```php
<?php
/**
 * Index Record Driver
 *
 * PHP version 5
 *
 * Copyright (C) Villanova University 2010.
 *
 * 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  RecordDrivers
 * @author   Demian Katz <demian.katz@villanova.edu>
 * @license  http://opensource.org/licenses/gpl-2.0.php GNU General Public License
 * @link     http://vufind.org/wiki/other_than_marc Wiki
 */
require_once 'RecordDrivers/Interface.php';

require_once 'services/MyResearch/lib/User.php';
require_once 'services/MyResearch/lib/Resource.php';
require_once 'services/MyResearch/lib/Resource_tags.php';
require_once 'services/MyResearch/lib/Tags.php';

/**
 * Index Record Driver
 *
 * This class is designed to handle records in a generic fashion, using
 * fields from the index.  It is invoked when a record-format-specific
 * driver cannot be found.
 *
 * @category VuFind
 * @package  RecordDrivers
 * @author   Demian Katz <demian.katz@villanova.edu>
 * @license  http://opensource.org/licenses/gpl-2.0.php GNU General Public License
 * @link     http://vufind.org/wiki/other_than_marc Wiki
 */
class IndexRecord implements RecordInterface
{
    protected $fields;
    protected $index = false;

    /**
     * These Solr fields should be used for snippets if available (listed in order
     * of preference).
     *
     * @var    array
```

```
     * @access protected
     */
    protected $preferredSnippetFields = array(
        'contents', 'topic'
    );

    /**
     * These Solr fields should NEVER be used for snippets.  (We exclude author
     * and title because they are already covered by displayed fields; we exclude
     * spelling because it contains lots of fields jammed together and may cause
     * glitchy output; we exclude ID because random numbers are not helpful).
     *
     * @var    array
     * @access protected
     */
    protected $forbiddenSnippetFields = array(
        'author', 'author-letter', 'title', 'title_short', 'title_full',
        'title_full_unstemmed', 'title_auth', 'title_sub', 'spelling', 'id'
    );

    /**
     * These are captions corresponding with Solr fields for use when displaying
     * snippets.
     *
     * @var    array
     * @access protected
     */
    protected $snippetCaptions = array();

    /**
     * Should we highlight fields in search results?
     *
     * @var    bool
     * @access protected
     */
    protected $highlight = false;

    /**
     * Should we include snippets in search results?
     *
     * @var    bool
     * @access protected
     */
    protected $snippet = false;

    /**
     * Constructor.  We build the object using all the data retrieved
     * from the (Solr) index (which also happens to include the
     * 'fullrecord' field containing raw metadata).  Since we have to
     * make a search call to find out which record driver to construct,
     * we will already have this data available, so we might as well
     * just pass it into the constructor.
     *
     * @param array $indexFields All fields retrieved from the index.
     *
     * @access public
     */
```

```
    public function __construct($indexFields)
    {
        $this->fields = $indexFields;

        // Load highlighting/snippet preferences:
        $searchSettings = getExtraConfigArray('searches');
        $this->highlight = !isset($searchSettings['General']['highlighting'])
            ? false : $searchSettings['General']['highlighting'];
        $this->snippet = !isset($searchSettings['General']['snippets'])
            ? false : $searchSettings['General']['snippets'];
        $this->snippetCaptions = isset($searchSettings['Snippet_Captions'])
            && is_array($searchSettings['Snippet_Captions']) ?
            $searchSettings['Snippet_Captions'] : array();
    }

    /**
     * Get text that can be displayed to represent this record in
     * breadcrumbs.
     *
     * @return string Breadcrumb text to represent this record.
     * @access public
     */
    public function getBreadcrumb()
    {
        return $this->getShortTitle();
    }

    /**
     * Assign necessary Smarty variables and return a template name
     * to load in order to display the requested citation format.
     * For legal values, see getCitationFormats().  Returns null if
     * format is not supported.
     *
     * @param string $format Citation format to display.
     *
     * @return string        Name of Smarty template file to display.
     * @access public
     */
    public function getCitation($format)
    {
        // Build author list:
        $authors = array();
        $primary = $this->getPrimaryAuthor();
        if (!empty($primary)) {
            $authors[] = $primary;
        }
        $authors = array_unique(
            array_merge($authors, $this->getSecondaryAuthors())
        );

        // Collect all details for citation builder:
        $publishers = $this->getPublishers();
        $pubDates = $this->getPublicationDates();
        $pubPlaces = $this->getPlacesOfPublication();
        $details = array(
            'authors' => $authors,
            'title' => $this->getShortTitle(),
```

```
            'subtitle' => $this->getSubtitle(),
            'pubPlace' => count($pubPlaces) > 0 ? $pubPlaces[0] : null,
            'pubName' => count($publishers) > 0 ? $publishers[0] : null,
            'pubDate' => count($pubDates) > 0 ? $pubDates[0] : null,
            'edition' => array($this->getEdition())
        );

        // Build the citation:
        $citation = new CitationBuilder($details);
        if (in_array($format, $citation->getSupportedCitationFormats())) {
            return $citation->getCitation($format);
        } else {
            return '';
        }
    }

    /**
     * Get an array of strings representing citation formats supported
     * by this record's data (empty if none).
     *
     * @return array Strings representing citation formats.
     * @access public
     */
    public function getCitationFormats()
    {
        return CitationBuilder::getSupportedCitationFormats();
    }

    /**
     * Assign necessary Smarty variables and return a template name to
     * load in order to display core metadata (the details shown in the
     * top portion of the record view pages, above the tabs).
     *
     * @return string Name of Smarty template file to display.
     * @access public
     */
    public function getCoreMetadata()
    {

       global $configArray;
        global $interface;

        // Assign required variables (some of these are also used by templates for
        // tabs, since every tab can assume that the core data is already assigned):
        // $interface->assign('coreAltTitle', $this->getAltTitle());
        $this->assignTagList();
        $interface->assign('isbn', $this->getCleanISBN());  // needed for covers
        $interface->assign('recordFormat', $this->getFormats());
        $interface->assign('recordLanguage', $this->getLanguages());

        // These variables are only used by the core template, and they are prefixed
        // with "core" to prevent conflicts with other variable names.
