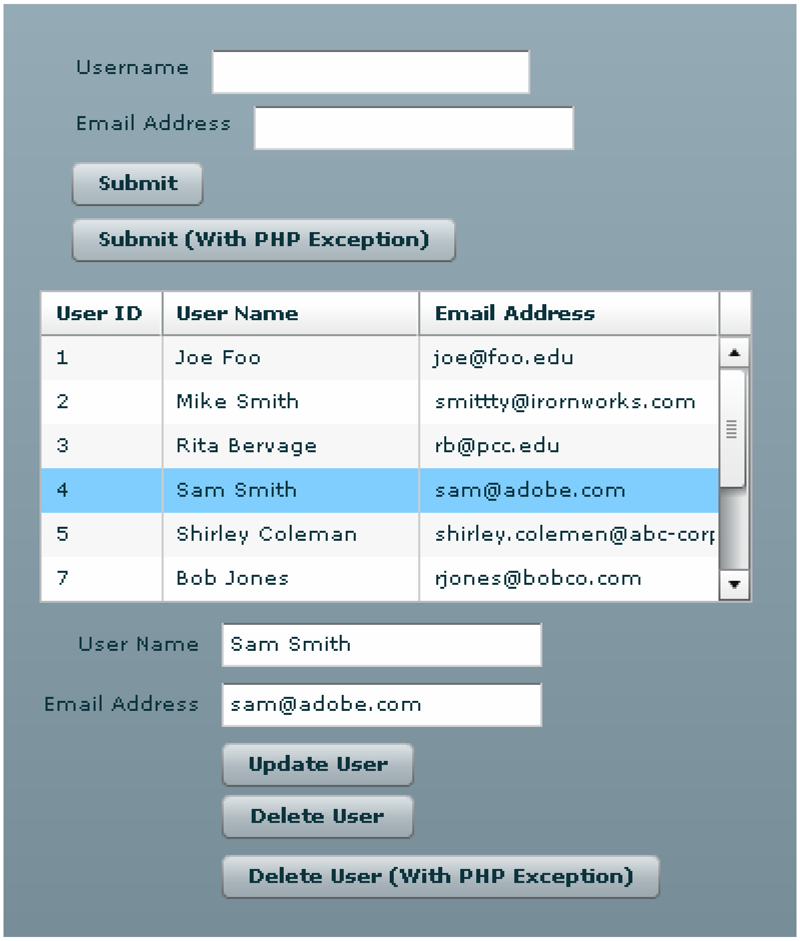
SEAQUEST®
SOFTWARE

www.seaquest.com

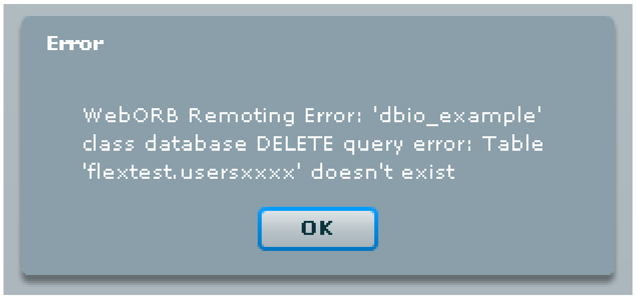
This **simple_weborb_php** example WebORB for PHP application shows you how to message between Flex 2 clients and PHP Web server performing MySQL database I/O. The example shows how PHP generates exceptions and how Flex 2 displays them. The **simple_weborb_php** example can demonstrate this by clicking either or both exception buttons—shown below, with exception error display following.



User ID	User Name	Email Address
1	Joe Foo	joe@foo.edu
2	Mike Smith	smittty@irornworks.com
3	Rita Bervage	rb@pcc.edu
4	Sam Smith	sam@adobe.com
5	Shirley Coleman	shirley.colemen@abc-corp
7	Bob Jones	rjones@bobco.com

