```php
<?php
/**
 * Advanced Dummy ILS Driver -- Returns sample values based on Solr index.
 *
 * Note that some sample values (holds, transactions, fines) are stored in
 * the session.  You can log out and log back in to get a different set of
 * values.
 *
 * PHP version 5
 *
 * Copyright (C) Villanova University 2007.
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2,
 * as published by the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 * @category VuFind
 * @package  ILS_Drivers
 * @author   Greg Pendlebury <vufind-tech@lists.sourceforge.net>
 * @license  http://opensource.org/licenses/gpl-2.0.php GNU General Public License
 * @link     http://vufind.org/wiki/building_an_ils_driver Wiki
 */
require_once 'Interface.php';

/**
 * Advanced Dummy ILS Driver -- Returns sample values based on Solr index.
 *
 * @category VuFind
 * @package  ILS_Drivers
 * @author   Greg Pendlebury <vufind-tech@lists.sourceforge.net>
 * @license  http://opensource.org/licenses/gpl-2.0.php GNU General Public License
 * @link     http://vufind.org/wiki/building_an_ils_driver Wiki
 */
class Demo implements DriverInterface
{
    // Used when getting random bib ids from solr
    private $_db;
    private $_totalRecords;

    /**
     * Constructor
     */
    public function __construct()
    {
        // Establish an array in the session for persisting fake data (to save
        // on Solr hits):
        if (!isset($_SESSION['demoData'])) {
            $_SESSION['demoData'] = array();
```

```
        }
    }

    /**
     * Generate a fake location name.
     *
     * @return string
     * @access private
     */
    private function _getFakeLoc()
    {
        $loc = rand()%3;
        switch ($loc) {
        case 0:
            return "Campus A";
        case 1:
            return "Campus B";
        case 2:
            return "Campus C";
        }
    }

    /**
     * Generate a fake status message.
     *
     * @return string
     * @access private
     */
    private function _getFakeStatus()
    {
        $loc = rand()%10;
        switch ($loc) {
        case 10:
            return "Missing";
        case  9:
            return "On Order";
        case  8:
            return "Invoiced";
        default:
            return "Available";
        }
    }

    /**
     * Generate a fake call number.
     *
     * @return string
     * @access private
     */
    private function _getCallNum()
    {
        $codes = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        $a = $codes[rand()%strlen($codes)];
        $b = rand()%899 + 100;
        $c = rand()%9999;
        return $a.$b."."$c;
    }
```

```php
    /**
     * Set up the Solr index so we can retrieve some record data.
     *
     * @return void
     * @access private
     */
    private function _prepSolr()
    {
        // Create or solr connection
        $this->_db = ConnectionManager::connectToIndex();

        // Get the total # of records in the system
        $result = $this->_db->search('*:*');
        $this->_totalRecords = $result['response']['numFound'];
    }

    /**
     * Get a random ID from the Solr index.
     *
     * @return string
     * @access private
     */
    private function _getRandomBibId()
    {
        // Let's keep away from both ends of the index
        $result = $this->_db->search(
            '*:*', null, null, rand()%($this->_totalRecords-1), 1
        );
        return $result['response']['docs'][0]['id'];
    }

    /**
     * Get Status
     *
     * This is responsible for retrieving the status information of a certain
     * record.
     *
     * @param string $id The record id to retrieve the holdings for
     *
     * @return mixed     On success, an associative array with the following keys:
     * id, availability (boolean), status, location, reserve, callnumber; on
     * failure, a PEAR_Error.
     * @access public
     */
    public function getStatus($id)
    {
        // How many items are there?
        $records = rand()%15;
        $holding = array();

        // NOTE: Ran into an interesting bug when using:

        // 'availability' => rand()%2 ? true : false

        // It seems rand() returns alternating even/odd
        // patterns on some versions running under windows.
```

```
        // So this method gives better 50/50 results:

        // 'availability' => (rand()%100 > 49) ? true : false

        // Create a fake entry for each one
        for ($i = 0; $i < $records; $i++) {
            $holding[] = array(
                'id'           => $id,
                'number'       => $i+1,
                'barcode'      => sprintf("%08d", rand()%50000),
                'availability' => (rand()%100 > 50) ? true : false,
                'status'       => $this->_getFakeStatus(),
                'location'     => $this->_getFakeLoc(),
                'reserve'      => (rand()%100 > 49) ? 'Y' : 'N',
                'callnumber'   => $this->_getCallNum(),
                'duedate'      => '',
                'is_holdable'  => true,
                'addLink'      => rand()%10 == 0 ? 'block' : true
            );
        }
        return $holding;
    }

    /**
     * Get Statuses
     *
     * This is responsible for retrieving the status information for a
     * collection of records.