        $interface->assign('coreShortTitle', $this->getShortTitle());
        $interface->assign('coreSubtitle', $this->getSubtitle());
        $interface->assign('coreTitleStatement', $this->getTitleStatement());
        $interface->assign('coreTitleSection', $this->getTitleSection());
        $interface->assign('coreNextTitles', $this->getNewerTitles());
```

```
        $interface->assign('corePrevTitles', $this->getPreviousTitles());
        $interface->assign('corePublications', $this->getPublicationDetails());
        $interface->assign('coreEdition', $this->getEdition());
        $interface->assign('coreSeries', $this->getSeries());
        $interface->assign('coreSubjects', $this->getAllSubjectHeadings());
        $interface->assign('coreRecordLinks', $this->getAllRecordLinks());
        $interface->assign('coreThumbMedium', $this->getThumbnail('medium'));
        $interface->assign('coreThumbLarge', $this->getThumbnail('large'));

        // Only display OpenURL link if the option is turned on and we have
        // an ISSN.  We may eventually want to make this rule more flexible,
        // but for now the ISSN restriction is designed to be consistent with
        // the way we display items on the search results list.
        $hasOpenURL = ($this->openURLActive('record') && $this->getCleanISSN());
        if ($hasOpenURL) {
            $interface->assign('coreOpenURL', $this->getOpenURL());
        }

        // Only load URLs if we have no OpenURL or we are configured to allow
        // URLs and OpenURLs to coexist:
        if (!isset($configArray['OpenURL']['replace_other_urls'])
            || !$configArray['OpenURL']['replace_other_urls'] || !$hasOpenURL
        ) {
            $interface->assign('coreURLs', $this->getURLs());
        }

        // The secondary author array may contain a corporate or primary author;
        // let's be sure we filter out duplicate values.
        $mainAuthor = $this->getPrimaryAuthor();
        $corpAuthor = $this->getCorporateAuthor();
        $secondaryAuthors = $this->getSecondaryAuthors();
        $duplicates = array();
        if (!empty($mainAuthor)) {
            $duplicates[] = $mainAuthor;
        }
        if (!empty($corpAuthor)) {
            $duplicates[] = $corpAuthor;
        }
        if (!empty($duplicates)) {
            $secondaryAuthors = array_diff($secondaryAuthors, $duplicates);
        }
        $interface->assign('coreMainAuthor', $mainAuthor);
        $interface->assign('coreCorporateAuthor', $corpAuthor);
        $interface->assign('coreContributors', $secondaryAuthors);

        // Assign only the first piece of summary data for the core; we'll get the
        // rest as part of the extended data.
        $summary = $this->getSummary();
        $summary = count($summary) > 0 ? $summary[0] : null;
        $interface->assign('coreSummary', $summary);

        // Send back the template name:
        return 'RecordDrivers/Index/core.tpl';
    }

    /**
     * Get an array of search results for other editions of the title
```

```
     * represented by this record (empty if unavailable).  In most cases,
     * this will use the XISSN/XISBN logic to find matches.
     *
     * @return mixed Editions in index engine result format (or null if no
     * hits, or PEAR_Error object).
     * @access public
     */
    public function getEditions()
    {
        include_once 'sys/WorldCatUtils.php';
        $wc = new WorldCatUtils();

        // Try to build an array of OCLC Number, ISBN or ISSN-based sub-queries:
        $parts = array();
        $oclcNum = $this->getCleanOCLCNum();
        if (!empty($oclcNum)) {
            $oclcList = $wc->getXOCLCNUM($oclcNum);
            foreach ($oclcList as $current) {
                $parts[] = "oclc_num:" . $current;
            }
        }
        $isbn = $this->getCleanISBN();
        if (!empty($isbn)) {
            $isbnList = $wc->getXISBN($isbn);
            foreach ($isbnList as $current) {
                $parts[] = 'isbn:' . $current;
            }
        }
        $issn = $this->getCleanISSN();
        if (!empty($issn)) {
            $issnList = $wc->getXISSN($issn);
            foreach ($issnList as $current) {
                $parts[] = 'issn:' . $current;
            }
        }

        // If we have query parts, we should try to find related records:
        if (!empty($parts)) {
            // Assemble the query parts and filter out current record:
            $query = '(' . implode(' OR ', $parts) . ') NOT id:' .
                $this->getUniqueID();

            // Perform the search and return either results or an error:
            $index = $this->getIndexEngine();
            $result = $index->search($query, null, null, 0, 5);
            if (PEAR::isError($result)) {
                return $result;
            }
            if (isset($result['response']['docs'])
                && !empty($result['response']['docs'])
            ) {
                return $result['response']['docs'];
            }
        }

        // If we got this far, we were unable to find any results:
        return null;
```

```
    }

    /**
     * Get the text to represent this record in the body of an email.
     *
     * @return string Text for inclusion in email.
     * @access public
     */
    public function getEmail()
    {
        return "   " . $this->getTitle() . "\n";
    }

    /**
     * Get any excerpts associated with this record.  For details of
     * the return format, see sys/Excerpts.php.
     *
     * @return array Excerpt information.
     * @access public
     */
    public function getExcerpts()
    {
        include_once 'sys/Excerpts.php';

        $ed = new ExternalExcerpts($this->getCleanISBN());
        return $ed->fetch();
    }

    /**
     * Assign necessary Smarty variables and return a template name to
     * load in order to export the record in the requested format.  For
     * legal values, see getExportFormats().  Returns null if format is
     * not supported.
     *
     * @param string $format Export format to display.
     *
     * @return string        Name of Smarty template file to display.
     * @access public
     */
    public function getExport($format)
    {
        // Not currently supported for index-based records:
        return null;
    }

    /**
     * Get an array of strings representing formats in which this record's
     * data may be exported (empty if none).  Legal values: "RefWorks",
     * "EndNote", "MARC", "RDF".
     *
     * @return array Strings representing export formats.