User Name: Sam Smith

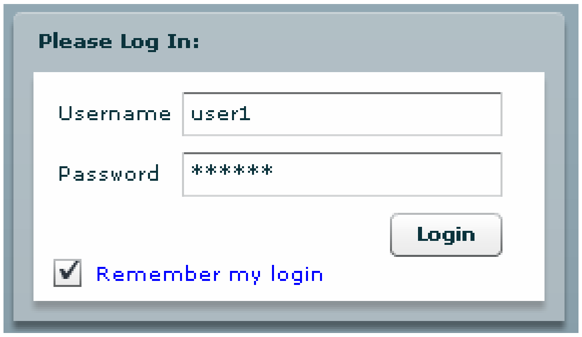
Email Address: sam@adobe.com



WebORB Remoting Error: 'dbio_example' class database DELETE query error: Table 'flextest.usersxxxx' doesn't exist

Note: This install document provides the necessary information to deploy the **simple_weborb_php** examples hosted at Google Code **adobe-php-sdk**: http://code.google.com/p/adobe-php-sdk
We assume the reader of this document is familiar with using Adobe Flex 2 development tools and has experience in administration and deployment of AMP (Apache, MySQL, PHP) system elements. For example, we do not provide the "how to" details about installing MySQL databases and tables.

This example package provides two nearly identical **simple_weborb_php examples**. One without a start-up login dialog and one with a start-up login dialog—shown here.



Please Log In:

Username user1

Password ******

The two separate and independent application example Flex 2 source files are:

```
simple_weborb_php.mxml
simple_weborb_php_logon.mxml
```

The above two Flex 2 simple_weborb_php modules provide examples about:

- Requesting authentication information from the user, i.e., **Username** and **Password** and pass this information from Flex 2 via WebORB for PHP AMF3 messaging to authenticate the credentials from login information contained in a MySQL database. With WebORB for PHP, version 3.0, or newer, you can also authenticate users via parameters such as their IP address or hostname. Presently these examples do not support this capability.

- Log, in the MySQL database, any unauthorized attempts to log in with invalid username or password, including the date and time of the attempted log-in authentication.

- Maintain a log-in session token, using Flex 2 shared objects, to simplify second-time login.

- Displays **User Name** and **Email Address** information from a remote access MySQL database in the Flex 2 data grid.

- The **User Name** and **Email Address** columns in the Flex 2 data grid are editable and updatable to the remote MySQL database via WebORB for PHP AMF3 messaging.

- Provides the user with the ability edit update of **User Name** and **Email Address** data in the Flex 2 data grid—with a commit to the remote MySQL database using AMF3 messaging via WebORB for PHP. Committed MySQL data is immediately posted back to the Flex 2 client's data grid from WebORB for PHP.

- Allows the user to submit new **User Name** and **Email Address** entries to the remote MySQL database using AMF3 messaging via WebORB for PHP. Submitted data is immediately posted back to the Flex 2 client's data grid from WebORB for PHP.

- Allows the user to delete entries in the Flex 2 data grid and post the delete entry to the remote MySQL database using AMF3 messaging via WebORB for PHP. Deleted entry is removed from the Flex 2 data grid display via WebORB for PHP.

Install WebORB for PHP and simple_weborb_php Examples

1. First install WebORB for PHP 1.3 or newer (1.0 will not work) into your Web server root. Expand the distribution archive into the root of your web server.

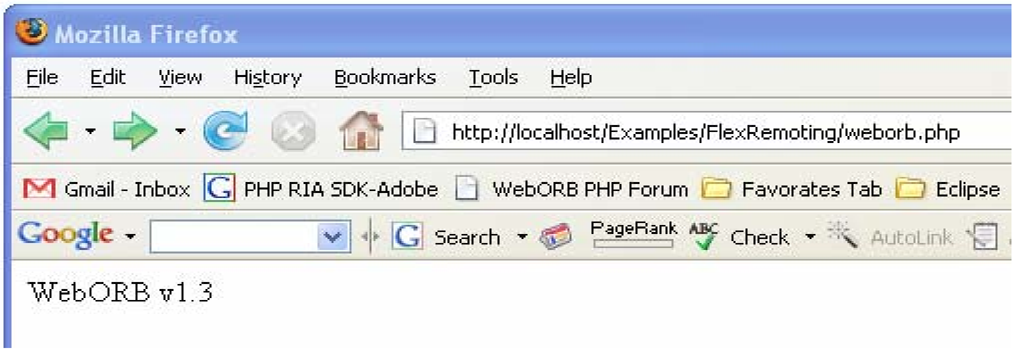
   See the URL— `http://themidnightcoders.com/weborb/php/gettingstarted.htm` —for more WebORB for PHP install layout details.