     *
     * @param array $ids The array of record ids to retrieve the status for
     *
     * @return mixed     An array of getStatus() return values on success,
     * a PEAR_Error object otherwise.
     * @access public
     */
    public function getStatuses($ids)
    {
        // Random Seed
        srand(time());

        $status = array();
        foreach ($ids as $id) {
            $status[] = $this->getStatus($id);
        }
        return $status;
    }

    /**
     * Get Holding
     *
     * This is responsible for retrieving the holding information of a certain
     * record.
     *
     * @param string $id     The record id to retrieve the holdings for
     * @param array  $patron Patron data
     *
     * @return mixed     On success, an associative array with the following keys:
```

```
     * id, availability (boolean), status, location, reserve, callnumber, duedate,
     * number, barcode; on failure, a PEAR_Error.
     * @access public
     */
    public function getHolding($id, $patron = false)
    {
        return $this->getStatus($id);
    }

    /**
     * Get Purchase History
     *
     * This is responsible for retrieving the acquisitions history data for the
     * specific record (usually recently received issues of a serial).
     *
     * @param string $id The record id to retrieve the info for
     *
     * @return mixed     An array with the acquisitions data on success, PEAR_Error
     * on failure
     * @access public
     */
    public function getPurchaseHistory($id)
    {
        return array();
    }

    /**
     * Patron Login
     *
     * This is responsible for authenticating a patron against the catalog.
     *
     * @param string $barcode  The patron barcode
     * @param string $password The patron password
     *
     * @return mixed           Associative array of patron info on successful login,
     * null on unsuccessful login, PEAR_Error on error.
     * @access public
     */
    public function patronLogin($barcode, $password)
    {
        $user = array();

        $user['id']           = trim($barcode);
        $user['firstname']    = trim("Lib");
        $user['lastname']     = trim("Rarian");
        $user['cat_username'] = trim($barcode);
        $user['cat_password'] = trim($password);
        $user['email']        = trim("Lib.Rarian@library.not");
        $user['major']        = null;
        $user['college']      = null;

        return $user;
    }

    /**
     * Get Patron Profile
     *
```

```
     * This is responsible for retrieving the profile for a specific patron.
     *
     * @param array $patron The patron array
     *
     * @return mixed        Array of the patron's profile data on success,
     * PEAR_Error otherwise.
     * @access public
     */
    public function getMyProfile($patron)
    {
        $patron = array(
            'firstname' => trim("Lib"),
            'lastname'  => trim("Rarian"),
            'address1'  => trim("Somewhere ..."),
            'address2'  => trim("Other the Rainbow"),
            'zip'       => trim("12345"),
            'phone'     => trim("1900 CALL ME"),
            'group'     => trim("Library Staff")
        );
        return $patron;
    }

    /**
     * Get Patron Fines
     *
     * This is responsible for retrieving all fines by a specific patron.
     *
     * @param array $patron The patron array from patronLogin
     *
     * @return mixed        Array of the patron's fines on success, PEAR_Error
     * otherwise.
     * @access public
     */
    public function getMyFines($patron)
    {
        if (!isset($_SESSION['demoData']['fines'])) {
            // How many items are there? %20 - 2 = 10% chance of none,
            // 90% of 1-18 (give or take some odd maths)
            $fines = rand()%20 - 2;

            // Do some initial work in solr so we aren't repeating it inside this
            // loop.
            $this->_prepSolr();

            $fineList = array();
            for ($i = 0; $i < $fines; $i++) {
                // How many days overdue is the item?
                $day_overdue = rand()%30 + 5;
                // Calculate checkout date:
                $checkout = strtotime("now - ".($day_overdue+14)." days");
                // 50c a day fine?
