```
 _     ___ __  __ _____  _____                          _
| |   |_ _|  \/  / ___|| ____|_  ___ __   ___ _ __| |_
| |    | || |\/| \___ \|  _| \ \/ / '_ \ / _ \ '__| __|
| |___ | || |  | |___) | |___ >  <| |_) |  __/ |  | |_
|_____|___|_|  |_|____/|_____/_/\_\ .__/ \___|_|   \__|
                                  |_|
 _____                 _ _       _     _
/ ___|  ___ _ __(_) |__ | |__ | | ___  ___
\___ \ / __| '__| | '_ \| '_ \| |/ _ \/ __|
 ___) | (__| |  | | |_) | |_) | |  __/\__ \
|____/ \___|_|  |_|.__/|_.__/|_|\___||___/
```

About LIMSExpert Scribbles

LIMSExpert.com Scribbles are short lab reports. They cover a particular lab in detail by providing the code in line-numbered format accompanied by an analysis of each file.

This document is created by a script so code formatting might be wrong. Do not copy this code to run. Instead visit the LIMSExpert.com website Downloads page and use the tar zipped version.

Line-by-line discussion follows the line-numbered code.

Copyright Notice
==========

~~~~~~~~~~~~~~~~~~
START
~~~~~~~~~~~~~~~~~~

./lab-a-series/aclexample.php

```
     1  <?php
     2       /*
     3            ACLEXAMPLE.PHP produced for www.LIMSExpert.com
     4            License: http://opensource.org/licenses/MIT
     5       */
     6
     7       define('CRLF',"\n");
     8       require_once('./lib/UserAccess.php');
     9       require_once('./lib/DataAccessor.php');
    10       require_once('./lib/GenericResource.php');
    11
    12
    13       /* Direct access test -- notice that it hands out the information */
    14
    15       $anObj = new GenericResource('equipment');
```

```
    16       echo 'Direct access request example: ' . $anObj->resourceName . CRLF;
    17
    18       /* Indirect access test -- rightly refuses ... I donâ\200\231t have the right
groups! */
    19
    20       $userAccess = new UserAccess('sally','Sally Koch'); /* Like a key */
    21       $dataAccess = new DataAccessor($anObj, $userAccess); /* Like the lock the key
goes into */
    22       echo 'Using a data accessor: ' . $dataAccess->get('resourceName') . CRLF;
    23
    24  ?>
```

Analysis

Line 7: We really only need this because we are running these examples from the command line.

Lines 8 - 10: Here we require from a central location instead of having to guess where a library might be. PHP allows for relative requires (a require that will load from the current script's location) but this does not encourage good coding practice. When we get into configuration files we will be very explicit about where the libraries are kept so that any class can find any other class.

Lines 15,16: Here we instantiate a GenericResource object and then use it to fetch the information. This is an example of what not to allow.

Line 20: Here we instantiate a UserAccess object which, from the notes, you can see is a bit like a key. When people want to access things that are secure they normally need keys. This fetches security related information for the user with the identity 'sally.'

Line 21: This instantiates a DataAccessor object that takes the GenericResource object and the key and does the work of fetching the information. If the key is bad you should get an error (error handling omitted for brevity).

Line 22: Finally we try to fetch some property of the GenericResource object. The DataAccessor acts as a gateway, denying the access.

./lab-a-series/lib/DataAccessor.php

```
     1  <?php
     2       /*
     3            DATAACCESSOR.PHP produced for www.LIMSExpert.com
     4            License: http://opensource.org/licenses/MIT
     5       */
     6
     7       class DataAccessor {
     8            var $desiredResource = null;
     9            var $whoIsAsking     = null;
    10            function DataAccessor($resourceObject, $userAccessObj) {
    11                 $this->desiredResource = $resourceObject;
    12                 $this->whoIsAsking     = $userAccessObj;
    13            }
    14            function get($property){
    15              if ($this->desiredResource != null) {
    16                   if($this->userHasAccess()) {
    17                     return $this->desiredResource->$property;
    18                   } else {
    19                     return 'Unauthorized.';
    20                   }
    21              } else {
    22                   return 'Not properly initialized.';
    23              }
    24            }
```

```
    25            function userHasAccess() {
    26                 if($this->whoIsAsking != null) {
    27                   for($i = 0; $i<count($this->desiredResource->acl); $i++)
    28                   {
    29                     for($m = 0; $m<count($this->whoIsAsking->groups); $m++) {
    30                       if ($this->desiredResource->acl[$i] ==
$this->whoIsAsking->groups[$m]) {
    31                         return true ;
    32                       }
    33                     }
    34                   }
    35                 } else {
    36                     return false;
    37                 }
    38            }
    39       }
    40
    41
    42
    43
    44  ?>
```

Analysis

Line 7: PHP has object oriented capabilities, so it makes sense to put reusable things in classes.

Lines 8 - 9: Declare our properties. These are publicly scoped so anyone using this class can reference them. The use of the var keyword does this. You could specify the scope explicitly using public, private, or protected. See the PHP manual for details.