   Once the WebORB for PHP solution is installed, you should see the WebORB for PHP directory structure as shown at the top of the following page.

2. Run two WebORB for PHP test examples to verify that WebORB for PHP 1.3 is working properly. From your Web browser:

   Run: `http://localhost/Examples/FlexRemoting/weborb.php`

   You should see the WebORB for PHP version displayed on your browser as follows.



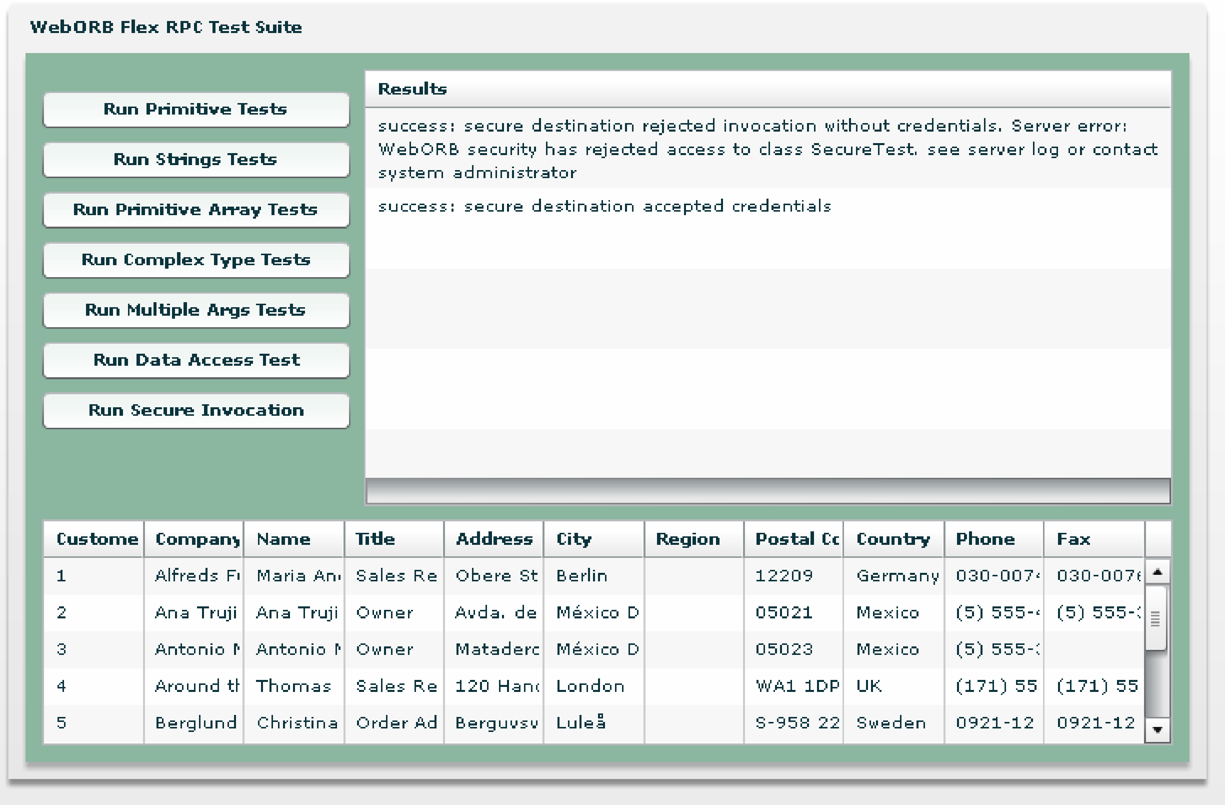
http://localhost/Examples/FlexRemoting/weborb.php

```
WebORB v1.3
```

Install WebORB for PHP and simple_weborb_php Examples (continued)

Run: `http://localhost/Examples/FlexRemoting/main.html`

You should see the WebORB for PHP version displayed on your browser as follows.
Click on the respective **Test Suite** buttons to run the individual tests.



