") {
    $doc_counter++;
}
$current_column="";
```

If you download the sample application and run it on your own server, your version of the Domino mail database may have different placement of columns in the mail views. If so, you must adjust the column numbers in the PHP code (docfunctions.php).

### The sample PHP application and the LotusScript agent

This section is about a Domino agent called ProcessDocWebAction, which is included as part of the sample application you can download with this article. The agent is programmed in LotusScript and uses the DOM to parse incoming XML requests. Because it uses the new XML script classes, you can use the application on Domino release 6 and release 7 servers, but not on release 5 servers.

The requests that this agent handles are initiated from the sample PHP application and can be of the following types:

- Create and send a mail message
- Save an existing mail document
- Save a new mail document as draft
- Request information about the location of a user's file
- A dummy Ping request to make sure that a user is still logged in

To deploy the agent, complete these steps:

1. Copy all the text from the xmlagent.lss file.
2. Create a new agent in a database of your choice.
3. Set the properties of the agent to "Run as Web user" and the target to None.
4. Paste the copied code into this new agent.
5. Name the agent ProcessDocWebAction, and then save it.
6. In the PHP file (dominomail.php), point to the URL location of the database in which the agent resides.

One of our main priorities during the creation of the PHP sample application was to ensure that it worked on all standard mail databases without design modifications. That's why this application doesn't require any changes to the mail database design. You can place the agent for processing the XML outside the user's mail database, and the same agent can be used for serving all users.

When a user authenticates for the first time, the PHP application, after retrieving the session cookie, asks this agent for the location of the user's mail file. The agent knows who the invoker of the agent is because the agent is running with the "Run as Web user" property activated. After
this initial request, the path to the user's mail database is saved for re-use until the user closes the Web browser or the session expires.

A user is always authenticated using real Domino authentication. The user cannot access any resources he or she cannot access using a Web browser client, even if the user could modify the PHP source code on the Web server.

**Conclusion**

If you're a Domino developer or administrator and your Domino server isn't accessible from the Internet, you can still give users a way to access data stored in Domino databases and allow them to read and write email messages. Using the XML techniques described in this article and applied in the sample application, you can easily implement Domino authentication in your own PHP applications to verify logins against the Domino Directory.

There is no overriding of a user's access rights in the Web-based XML solution; the user is using his or her own credentials in each operation. This ensures that users cannot access unauthorized data. To further increase security, you can implement logging of failed login attempts.
# Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source code of PHP application</td>
<td>php_app_sample.zip</td>
<td>10KB</td>
</tr>
<tr>
<td>LotusScript agent used for processing XML</td>
<td>xmlagent.zip</td>
<td>2KB</td>
</tr>
<tr>
<td>Compiled version of Notes API extension</td>
<td>php_notes_dll.zip</td>
<td>21KB</td>
</tr>
</tbody>
</table>
Related topics

- developerWorks article, "Reading and writing the XML DOM with PHP"
- PHP Web site
- PHP: XML Parser Functions
- Try a live demo of the PHP interface demo to a Domino mail database.
- Download a trial version of Lotus Notes 7 from developerWorks.
- Download a trial version of Lotus Domino 7 from developerWorks.
- Download the source code of the Notes API extension for PHP from the PHP Extension Community Library (PECL) site.