Line 10: Constructor. Constructor syntax in PHP has undergone some revisions over time. Depending on which version of PHP you are using and whether your class is in a namespace you can use a different syntax. According to the documentation (see http://www.php.net) this syntax would be treated as a constructor in a namespaced class in version 5.3.0-5.3.2 but as a regular method on 5.3.3 and beyond. This code was originally tested on version 5.5.9. It is *not* explicitly placed in a namespace, and yet the code runs the method as a constructor. Hence, this syntax works but is probably unreliable. In the future the __construct() syntax will be used for safety.

Lines 11,12: Set the passed-in variables to the object’s properties.

Line 14: Our generic getter property takes in a property name, checks to see if we have something in our desiredResource property and whether the user has access.

Line 17: If all is okay then return the property using some very interesting syntax. A great test would be to see what happens if the desired property did not exist or was privately scoped.

Line 19: If the userHasAccess() method fails then we should return some kind of error message. We could raise an error as well but for our purposes this will suffice.

Line 22: Because we have a loosely-typed language it would have been possible to create the object using null as the property and no error would have been issued. Hence we check for it here.

Lines 25 - 38: After ensuring that there is something in the whoIsAsking property the code tries to find at least one matching group between the user’s list of groups and the resources. Now, we only need all of this code if we presume that a resource could have more than one group, which in practice it really should not. The resource should be created using the creator’s default group -- a property of an operator that can create data resources. We

will clean this code up in the future.

./lab-a-series/lib/GenericResource.php

```
 1  <?php
 2
 3       /*
 4            GENERICRESOURCE.PHP produced for www.LIMSExpert.com
 5            License: http://opensource.org/licenses/MIT
 6       */
 7
 8       class GenericResource {
 9            var $resourceName = '';
10            /* should read this from a table upon instantiation */
11            var $acl = array();
12            function GenericResource($resource) {
13                 $this->resourceName = $resource;
14                 /* Should be handled by code that knows the structure of the
15                    data -- this is just an example */
16                 if ($resource == 'equipment')
17                 {
18                      $this->acl =
array('RAWMAT','LABADMIN','MANUFACTURING','ACCOUNTING');
19                 }
20            }
21       }
22  ?>
```

Analysis

Line 11: This should be replaced with code that fetches from the database, BUT it should only return a single entry. A resource must not have more than one group.

Line 16: This is tricky. A GenericResource could literally refer to any setup/master type data in the system. Here our example has 'equipment' but it could be a list of operators, elements in the periodic table, etc. As we move into the persistent data storage discussion for our labs we will have to address this in more detail later.

./lab-a-series/lib/UserAccess.php

```
 1  <?php
 2
 3       /*
 4            USERACCESS.PHP produced for www.LIMSExpert.com
 5            License: http://opensource.org/licenses/MIT
 6       */
 7
 8       class UserAccess {
 9            var $userId   = '';
10            var $longname = '';
11            /* should read this from a table upon instantiation */
12            var $groups   = array('OMANUFACTURING','RACCOUNTING');
13            function UserAccess($id) {
14                 $this->userId = $id;
15            }
16       }
17
18  ?>
```

Analysis

Line 12: Similar to the GenericResource class but here we know that each user will normally belong to numerous groups. These should be fetched from the database and likely stored in the user's session. We will cover sessions more later as well.

./lab-a-series/test/test_main.php

```
 1  <?php
 2       /*
 3            TEST_MAIN.PHP produced for www.LIMSExpert.com
 4            License: http://opensource.org/licenses/MIT
 5
 6            Notes
 7            -----
 8
 9            1. Tests are to be run from the /test directory.
10            2. You'll need to get a copy of simpletest
(http://www.simpletest.org/en/first_test_tutorial.html) to use these tests. Extract it to the
/tmp directory.
11
12       */
13
14       require_once('/tmp/simpletest/autorun.php');
15
16       require_once('../lib/UserAccess.php');
17       require_once('../lib/DataAccessor.php');
18       require_once('../lib/GenericResource.php');
19
20       class BasicAccessTest extends UnitTestCase {
21            function TestOfBasicAccess() {
22                 $dataResource = new GenericResource('equipment');
23                 $userKey = new UserAccess('sally','parsons');
24                 $dataAccess = new DataAccessor($dataResource, $userKey);
25
26                 assert($dataAccess->get('resourceName') == 'Unauthorized.');
27
28            }
29       }
30  ?>
```

Analysis

Line 14: Simpletest should be extracted into the /tmp directory for this example to work. You can get simpletest by visiting www.simpletest.org.

Lines 15 - 18: We are running our tests from the /test directory itself so the libraries are not normally where they would be. This is important because we want to avoid accidentally creating a dependency between code in our test directory and anything else.

Line 20: Refer to simpletest's documentation for the correct syntax for a test.

Lines 22 - 26: Notice that this essentially replaces aclexample.php. We want to migrate away from simple examples and hand-coded tests and move toward fully automated regression tests that can be run without human intervention. So the only reason you would write example code would be to teach other developers how to use your code. It is not a replacement for comprehensive testing.

Wrap-up
=======

The aclexample originally developed was moved to a set of separate libraries and automated

testing capability was added.

Feedback
========

Visit the www.limsexpert.com Contacts page for contact information if you have questions or comments about this scribble.