Results

success: secure destination rejected invocation without credentials. Server error: WebORB security has rejected access to class SecureTest. see server log or contact system administrator

success: secure destination accepted credentials

Custome	Company	Name	Title	Address	City	Region	Postal Cc	Country	Phone	Fax
1	Alfreds Fı	Maria Anı	Sales Re	Obere St	Berlin		12209	Germany	030-007	030-007
2	Ana Truji	Ana Truji	Owner	Avda. de	México D		05021	Mexico	(5) 555-	(5) 555-
3	Antonio M	Antonio M	Owner	Matadero	México D		05023	Mexico	(5) 555-	
4	Around th	Thomas	Sales Re	120 Hano	London		WA1 1DP	UK	(171) 55	(171) 55
5	Berglund	Christina	Order Ad	Berguvsv	Luleå		S-958 22	Sweden	0921-12	0921-12

Note: Stop here if one or both of the above two WebORB for PHP test examples do not work. You should not go ahead to install and run the two **simple_weborb_php** examples if WebORB for PHP is not running properly by displaying the above two test's example output.

3. Unzip the `“simple_weborb_php.zip`” file.

4. Place the enclosed `"adobe_php_sdk"` folder into your `web root.../Services` folder, as show on the next page (5) in **bold red**

Note: Web server **simple_weborb_php** installs are shown in **`bold red`** in the following, page 5, WebORB for PHP install path hierarchy diagram.

Install WebORB for PHP and simple_weborb_php Examples (continued)

```
+-web server root
|
+--/Services  —  Contains deployed ‘remotable’ PHP classes
|    |
|    +--/adobe_php_sdk
|         |
|         +--/simpleWebOrbPHP
|              |
|              +-dbio_example.php — The PHP class to perform the database I/O
|              |
|              +-flextest.sql      —  SQL to create “flextest” example database.
|              |
|              +-test-dbio_example_object.php —  PHP test stub for direct testing
|                                                of class methods in:
|                                                ‘dbio_example.php’
+--/Examples  —  Contains examples shipped with WebORB
|    |           (Below is a partial Examples folder hierarchy expansion.)
|    |
|    +--/FlexRemoting
|    |    |
|    |    +--/srcview
|    |         |
|    |         +--/source
|    |
|    +--/SampleApp
|    |
|    +-main.html
|    |
|    +-weborb.php
|    |
|    +-......
|    |
|    +-......
|
+--/Weborb    —  Contains configuration, log and WebORB for PHP source code.
     |
     |
     +--weborb-config.xml  —  Contains a reference to the /Services folder, as well
     |                         as other important WebORB configuration data.
     |
     |
     +--weborb-log.txt  —  The WebORB log file.
     |
     +--/WEB-INF
          |
          +--/flex
               |
               +--remoting-config.xml  —  Lists deployed Flex RPC services.
               |                |
               +--services-config.xml  —  Configures Flex RPC endpoint.
```

Install WebORB for PHP and simple_weborb_php Examples (continued)

5. Using your preferred MySQL client, create the required “flextest” example database by running “flextest.sql” SQL schema file located in your Web site “Services” folder. See directory structure diagram, as shown above on page 5, for location of the “flextest.sql” SQL schema file

   Note: If you have previously installed the AMFPHP example titled “simple_amfphp” from the Google Code **adobe-php-sdk** example repository, then uncomment all of the **#DROP TABLE IF EXISTS `xxxx`** SQL statements in “flextest.sql” before running to create new database tables. This will replace your present MySQL tables plus add some new database table fields.

6. With Adobe Flex Builder 2, create a Flex 2 project following the instructions at:
   http://themidnightcoders.com/weborb/php/gettingstarted.htm
   from the section: **GETTING STARTED - CREATING A FLEX APPLICATION**

   Name the Eclipse project “simple_weborb_php”.

7. Replace the new Eclipse project generated “simple_weborb_php.mxml” file with the one provided in your Zip file package.

   In your Eclipse Navigator window, right-click on “simple_weborb_php.mxml” and select Set as Default Application from the pop-up menu.

   Note: This **simple_weborb_php** example is without a start-up login dialog.

   With Adobe Flex Builder 2, you can optionally run and observe the example’s behavior. From your Eclipse IDE, **Run** => **Run As** => **Flex Application**, and observe **simple_weborb_php** operation.