     * @access public
     */
    public function getExportFormats()
    {
        // No export formats currently supported for index-based records:
        return array();
```

```
    }

    /**
     * Assign necessary Smarty variables and return a template name to
     * load in order to display extended metadata (more details beyond
     * what is found in getCoreMetadata() -- used as the contents of the
     * Description tab of the record view).
     *
     * @return string Name of Smarty template file to display.
     * @access public
     */
    public function getExtendedMetadata()
    {
        global $interface;

        // Assign various values for display by the template; we'll prefix
        // everything with "extended" to avoid clashes with values assigned
        // elsewhere.
        $interface->assign('extendedDescription', $this->getDescription());
        $interface->assign('extendedSummary', $this->getSummary());
        $interface->assign('extendedAccess', $this->getAccessRestrictions());
        $interface->assign('extendedRelated', $this->getRelationshipNotes());
        $interface->assign('extendedNotes', $this->getGeneralNotes());
        $interface->assign('extendedDateSpan', $this->getDateSpan());
        $interface->assign('extendedISBNs', $this->getISBNs());     /* Zeile 424 ist für den
        Beschreibungsindex speziell für die ISBN */
        $interface->assign('extendedcallnumber', $this->getcallnumber());   /* Zeile 425 ist
        für den Beschreibungsindex speziell für die Signatur */
        $interface->assign('extendedISSNs', $this->getISSNs());
        $interface->assign('extendedPhysical', $this->getPhysicalDescriptions());
        $interface->assign('extendedFrequency', $this->getPublicationFrequency());
        $interface->assign('extendedPlayTime', $this->getPlayingTimes());
        $interface->assign('extendedSystem', $this->getSystemDetails());
        $interface->assign('extendedAudience', $this->getTargetAudienceNotes());
        $interface->assign('extendedAwards', $this->getAwards());
        $interface->assign('extendedCredits', $this->getProductionCredits());
        $interface->assign('extendedBibliography', $this->getBibliographyNotes());
        $interface->assign('extendedFindingAids', $this->getFindingAids());

        return 'RecordDrivers/Index/extended.tpl';
    }

    /**
     * Assign necessary Smarty variables and return a template name to
     * load in order to display holdings extracted from the base record
     * (i.e. URLs in MARC 856 fields) and, if necessary, the ILS driver.
     * Returns null if no data is available.
     *
     * @param array $patron An array of patron data
     *
     * @return string Name of Smarty template file to display.
     * @access public
     */
    public function getHoldings($patron = false)
    {
        global $interface;
        global $configArray;
```

Index und Signatur einfügen

```
        $interface->assign('extendedISBNs', $this->getISBNs());

        if ("driver" == CatalogConnection::getHoldsMode()) {
            $interface->assign('driverMode', true);
            if (!UserAccount::isLoggedIn()) {
                $interface->assign('showLoginMsg', true);
            }
        }

        // Only display OpenURL link if the option is turned on and we have
        // an ISSN.  We may eventually want to make this rule more flexible,
        // but for now the ISSN restriction is designed to be consistent with
        // the way we display items on the search results list.
        $hasOpenURL = ($this->openURLActive('holdings') && $this->getCleanISSN());
        if ($hasOpenURL) {
            $interface->assign('holdingsOpenURL', $this->getOpenURL());
        }

        // Display regular URLs unless OpenURL is present and configured to
        // replace them:
        if (!isset($configArray['OpenURL']['replace_other_urls'])
            || !$configArray['OpenURL']['replace_other_urls'] || !$hasOpenURL
        ) {
            $interface->assign('holdingURLs', $this->getURLs());
        }
        $interface->assign('holdingLCCN', $this->getLCCN());
        $interface->assign('holdingArrOCLC', $this->getOCLC());

        // Load real-time data if available:
        $interface->assign('holdings', $this->getRealTimeHoldings($patron));
        $interface->assign('history', $this->getRealTimeHistory());

        return 'RecordDrivers/Index/holdings.tpl';
    }

    /**
     * Assign necessary Smarty variables and return a template name to
     * load in order to display a summary of the item suitable for use in
     * user's favorites list.
     *
     * @param object $user      User object owning tag/note metadata.
     * @param int    $listId    ID of list containing desired tags/notes (or null
     * to show tags/notes from all user's lists).
     * @param bool   $allowEdit Should we display edit controls?
     *
     * @return string           Name of Smarty template file to display.
     * @access public
     */
    public function getListEntry($user, $listId = null, $allowEdit = true)
    {
        global $interface;

        // Extract bibliographic metadata from the record:
        $id = $this->getUniqueID();
        $interface->assign('listId', $id);
        $interface->assign('listFormats', $this->getFormats());
        $interface->assign('listTitle', $this->getTitle());
```

```
        $interface->assign('listAuthor', $this->getPrimaryAuthor());
        $interface->assign('listThumb', $this->getThumbnail());

        // Extract user metadata from the database:
        $notes = array();
        $data = $user->getSavedData($id, $listId);
        foreach ($data as $current) {
            if (!empty($current->notes)) {
                $notes[] = $current->notes;
            }
        }
        $interface->assign('listNotes', $notes);
        $interface->assign('listTags', $user->getTags($id, $listId));

        // Pass some parameters along to the template to influence edit controls:
        $interface->assign('listSelected', $listId);
        $interface->assign('listEditAllowed', $allowEdit);

        return 'RecordDrivers/Index/listentry.tpl';
    }

    /**
     * Get the OpenURL parameters to represent this record (useful for the
     * title attribute of a COinS span tag).
     *
     * @return string OpenURL parameters.
     * @access public
     */
    public function getOpenURL()
    {
        // Get the COinS ID -- it should be in the OpenURL section of config.ini,
        // but we'll also check the COinS section for compatibility with legacy
        // configurations (this moved between the RC2 and 1.0 releases).
        global $configArray;
        $coinsID = isset($configArray['OpenURL']['rfr_id']) ?