                $fine = $day_overdue * 0.50;

                $fineList[] = array(
                    "amount"   => $fine * 100,
                    "checkout" => date("j-M-y", $checkout),
                    // After 20 days it becomes 'Long Overdue'
```

```
                "fine"     => $day_overdue > 20 ? "Long Overdue" : "Overdue",
                // 50% chance they've paid half of it
                "balance"  => (rand()%100 > 49 ? $fine/2 : $fine) * 100,
                "duedate"  =>
                    date("j-M-y", strtotime("now - $day_overdue days")),
                "id"       => $this->_getRandomBibId()
            );
        }
        $_SESSION['demoData']['fines'] = $fineList;
    }
    return $_SESSION['demoData']['fines'];
}

/**
 * Get Patron Holds
 *
 * This is responsible for retrieving all holds by a specific patron.
 *
 * @param array $patron The patron array from patronLogin
 *
 * @return mixed        Array of the patron's holds on success, PEAR_Error
 * otherwise.
 * @access public
 */
public function getMyHolds($patron)
{
    if (!isset($_SESSION['demoData']['holds'])) {
        // How many items are there?  %10 - 1 = 10% chance of none,
        // 90% of 1-9 (give or take some odd maths)
        $holds = rand()%10 - 1;

        // Do some initial work in solr so we aren't repeating it inside this
        // loop.
        $this->_prepSolr();

        $holdList = array();
        for ($i = 0; $i < $holds; $i++) {
            $holdList[] = array(
                "id"       => $this->_getRandomBibId(),
                "location" => $this->_getFakeLoc(),
                "expire"   => date("j-M-y", strtotime("now + 30 days")),
                "create"   =>
                    date("j-M-y", strtotime("now - ".(rand()%10)." days")),
                "reqnum"   => sprintf("%06d", $i),
                "item_id" => $i
            );
        }
        $_SESSION['demoData']['holds'] = $holdList;
    }
    return $_SESSION['demoData']['holds'];
}

/**
 * Get Patron Transactions
 *
 * This is responsible for retrieving all transactions (i.e. checked out items)
 * by a specific patron.
```

```
     *
     * @param array $patron The patron array from patronLogin
     *
     * @return mixed        Array of the patron's transactions on success,
     * PEAR_Error otherwise.
     * @access public
     */
    public function getMyTransactions($patron)
    {
        if (!isset($_SESSION['demoData']['transactions'])) {
            // How many items are there?  %10 - 1 = 10% chance of none,
            // 90% of 1-9 (give or take some odd maths)
            $trans = rand()%10 - 1;

            // Do some initial work in solr so we aren't repeating it inside this
            // loop.
            $this->_prepSolr();

            $transList = array();
            for ($i = 0; $i < $trans; $i++) {
                // When is it due? +/- up to 15 days
                $due_relative = rand()%30 - 15;
                // Due date
                if ($due_relative >= 0) {
                    $due_date = date("j-M-y", strtotime("now +$due_relative days"));
                } else {
                    $due_date = date("j-M-y", strtotime("now $due_relative days"));
                }

                // Times renewed    : 0,0,0,0,0,1,2,3,4,5
                $renew = rand()%10 - 5;
                if ($renew < 0) {
                    $renew = 0;
                }

                // Pending requests : 0,0,0,0,0,1,2,3,4,5
                $req = rand()%10 - 5;
                if ($req < 0) {
                    $req = 0;
                }

                $transList[] = array(
                    'duedate' => $due_date,
                    'barcode' => sprintf("%08d", rand()%50000),
                    'renew'   => $renew,
                    'request' => $req,
                    "id"      => $this->_getRandomBibId(),
                    'item_id' => $i,
                    'renewable' => true
                );
            }
            $_SESSION['demoData']['transactions'] = $transList;
        }
        return $_SESSION['demoData']['transactions'];
    }

    /**
```

```
     * Get Funds
     *
     * Return a list of funds which may be used to limit the getNewItems list.
     *
     * @return array An associative array with key = fund ID, value = fund name.
     * @access public
     */
    public function getFunds()
    {
        return array("Fund A", "Fund B", "Fund C");
    }

    /**
     * Get Departments
     *
     * Obtain a list of departments for use in limiting the reserves list.
     *
     * @return array An associative array with key = dept. ID, value = dept. name.
     * @access public
     */
    public function getDepartments()
    {
        return array("Dept. A", "Dept. B", "Dept. C");
    }

    /**
     * Get Instructors
     *
     * Obtain a list of instructors for use in limiting the reserves list.
     *
     * @return array An associative array with key = ID, value = name.