8. Next you will install and deploy the **simple_weborb_php_login** example.
   Place the “simple_weborb_php_login.mxml” file, provided in the Zip file package, in the root of the **simple_weborb_php** Eclipse project folder (by copy / paste into your Eclipse **Navigator** window).

   Note: This **simple_weborb_php_login** is the example with the start-up login dialog.

9. That’s it. From your Eclipse IDE select, **Run** => **Run As** => **Flex Application**, and observe **simple_weborb_php** operation.

   Do you see a “channel disconnected error? If yes, then see the note about this on page 9.

10. Next, you will run the **simple_weborb_php_login** example.
    Close your running **simple_weborb_php** example.
    In your Eclipse Navigator window, right-click on “simple_weborb_php_login.mxml” and select **Set as Default Application** from the pop-up menu.

Install WebORB for PHP and simple_weborb_php Examples (continued)

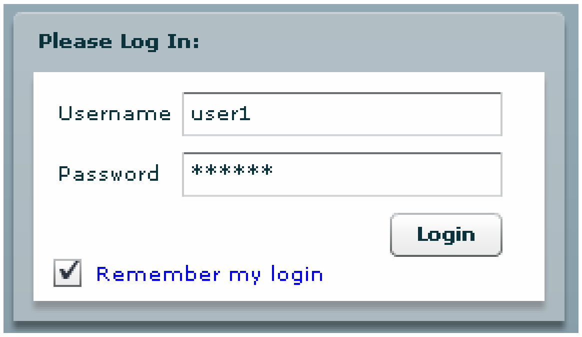
11. Again....From your Eclipse IDE select, **Run** => **Run As** => **Flex Application**, and observe **simple_weborb_php_login** operation.

    Note: You will need a username and password to log in. MySQL database pre-installed log-in access parameters are:

Username	Password
admin	admin
user1	user11
user2	user22
user3	user33

Using Flex 2 Shared Objects

Note in the **simple_weborb_php_login** example start up dialog, that you can optionally remember you log-in parameters for the next log-in.



Please Log In:

Username: user1

Password: ******

☑ Remember my login

This functionality is provided by using Flex 2 shared objects residing on your client—not the remote server. To see how Flex 2 shared objects are used for login in the example code, search for the word `loginSO` using your Eclipse IDE editor on the file `“simple_weborb_php_login.mxml”`.

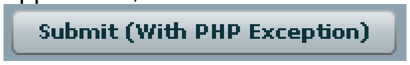
Flex 2 shared objects provide powerful and convenient application functionality for applications. They are most easy to deploy. I encourage you to think about ways you might use shared objects for things like forms content posting for subsequent user visits to the form.

Further details about what Flex 2 shared objects can provide for you, can be found in the Adobe document, *Adobe Flex 2 Developer’s Guide*, **Chapter 34**: Using Shared Objects, page 955. A PDF version of this document is available on the Adobe Web site: `http://www.adobe.com/support/documentation/en/flex`

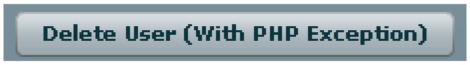
Handling PHP Generated Exceptions in Flex 2

The two **simple_weborb_php** examples handle and display MySQL database I/O exceptions rather nicely. MySQL database I/O exceptions are displayed in an Alert dialog. What's neat is that the example is displaying the exact exception message generated by the `“mysqli”` database access PHP extension. While the `“mysqli”` generated error messages are much too wordy and technical for general user population consumption, these two**simple_weborb_php** examples do show what can be provided in the way of detailed I/O system information from PHP, which can be displayed by Flex 2.

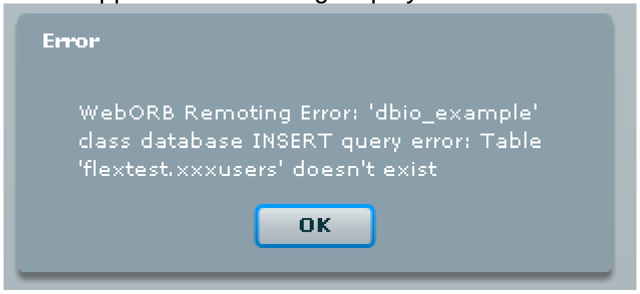
For a demonstration of PHP exception handling and MySQL error reporting back to the Flex 2 application, click on one or both of these buttons:



or

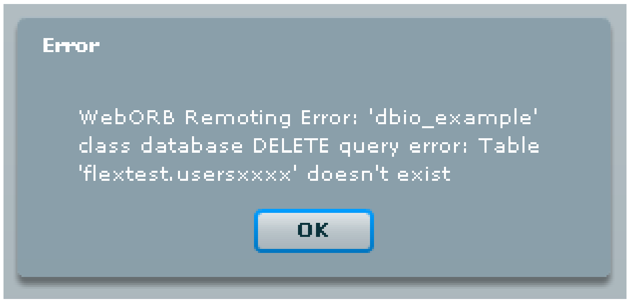


These buttons call one of two object methods in the `“dbio_example.php”` module. See page 5 for the directory path location of the PHP class module, which is located in WebORB for PHP **Services** folder. These two PHP methods, submitUserException and deleteUserException, are hard coded to always produce a PHP exception and return the exception message back to the Flex 2 application for dialog display. The returned exception messages will display as:



WebORB Remoting Error: 'dbio_example' class database INSERT query error: Table 'flextest.xxxusers' doesn't exist

and



WebORB Remoting Error: 'dbio_example' class database DELETE query error: Table 'flextest.usersxxxx' doesn't exist

Review the submitUserException and deleteUserException PHP object method code in the `“dbio_example.php”` module to master what is happening with these always forced exceptions.

So why are handling and displaying PHP method exceptions from WebORB for PHP so important? A lot of things can happen on the remote server. For one example, the MySQL server might not even be running due to a sysadmin restart error, etc. or, heaven forbid, a database table was inadvertently deleted. In these cases you never want to leave the remote user, running a client Flash application, hanging in a lurch. You must always keep the user apprised of server problems—especially if it is some straightforward like the user simply clicking the wrong button.

Always practice full and complete exception handing and client message posting within your WebORB for PHP object methods. Shame on you, if you do not.

If I go to post my shopping cart that just took me thirty minutes to generate and my Flash client hangs with no indication of what is wrong on the server, my inclination will be to think what a worthless ecommerce site. "I'll never again visit here."

What is the "test-dbio_example_object.php" file for?

Initial debugging of Flex 2 to/from WebORB for PHP (PHP AMF Gateway) messaging can be a real pain to debug. If your messaging is not working, is it the transmitter or receiver? There's no need to be debugging messaging problems on both client and server simultaneously. One way to reduce the PHP messaging debugging is to create a PHP test stub to locally evaluate proper operation of your from WebORB for PHP messaging object. The enclosed file `"test-dbio_example_object.php"` is a reference example of a test stub to run against the `"dbio_example.php"` PHP 5 object to locally test proper PHP messaging operation.

I strongly recommend always performing PHP CLI[1] stand-alone testing of your WebORB for PHP objects methods before attempting messaging via Flex 2. Use the enclosed test stub to develop test messaging for your WebORB for PHP objects. Uncomment respective sections and run against the PHP class via the command line to appreciate what it provides. (Run with both PHP modules in the same folder.)

What to do if you see a "channel disconnected" error when running either example

This is very common initial example deployment error, which is coming from WebORB for PHP. Most likely the error is because you do not have your WebORB PHP code installed properly.

The problem is that the PHP class cannot instantiate for some PHP code syntax reason. When this happens, run your PHP WebORB class against a PHP test stub. The problem will readily jump out via syntax error messages—thus easily fixable.

Flex 2 to / from remote PHP means you have both the client and server working against you when it comes to a problem. I've found it best to first thoroughly test your WebORB class with a PHP test stub—including exception handling—before attempting to remote message via Flex 2. See above, **What is the "test-dbio_example_object.php" file for?** This way you are nearly 100% fighting problems only on the Flex 2 client side by reducing the wondering if your problem is on the server.

I will not use WebORB PHP class, even if one line is added or changed, without first testing against a PHP test stub. Simply, it's too much hassle to be debugging a problem and simultaneously be wondering if it exists on the client or server side. Divide and conquer your client / server anomalies.

Pete Mackie
Seaquest Software
pete@seaquest.com
http://www.seaquest.com

[1] **Using PHP from the command line**: http://us2.php.net/manual/en/features.commandline.php