            $configArray['OpenURL']['rfr_id'] :
            $configArray['COinS']['identifier'];
        if (empty($coinsID)) {
            $coinsID = 'vufind.svn.sourceforge.net';
        }

        // Get a representative publication date:
        $pubDate = $this->getPublicationDates();
        $pubDate = empty($pubDate) ? '' : $pubDate[0];

        // Start an array of OpenURL parameters:
        $params = array(
            'ctx_ver' => 'Z39.88-2004',
            'ctx_enc' => 'info:ofi/enc:UTF-8',
            'rfr_id' => "info:sid/{$coinsID}:generator",
            'rft.title' => $this->getTitle(),
            'rft.date' => $pubDate
        );

        // Add additional parameters based on the format of the record:
        $formats = $this->getFormats();
```

```
        // If we have multiple formats, Book and Journal are most important...
        if (in_array('Book', $formats)) {
            $format = 'Book';
        } else if (in_array('Journal', $formats)) {
            $format = 'Journal';
        } else {
            $format = $formats[0];
        }
        switch($format) {
        case 'Book':
            $params['rft_val_fmt'] = 'info:ofi/fmt:kev:mtx:book';
            $params['rft.genre'] = 'book';
            $params['rft.btitle'] = $params['rft.title'];
            $series = $this->getSeries();
            if (count($series) > 0) {
                // Handle both possible return formats of getSeries:
                $params['rft.series'] = is_array($series[0]) ?
                    $series[0]['name'] : $series[0];
            }
            $params['rft.au'] = $this->getPrimaryAuthor();
            $publishers = $this->getPublishers();
            if (count($publishers) > 0) {
                $params['rft.pub'] = $publishers[0];
            }
            $params['rft.edition'] = $this->getEdition();
            $params['rft.isbn'] = $this->getCleanISBN();
            break;
        case 'Journal':
            /* This is probably the most technically correct way to represent
             * a journal run as an OpenURL; however, it doesn't work well with
             * Zotero, so it is currently commented out -- instead, we just add
             * some extra fields and then drop through to the default case.
            $params['rft_val_fmt'] = 'info:ofi/fmt:kev:mtx:journal';
            $params['rft.genre'] = 'journal';
            $params['rft.jtitle'] = $params['rft.title'];
            $params['rft.issn'] = $this->getCleanISSN();
            $params['rft.au'] = $this->getPrimaryAuthor();
            break;
             */
            $params['rft.issn'] = $this->getCleanISSN();

            // Including a date in a title-level Journal OpenURL may be too
            // limiting -- in some link resolvers, it may cause the exclusion
            // of databases if they do not cover the exact date provided!
            unset($params['rft.date']);

            // If we're working with the SFX resolver, we should add a
            // special parameter to ensure that electronic holdings links
            // are shown even though no specific date or issue is specified:
            if (isset($configArray['OpenURL']['resolver'])
                && strtolower($configArray['OpenURL']['resolver']) == 'sfx'
            ) {
                $params['sfx.ignore_date_threshold'] = 1;
            }
        default:
            $params['rft_val_fmt'] = 'info:ofi/fmt:kev:mtx:dc';
            $params['rft.creator'] = $this->getPrimaryAuthor();
```

```
            $publishers = $this->getPublishers();
            if (count($publishers) > 0) {
                $params['rft.pub'] = $publishers[0];
            }
            $params['rft.format'] = $format;
            $langs = $this->getLanguages();
            if (count($langs) > 0) {
                $params['rft.language'] = $langs[0];
            }
            break;
        }

        // Assemble the URL:
        $parts = array();
        foreach ($params as $key => $value) {
            $parts[] = $key . '=' . urlencode($value);
        }
        return implode('&', $parts);
    }

    /**
     * Get an XML RDF representation of the data in this record.
     *
     * @return mixed XML RDF data (false if unsupported or error).
     * @access public
     */
    public function getRDFXML()
    {
        // Not supported.
        return false;
    }

    /**
     * Get any reviews associated with this record.  For details of
     * the return format, see sys/Reviews.php.
     *
     * @return array Review information.
     * @access public
     */
    public function getReviews()
    {
        include_once 'sys/Reviews.php';

        $rev = new ExternalReviews($this->getCleanISBN());
        return $rev->fetch();
    }

    /**
     * Assign necessary Smarty variables and return a template name for the current
     * view to load in order to display a summary of the item suitable for use in
     * search results.
     *
     * @param string $view The current view.
     *
     * @return string      Name of Smarty template file to display.
     * @access public
     */
```

```
    public function getSearchResult($view = 'list')
    {
        global $configArray;
        global $interface;

        /** $interface->assign('summAltTitle', $this->getAltTitle()); */
        $interface->assign('summId', $this->getUniqueID());
        $interface->assign('summFormats', $this->getFormats());
        $interface->assign('summHighlightedTitle', $this->getHighlightedTitle());
        $interface->assign('summTitle', $this->getTitle());
        $interface->assign('summHighlightedAuthor', $this->getHighlightedAuthor());
        $interface->assign('summAuthor', $this->getPrimaryAuthor());
        $interface->assign('summDate', $this->getPublicationDates());
        $interface->assign('summISBN', $this->getCleanISBN());
        $interface->assign('summThumb', $this->getThumbnail());
        $interface->assign('summThumbLarge', $this->getThumbnail('large'));
        $issn = $this->getCleanISSN();
        $interface->assign('summISSN', $issn);
        $interface->assign('summLCCN', $this->getLCCN());
        $interface->assign('summOCLC', $this->getOCLC());
        $interface->assign('summCallNo', $this->getCallNumber());

        // Obtain and assign snippet information:
        $snippet = $this->getHighlightedSnippet();
        $interface->assign(
            'summSnippetCaption', $snippet ? $snippet['caption'] : false
        );
        $interface->assign('summSnippet', $snippet ? $snippet['snippet'] : false);

        // Only display OpenURL link if the option is turned on and we have
        // an ISSN.  We may eventually want to make this rule more flexible,
        // but for now the ISSN restriction is designed to be consistent with
        // the way we display items on the search results list.