     * @access public
     */
    public function getInstructors()
    {
        return array("Instructor A", "Instructor B", "Instructor C");
    }

    /**
     * Get Courses
     *
     * Obtain a list of courses for use in limiting the reserves list.
     *
     * @return array An associative array with key = ID, value = name.
     * @access public
     */
    public function getCourses()
    {
        return array("Course A", "Course B", "Course C");
    }

    /**
     * Get New Items
     *
     * Retrieve the IDs of items recently added to the catalog.
     *
     * @param int $page    Page number of results to retrieve (counting starts at 1)
```

```
     * @param int $limit   The size of each page of results to retrieve
     * @param int $daysOld The maximum age of records to retrieve in days (max. 30)
     * @param int $fundId  optional fund ID to use for limiting results (use a value
     * returned by getFunds, or exclude for no limit); note that "fund" may be a
     * misnomer - if funds are not an appropriate way to limit your new item
     * results, you can return a different set of values from getFunds. The
     * important thing is that this parameter supports an ID returned by getFunds,
     * whatever that may mean.
     *
     * @return array       Associative array with 'count' and 'results' keys
     * @access public
     */
    public function getNewItems($page, $limit, $daysOld, $fundId = null)
    {
        // Do some initial work in solr so we aren't repeating it inside this loop.
        $this->_prepSolr();

        // Pick a random number of results to return -- don't exceed limit or 30,
        // whichever is smaller (this can be pretty slow due to the random ID code).
        $count = rand(0, $limit > 30 ? 30 : $limit);
        $results = array();
        for ($x = 0; $x < $count; $x++) {
            $randomId = $this->_getRandomBibId();

            // avoid duplicate entries in array:
            if (!in_array($randomId, $results)) {
                $results[] = $randomId;
            }
        }
        $retVal = array('count' => count($results), 'results' => array());
        foreach ($results as $result) {
            $retVal['results'][] = array('id' => $result);
        }
        return $retVal;
    }

    /**
     * Find Reserves
     *
     * Obtain information on course reserves.
     *
     * @param string $course ID from getCourses (empty string to match all)
     * @param string $inst   ID from getInstructors (empty string to match all)
     * @param string $dept   ID from getDepartments (empty string to match all)
     *
     * @return mixed An array of associative arrays representing reserve items
     * (or a PEAR_Error object if there is a problem)
     * @access public
     */
    public function findReserves($course, $inst, $dept)
    {
        // Do some initial work in solr so we aren't repeating it inside this loop.
        $this->_prepSolr();

        // Pick a random number of results to return -- don't exceed 30.
        $count = rand(0, 30);
        $results = array();
```

```
        for ($x = 0; $x < $count; $x++) {
            $randomId = $this->_getRandomBibId();

            // avoid duplicate entries in array:
            if (!in_array($randomId, $results)) {
                $results[] = $randomId;
            }
        }

        $retVal = array();
        foreach ($results as $current) {
            $retVal[] = array('BIB_ID' => $current);
        }
        return $retVal;
    }

    /**
     * Cancel Holds
     *
     * Attempts to Cancel a hold or recall on a particular item. The
     * data in $cancelDetails['details'] is determined by getCancelHoldDetails().
     *
     * @param array $cancelDetails An array of item and patron data
     *
     * @return array               An array of data on each request including
     * whether or not it was successful and a system message (if available)
     * @access public
     */
    public function cancelHolds($cancelDetails)
    {
        // Rewrite the holds in the session, removing those the user wants to
        // cancel.
        $newHolds = array();
        $retVal = array('count' => 0, 'items' => array());
        foreach ($_SESSION['demoData']['holds'] as $current) {
            if (!in_array($current['reqnum'], $cancelDetails['details'])) {
                $newHolds[] = $current;
            } else {
                // 50% chance of cancel failure for testing purposes
                if (rand() % 2) {
                    $retVal['count']++;
                    $retVal['items'][$current['item_id']] = array(
                        'success' => true,
                        'status' => 'hold_cancel_success'
                    );
                } else {
                    $newHolds[] = $current;
                    $retVal['items'][$current['item_id']] = array(
                        'success' => false,
                        'status' => 'hold_cancel_fail',
                        'sysMessage' =>
                            'Demonstrating failure; keep trying and ' .
                            'it will work eventually.'