        $hasOpenURL = ($this->openURLActive('results') && $issn);
        $openURL = $this->getOpenURL();
        $interface->assign('summOpenUrl', $hasOpenURL ? $openURL : false);

        // Always provide an OpenURL for COinS purposes:
        $interface->assign('summCOinS', $openURL);

        // Display regular URLs unless OpenURL is present and configured to
        // replace them:
        if (!isset($configArray['OpenURL']['replace_other_urls'])
            || !$configArray['OpenURL']['replace_other_urls'] || !$hasOpenURL
        ) {
            $interface->assign('summURLs', $this->getURLs());
        } else {
            $interface->assign('summURLs', array());
        }

        // By default, do not display AJAX status; we won't assume that all
        // records exist in the ILS.  Child classes can override this setting
        // to turn on AJAX as needed:
        $interface->assign('summAjaxStatus', false);

        // Send back the template to display:
        return 'RecordDrivers/Index/result-' . $view . '.tpl';
```

```
    }

    /**
     * Assign necessary Smarty variables and return a template name to
     * load in order to display the full record information on the Staff
     * View tab of the record view page.
     *
     * @return string Name of Smarty template file to display.
     * @access public
     */
    public function getStaffView()
    {
        global $interface;
        $interface->assign('details', $this->fields);
        return 'RecordDrivers/Index/staff.tpl';
    }

    /**
     * Assign necessary Smarty variables and return a template name to
     * load in order to display the Table of Contents extracted from the
     * record.  Returns null if no Table of Contents is available.
     *
     * @return string Name of Smarty template file to display.
     * @access public
     */
    public function getTOC()
    {
        global $interface;

        if (!$this->hasTOC()) {
            return null;
        }

        $interface->assign('toc', $this->fields['contents']);
        return 'RecordDrivers/Index/toc.tpl';
    }

    /**
     * Return the unique identifier of this record within the Solr index;
     * useful for retrieving additional information (like tags and user
     * comments) from the external MySQL database.
     *
     * @return string Unique identifier.
     * @access public
     */
    public function getUniqueID()
    {
        return $this->fields['id'];
    }

    /**
     * Return an XML representation of the record using the specified format.
     * Return false if the format is unsupported.
     *
     * @param string $format Name of format to use (corresponds with OAI-PMH
     * metadataPrefix parameter).
     *
```

```
     * @return mixed         XML, or false if format unsupported.
     * @access public
     */
    public function getXML($format)
    {
        global $interface;

        // For OAI-PMH Dublin Core, use the core metadata fields with a different
        // template to produce the necessary XML:
        if ($format == 'oai_dc') {
            $this->getCoreMetadata();
            $interface->assign('oaiFullTitle', $this->getTitle());
            $interface->assign('oaiPublishers', $this->getPublishers());
            $interface->assign('oaiPubDates', $this->getPublicationDates());
            return $interface->fetch('RecordDrivers/Index/oai_dc.tpl');
        }

        // Unsupported format:
        return false;
    }

    /**
     * Does this record have audio content available?
     *
     * @return bool
     * @access public
     */
    public function hasAudio()
    {
        /* Audio is not supported yet.
         */
        return false;
    }

    /**
     * Does this record have an excerpt available?
     *
     * @return bool
     * @access public
     */
    public function hasExcerpt()
    {
        // If we have ISBN(s), we might have reviews:
        $isbns = $this->getISBNs();
        return !empty($isbns);
    }

    /**
     * Does this record have searchable full text in the index?
     *
     * Note: As of this writing, searchable full text is not a VuFind feature,
     *       but this method will be useful if/when it is eventually added.
     *
     * @return bool
     * @access public
     */
    public function hasFullText()
```

```
    {
        /* Full text is not supported yet.
         */
        return false;
    }

    /**
     * Does this record have image content available?
     *
     * @return bool
     * @access public
     */
    public function hasImages()
    {
        // Images are not supported yet.
        return false;
    }

    /**
     * Does this record support an RDF representation?
     *
     * @return bool
     * @access public
     */
    public function hasRDF()
    {
        // No RDF for Solr-based entries yet.
        return false;
    }

    /**
     * Does this record have reviews available?
     *
     * @return bool
     * @access public
     */
    public function hasReviews()
    {
        // If we have ISBN(s), we might have reviews:
        $isbns = $this->getISBNs();
        return !empty($isbns);
    }

    /**
     * Does this record have a Table of Contents available?
     *
     * @return bool
     * @access public
     */
    public function hasTOC()
    {
        // Do we have a table of contents stored in the index?
        return (isset($this->fields['contents']) &&
            count($this->fields['contents']) > 0);
    }

    /**
```

```
     * Does this record have video content available?
     *
     * @return bool
     * @access public
     */
    public function hasVideo()
    {
        /* Video is not supported yet.
         */
        return false;
    }

    /**
     * Assign a tag list to the interface based on the current unique ID.
     *
     * @return void
     * @access protected
     */

     protected function getAltTitel()
{
return isset($this->fields['title_alt']) ?
$this->fields['title_alt'] : '';
}

    protected function assignTagList()
    {
        global $interface;

        // Retrieve tags associated with the record
        $resource = new Resource();
        $resource->record_id = $this->getUniqueID();
        $resource->source = 'VuFind';
        $tags = $resource->getTags();
        $interface->assign('tagList', is_array($tags) ? $tags : array());
    }

    /**
     * Get access restriction notes for the record.
     *
     * @return array
     * @access protected
     */
    protected function getAccessRestrictions()
    {
        // Not currently stored in the Solr index
        return array();
    }

    /**
     * Get all subject headings associated with this record.  Each heading is
     * returned as an array of chunks, increasing from least specific to most
     * specific.
     *
     * @return array
     * @access protected
     */
```

```
    protected function getAllSubjectHeadings()
    {
        $topic = isset($this->fields['topic']) ? $this->fields['topic'] : array();
        $geo = isset($this->fields['geographic']) ?
            $this->fields['geographic'] : array();
        $genre = isset($this->fields['genre']) ? $this->fields['genre'] : array();

        // The Solr index doesn't currently store subject headings in a broken-down
        // format, so we'll just send each value as a single chunk.  Other record
        // drivers (i.e. MARC) can offer this data in a more granular format.
        $retval = array();
        foreach ($topic as $t) {
            $retval[] = array($t);
        }
        foreach ($geo as $g) {
            $retval[] = array($g);
        }
        foreach ($genre as $g) {
            $retval[] = array($g);
        }

        return $retval;
    }

    /**
     * Get all record links related to the current record. Each link is returned as
     * array.
     * NB: to use this method you must override it.
     * Format:
     * <code>
     * array(
     *        array(
     *               'title' => label_for_title
     *               'value' => link_name
     *               'link'  => link_URI
     *        ),
     *        ...
     * )
     * </code>
     *
     * @return null|array
     * @access protected
     */
    protected function getAllRecordLinks()
    {
        return null;
    }

    /**
     * Get award notes for the record.
     *
     * @return array
     * @access protected
     */
    protected function getAwards()
    {
        // Not currently stored in the Solr index
```

```
        return array();
    }

    /**
     * Get notes on bibliography content.
     *
     * @return array
     * @access protected
     */
    protected function getBibliographyNotes()
    {
        // Not currently stored in the Solr index
        return array();
    }

    /**
     * Get the call number associated with the record (empty string if none).
     *
     * @return string
     * @access protected
     */
    protected function getCallNumber()
    {
        // Use the callnumber-a field from the Solr index; the plain callnumber
        // field is normalized to have no spaces, so it is unsuitable for display.
        return isset($this->fields['callnumber-a']) ?
            $this->fields['callnumber-a'] : '';
    }

    /**
     * Return the first valid ISBN found in the record (favoring ISBN-10 over
     * ISBN-13 when possible).
     *
     * @return mixed
     * @access protected
     */
    protected function getCleanISBN()
    {
        include_once 'sys/ISBN.php';

        // Get all the ISBNs and initialize the return value:
        $isbns = $this->getISBNs();
        $isbn13 = false;

        // Loop through the ISBNs:
        foreach ($isbns as $isbn) {
            // Strip off any unwanted notes:
            if ($pos = strpos($isbn, ' ')) {
                $isbn = substr($isbn, 0, $pos);
            }

            // If we find an ISBN-10, return it immediately; otherwise, if we find
            // an ISBN-13, save it if it is the first one encountered.
            $isbnObj = new ISBN($isbn);
            if ($isbn10 = $isbnObj->get10()) {
                return $isbn10;
            }
```

```
            if (!$isbn13) {
                $isbn13 = $isbnObj->get13();
            }
        }
        return $isbn13;
    }

    /**
     * Get just the base portion of the first listed ISSN (or false if no ISSNs).
     *
     * @return mixed
     * @access protected
     */
    protected function getCleanISSN()
    {
        $issns = $this->getISSNs();
        if (empty($issns)) {
            return false;
        }
        $issn = $issns[0];
        if ($pos = strpos($issn, ' ')) {
            $issn = substr($issn, 0, $pos);
        }
        return $issn;
    }

    /**
     * Get just the first listed OCLC Number (or false if none available).
     *
     * @return mixed
     * @access protected
     */
    protected function getCleanOCLCNum()
    {
        $nums = $this->getOCLC();
        return empty($nums) ? false : $nums[0];
    }

    /**
     * Get the main corporate author (if any) for the record.
     *
     * @return string
     * @access protected
     */
    protected function getCorporateAuthor()
    {
        // Not currently stored in the Solr index
        return null;
    }

    /**
     * Get the date coverage for a record which spans a period of time (i.e. a
     * journal).  Use getPublicationDates for publication dates of particular
     * monographic items.
     *
     * @return array
     * @access protected
```

```
     */
    protected function getDateSpan()
    {
        return isset($this->fields['dateSpan']) ?
            $this->fields['dateSpan'] : array();
    }

    /**
     * Get the edition of the current record.
     *
     * @return string
     * @access protected
     */
    protected function getEdition()
    {
        return isset($this->fields['edition']) ?
            $this->fields['edition'] : '';
    }

    /**
     * Get notes on finding aids related to the record.
     *
     * @return array
     * @access protected
     */
    protected function getFindingAids()
    {
        // Not currently stored in the Solr index
        return array();
    }

    /**
     * Get an array of all the formats associated with the record.
     *
     * @return array
     * @access protected
     */
    protected function getFormats()
    {
        return isset($this->fields['format']) ? $this->fields['format'] : array();
    }

    /**
     * Get general notes on the record.
     *
     * @return array
     * @access protected
     */
    protected function getGeneralNotes()
    {
        // Not currently stored in the Solr index
        return array();
    }

    /**
     * Get a highlighted author string, if available.
     *
```

```
     * @return string
     * @access protected
     */
    protected function getHighlightedAuthor()
    {
        // Don't check for highlighted values if highlighting is disabled:
        if (!$this->highlight) {
            return '';
        }
        return (isset($this->fields['_highlighting']['author'][0]))
            ? $this->fields['_highlighting']['author'][0] : '';
    }

    /**
     * Given a Solr field name, return an appropriate caption.
     *
     * @param string $field Solr field name
     *
     * @return mixed        Caption if found, false if none available.
     * @access protected
     */
    protected function getSnippetCaption($field)
    {
        return isset($this->snippetCaptions[$field])
            ? $this->snippetCaptions[$field] : false;
    }

    /**
     * Pick one line from the highlighted text (if any) to use as a snippet.
     *
     * @return mixed False if no snippet found, otherwise associative array
     * with 'snippet' and 'caption' keys.