                    );
                }
            }
        }
```

```
        $_SESSION['demoData']['holds'] = $newHolds;
        return $retVal;
    }

    /**
     * Get Cancel Hold Details
     *
     * In order to cancel a hold, Voyager requires the patron details an item ID
     * and a recall ID. This function returns the item id and recall id as a string
     * separated by a pipe, which is then submitted as form data in Hold.php. This
     * value is then extracted by the CancelHolds function.
     *
     * @param array $holdDetails An array of item data
     *
     * @return string Data for use in a form field
     * @access public
     */
    public function getCancelHoldDetails($holdDetails)
    {
        return $holdDetails['reqnum'];
    }

    /**
     * Renew My Items
     *
     * Function for attempting to renew a patron's items.  The data in
     * $renewDetails['details'] is determined by getRenewDetails().
     *
     * @param array $renewDetails An array of data required for renewing items
     * including the Patron ID and an array of renewal IDS
     *
     * @return array              An array of renewal information keyed by item ID
     * @access public
     */
    public function renewMyItems($renewDetails)
    {
        // Set up return value -- no blocks in demo driver currently.
        $finalResult = array('blocks' => array(), 'details' => array());

        foreach ($_SESSION['demoData']['transactions'] as $i => $current) {
            // Only renew requested items:
            if (in_array($current['item_id'], $renewDetails['details'])) {
                if (rand() % 2) {
                    $old = $_SESSION['demoData']['transactions'][$i]['duedate'];
                    $_SESSION['demoData']['transactions'][$i]['duedate']
                        = date("j-M-y", strtotime($old . " + 7 days"));

                    $finalResult['details'][$current['item_id']] = array(
                        "success" => true,
                        "new_date" =>
                            $_SESSION['demoData']['transactions'][$i]['duedate'],
                        "new_time" => '',
                        "item_id" => $current['item_id'],
                    );
                } else {
                    $finalResult['details'][$current['item_id']] = array(
```

```
                    "success" => false,
                    "new_date" => false,
                    "item_id" => $current['item_id'],
                    "sysMessage" =>
                        'Demonstrating failure; keep trying and ' .
                        'it will work eventually.'
                );
            }
        }
    }

    return $finalResult;
}

/**
 * Get Renew Details
 *
 * In order to renew an item, Voyager requires the patron details and an item
 * id. This function returns the item id as a string which is then used
 * as submitted form data in checkedOut.php. This value is then extracted by
 * the RenewMyItems function.
 *
 * @param array $checkOutDetails An array of item data
 *
 * @return string Data for use in a form field
 * @access public
 */
public function getRenewDetails($checkOutDetails)
{
    return $checkOutDetails['item_id'];
}

/**
 * Place Hold
 *
 * Attempts to place a hold or recall on a particular item and returns
 * an array with result details or a PEAR error on failure of support classes
 *
 * @param array $holdDetails An array of item and patron data
 *
 * @return mixed An array of data on the request including
 * whether or not it was successful and a system message (if available) or a
 * PEAR error on failure of support classes
 * @access public
 */
public function placeHold($holdDetails)
{
    // Simulate failure:
    if (rand() % 2) {
        return array(
            "success" => false,
            "sysMessage" =>
                'Demonstrating failure; keep trying and ' .
                'it will work eventually.'
        );
    }
```

```php
        if (!isset($_SESSION['demoData']['holds'])) {
            $_SESSION['demoData']['holds'] = array();
        }
        $lastHold = count($_SESSION['demoData']['holds']) - 1;
        $nextId = $lastHold >= 0
            ? $_SESSION['demoData']['holds'][$lastHold]['item_id'] + 1
            : 0;

        $_SESSION['demoData']['holds'][] = array(
            "id"       => $holdDetails['id'],
            "location" => $this->_getFakeLoc(),
            "expire"   => date("j-M-y", strtotime("now + 30 days")),
            "create"   =>
                date("j-M-y"),
            "reqnum"   => sprintf("%06d", $nextId),
            "item_id" => $nextId
        );

        return array('success' => true);
    }

    /**
     * Public Function which specifies renew, hold and cancel settings.
     *
     * @param string $function The name of the feature to be checked
     *
     * @return array An array with key-value pairs.
     * @access public
     */
    public function getConfig($function)
    {
        if ($function == 'Holds') {
            return array(
                'HMACKeys' => array('id'),
                'extraHoldFields' => 'comments'
            );
        }
        return array();
    }
}
?>
```