     * @access protected
     */
    protected function getHighlightedSnippet()
    {
        // Only process snippets if the setting is enabled:
        if ($this->snippet) {
            // First check for preferred fields:
            foreach ($this->preferredSnippetFields as $current) {
                if (isset($this->fields['_highlighting'][$current][0])) {
                    return array(
                        'snippet' => $this->fields['_highlighting'][$current][0],
                        'caption' => $this->getSnippetCaption($current)
                    );
                }
            }

            // No preferred field found, so try for a non-forbidden field:
            if (is_array($this->fields['_highlighting'])) {
                foreach ($this->fields['_highlighting'] as $key => $value) {
                    if (!in_array($key, $this->forbiddenSnippetFields)) {
                        return array(
                            'snippet' => $value[0],
                            'caption' => $this->getSnippetCaption($key)
                        );
                    }
```

```
                }
            }
        }

        // If we got this far, no snippet was found:
        return false;
    }

    /**
     * Get a highlighted title string, if available.
     *
     * @return string
     * @access protected
     */
    protected function getHighlightedTitle()
    {
        // Don't check for highlighted values if highlighting is disabled:
        if (!$this->highlight) {
            return '';
        }
        return (isset($this->fields['_highlighting']['title'][0]))
            ? $this->fields['_highlighting']['title'][0] : '';
    }

    /**
     * Get the index engine to do a follow-up query.
     *
     * @return object
     * @access protected
     */
    protected function getIndexEngine()
    {
        // Build the index engine if we don't already have one:
        if (!$this->index) {
            $this->index = ConnectionManager::connectToIndex();
        }

        return $this->index;
    }

    /**
     * Get an array of all ISBNs associated with the record (may be empty).
     *
     * @return array
     * @access protected
     */
    protected function getISBNs()
    {
        // If ISBN is in the index, it should automatically be an array... but if
        // it's not set at all, we should normalize the value to an empty array.
        return isset($this->fields['isbn']) && is_array($this->fields['isbn']) ?
            $this->fields['isbn'] : array();
    }

    /**
     * Get an array of all ISSNs associated with the record (may be empty).
     *
```

```
     * @return array
     * @access protected
     */
    protected function getISSNs()
    {
        // If ISSN is in the index, it should automatically be an array... but if
        // it's not set at all, we should normalize the value to an empty array.
        return isset($this->fields['issn']) && is_array($this->fields['issn']) ?
            $this->fields['issn'] : array();
    }

    /**
     * Get an array of all the languages associated with the record.
     *
     * @return array
     * @access protected
     */
    protected function getLanguages()
    {
        return isset($this->fields['language']) ?
            $this->fields['language'] : array();
    }

    /**
     * Get an array of newer titles for the record.
     *
     * @return array
     * @access protected
     */
    protected function getNewerTitles()
    {
        return isset($this->fields['title_new']) ?
            $this->fields['title_new'] : array();
    }

    /**
     * Get an array of physical descriptions of the item.
     *
     * @return array
     * @access protected
     */
    protected function getPhysicalDescriptions()
    {
        return isset($this->fields['physical']) ?
            $this->fields['physical'] : array();
    }

    /**
     * Get the item's place of publication.
     *
     * @return array
     * @access protected
     */
    protected function getPlacesOfPublication()
    {
        // Not currently stored in the Solr index
        return array();
```

```
    }

    /**
     * Get an array of playing times for the record (if applicable).
     *
     * @return array
     * @access protected
     */
    protected function getPlayingTimes()
    {
        // Not currently stored in the Solr index
        return array();
    }

    /**
     * Get an array of previous titles for the record.
     *
     * @return array
     * @access protected
     */
    protected function getPreviousTitles()
    {
        return isset($this->fields['title_old']) ?
            $this->fields['title_old'] : array();
    }

    /**
     * Get the main author of the record.
     *
     * @return string
     * @access protected
     */
    protected function getPrimaryAuthor()
    {
        return isset($this->fields['author']) ?
            $this->fields['author'] : '';
    }

    /**
     * Get credits of people involved in production of the item.
     *
     * @return array
     * @access protected
     */
    protected function getProductionCredits()
    {
        // Not currently stored in the Solr index
        return array();
    }

    /**
     * Get the publication dates of the record.  See also getDateSpan().
     *
     * @return array
     * @access protected
     */
    protected function getPublicationDates()
```

```
    {
        return isset($this->fields['publishDate']) ?
            $this->fields['publishDate'] : array();
    }

    /**
     * Get an array of publication detail lines combining information from
     * getPublicationDates(), getPublishers() and getPlacesOfPublication().
     *
     * @return array
     * @access protected
     */
    protected function getPublicationDetails()
    {
        $places = $this->getPlacesOfPublication();
        $names = $this->getPublishers();
        $dates = $this->getPublicationDates();

        $i = 0;
        $retval = array();
        while (isset($places[$i]) || isset($names[$i]) || isset($dates[$i])) {
            // Put all the pieces together, and do a little processing to clean up
            // unwanted whitespace.
            $retval[] = trim(
                str_replace(
                    '  ', ' ',
                    ((isset($places[$i]) ? $places[$i] . ' ' : '') .
                    (isset($names[$i]) ? $names[$i] . ' ' : '') .
                    (isset($dates[$i]) ? $dates[$i] : ''))
                )
            );
            $i++;
        }

        return $retval;
    }

    /**
     * Get an array of publication frequency information.
     *
     * @return array
     * @access protected
     */
    protected function getPublicationFrequency()
    {
        // Not currently stored in the Solr index
        return array();
    }

    /**
     * Get the publishers of the record.
     *
     * @return array
     * @access protected
     */
    protected function getPublishers()
    {
```

```
        return isset($this->fields['publisher']) ?
            $this->fields['publisher'] : array();
    }

    /**
     * Get an array of information about record history, obtained in real-time
     * from the ILS.
     *
     * @return array
     * @access protected
     */
    protected function getRealTimeHistory()
    {
        // Not supported by the Solr index -- implement in child classes.
        return array();
    }

    /**
     * Get an array of information about record holdings, obtained in real-time
     * from the ILS.
     *
     * @param array $patron An array for patron information
     *
     * @return array
     * @access protected
     */
    protected function getRealTimeHoldings($patron = false)
    {
        // Not supported by the Solr index -- implement in child classes.
        return array();
    }

    /**
     * Get an array of strings describing relationships to other items.
     *
     * @return array
     * @access protected
     */
    protected function getRelationshipNotes()
    {
        // Not currently stored in the Solr index
        return array();
    }

    /**
     * Get an array of all secondary authors (complementing getPrimaryAuthor()).
     *
     * @return array
     * @access protected
     */
    protected function getSecondaryAuthors()
    {
        return isset($this->fields['author2']) ?
            $this->fields['author2'] : array();
    }

    /**
```

```
     * Get an array of all series names containing the record.  Array entries may
     * be either the name string, or an associative array with 'name' and 'number'
     * keys.
     *
     * @return array
     * @access protected
     */
    protected function getSeries()
    {
        // Only use the contents of the series2 field if the series field is empty
        if (isset($this->fields['series']) && !empty($this->fields['series'])) {
            return $this->fields['series'];
        }
        return isset($this->fields['series2']) ?
            $this->fields['series2'] : array();
    }

    /**
     * Get the short (pre-subtitle) title of the record.
     *
     * @return string
     * @access protected
     */
    protected function getShortTitle()
    {
        return isset($this->fields['title_short']) ?
            $this->fields['title_short'] : '';
    }

    /**
     * Get the description of the record.
     *
     * @return string
     * @access protected
     */
    protected function getDescription()
    {
        return isset($this->fields['description']) ?
            $this->fields['description'] : '';
    }

    /**
     * Get the subtitle of the record.
     *
     * @return string
     * @access protected
     */
    protected function getSubtitle()
    {
        return isset($this->fields['title_sub']) ?
            $this->fields['title_sub'] : '';
    }

    /**
     * Get an array of technical details on the item represented by the record.
     *
     * @return array
```

```
     * @access protected
     */
    protected function getSystemDetails()
    {
        // Not currently stored in the Solr index
        return array();
    }

    /**
     * Get an array of summary strings for the record.
     *
     * @return array
     * @access protected
     */
    protected function getSummary()
    {
        // Not currently stored in the Solr index
        return array();
    }

    /**
     * Get an array of note about the record's target audience.
     *
     * @return array
     * @access protected
     */
    protected function getTargetAudienceNotes()
    {
        // Not currently stored in the Solr index
        return array();
    }

    /**
     * Get the full title of the record.
     *
     * @return string
     * @access protected
     */
    protected function getTitle()
    {
        return isset($this->fields['title']) ?
            $this->fields['title'] : '';
    }

    /**
     * Get the text of the part/section portion of the title.
     *
     * @return string
     * @access protected
     */
    protected function getTitleSection()
    {
        // Not currently stored in the Solr index
        return null;
    }

    /**
```

```
     * Get the statement of responsibility that goes with the title (i.e. "by John
     * Smith").
     *
     * @return string
     * @access protected
     */
    protected function getTitleStatement()
    {
        // Not currently stored in the Solr index
        return null;
    }

    /**
     * Return an associative array of URLs associated with this record (key = URL,
     * value = description).
     *
     * @return array
     * @access protected
     */
    protected function getURLs()
    {
        $urls = array();
        if (isset($this->fields['url']) && is_array($this->fields['url'])) {
            foreach ($this->fields['url'] as $url) {
                // The index doesn't contain descriptions for URLs, so we'll just
                // use the URL itself as the description.
                $urls[$url] = $url;
            }
        }
        return $urls;
    }

    /**
     * Does the OpenURL configuration indicate that we should display OpenURLs in
     * the specified context?
     *
     * @param string $area 'results', 'record' or 'holdings'
     *
     * @return bool
     * @access protected
     */
    protected function openURLActive($area)
    {
        global $configArray;

        // Doesn't matter the target area if no OpenURL resolver is specified:
        if (!isset($configArray['OpenURL']['url'])) {
            return false;
        }

        // If a setting exists, return that:
        if (isset($configArray['OpenURL']['show_in_' . $area])) {
            return $configArray['OpenURL']['show_in_' . $area];
        }

        // If we got this far, use the defaults -- true for results, false for
        // everywhere else.
```

```
        return ($area == 'results');
    }

    /**
     * Get a LCCN, normalised according to info:lccn
     *
     * @return string
     * @access protected
     */
    protected function getLCCN()
    {
        // Get LCCN from Index
        $raw = isset($this->fields['lccn']) ? $this->fields['lccn'] : '';

        // Remove all blanks.
        $raw = preg_replace('{[ \t]+}', '', $raw);

        // If there is a forward slash (/) in the string, remove it, and remove all
        // characters to the right of the forward slash.
        if (strpos($raw, '/') > 0) {
            $tmpArray = explode("/", $raw);
            $raw = $tmpArray[0];
        }
        /* If there is a hyphen in the string:
            a. Remove it.
            b. Inspect the substring following (to the right of) the (removed)
               hyphen. Then (and assuming that steps 1 and 2 have been carried out):
                    i.  All these characters should be digits, and there should be
                    six or less.
                    ii. If the length of the substring is less than 6, left-fill the
                    substring with zeros until  the length is six.
        */
        if (strpos($raw, '-') > 0) {
            // haven't checked for i. above. If they aren't all digits, there is
            // nothing that can be done, so might as well leave it.
            $tmpArray = explode("-", $raw);
            $raw = $tmpArray[0] . str_pad($tmpArray[1], 6, "0", STR_PAD_LEFT);
        }
        return $raw;
    }

    /**
     * Get the OCLC number of the record.
     *
     * @return array
     * @access protected
     */
    protected function getOCLC()
    {
        return isset($this->fields['oclc_num']) ?
            $this->fields['oclc_num'] : array();
    }

    /**
     * Return a URL to a thumbnail preview of the record, if available; false
     * otherwise.
     *
```

```
     * @param array $size Size of thumbnail (small, medium or large -- small is
     * default).
     *
     * @return mixed
     * @access protected
     */
    protected function getThumbnail($size = 'small')
    {
        global $configArray;

        if ($isbn = $this->getCleanISBN()) {
            return $configArray['Site']['url'] . '/bookcover.php?isn=' .
                urlencode($isbn) . '&size=' . urlencode($size);
        }
        return false;
    }
}

?>
```