IMu Documentation

IMu OAI-PMH Web Service

Document Version 1.1

EMu Version 4.0
IMu Version 1.0.03

www.kesoftware.com

Contents

SECTION 1

OAI-PMH Concepts

What is the OAI-PMH?

The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) specifies a relatively simple method for harvesting data from a data source. The data source, called a repository in OAI-PMH terms, accepts requests that conform to the protocol. The requests can query the repository for information about itself and the service it provides as well as the actual data, called metadata in OAI-PMH terms, that the repository exposes. Responses are returned in the Extensible Markup Language (XML) format, the structure of which is also specified by the protocol. The structure of any metadata included with a response is not specified by the OAI-PMH protocol and a repository may support any number of XML formats for metadata. However, the protocol requires that, at a minimum, all repositories disseminate metadata in the unqualified Dublin Core (oai_dc) XML format. The format of metadata included in a response can be specified in the request.

Requests are made to a repository using the `HTTP GET` or `POST` methods. There is a single base URL for all requests. All requests consist of a list of keyword arguments which take the form of `key=value` pairs. One of the `key=value` pairs must be the verb key paired with one of the six allowed verb values:

- `GetRecord`
- `Identify`
- `ListIdentifiers`
- `ListMetadataFormats`
- `ListRecords`
- `ListSets`

Additional mandatory and optional arguments are specified depending on the particular verb used.

Requests

Requests must include one and only one of the six verbs specified by the OAI-PMH:

Verb	Description
`GetRecord`	Retrieve an individual metadata record from the repository. For example: http://arXiv.org/oai2?verb=GetRecord&identifier=oai:arXiv.org:cs/0112017&metadataPrefix=oai_dc
`Identify`	Retrieve information about a repository. For example: http://arXiv.org/oai2?verb=Identify
`ListIdentifiers`	Retrieve a list of identifiers that uniquely identifies each metadata record in the repository. For example: http://arXiv.org/oai2?verb=ListIdentifiers&metadataPrefix=oai_dc&set=q-fin
`ListMetadataFormats`	Retrieve a list of the metadata formats that the repository supports. For example: http://arXiv.org/oai2?verb=ListMetadataFormats
`ListRecords`	Retrieve all metadata records in the repository. For example: http://arXiv.org/oai2verb=ListRecords&metadataPrefix=oai_dc&set=q-fin
`ListSets`	Retrieve the sets supported by the repository. Sets are predetermined groups of metadata records that identify related items. For example: http://arXiv.org/oai2?verb=ListSets

Other arguments

Some verbs require or allow extra arguments:

Parameter	Description
`from`	Restricts the results to metadata records modified on or after the given date. Optional for `ListIdentifiers` and `ListRecords`. For example: `from=1998-01-15`
`Identifier`	Specifies the metadata record to be retrieved. Required for `GetRecord`. For example: `identifier= oai:arXiv.org:cs/0112017`
`metadataPrefix`	Specifies the format of the retrieved metadata record(s). Required for `GetRecord`, `ListIdentifiers` and `ListRecords`. For example: `metadataPrefix=oai_dc`
`resumptionToken`	Beyond the scope of this document, see the OAI-PMH specification section on Flow Control.
`set`	Specify a set using a `setSpec` value for which we want to retrieve the identifiers or metadata records. Optional for `ListIdentifiers` and `ListRecords`. For example: `set=q-fin`
`until`	Restricts the results to metadata records modified on or before the given date. Optional for `ListIdentifiers` and `ListRecords`. For example: `until=2008-01-21`

Responses

Responses are returned in the XML format. The structure of the XML is mostly specified by the OAI-PMH. However, requests using the verbs `GetRecord` or `ListRecords` contain a metadata record section, the structure of which is specified by the `metadataPrefix` argument. Take, for example, the request:

http://arXiv.org/oai2?verb=GetRecord&identifier=oai:arXiv.org:cs/0112017&metadataPrefix=oai_dc

The `GetRecord` verb means that we are attempting to retrieve the metadata record for a specific item in the repository. The `identifier` argument specifies the particular item we are attempting to retrieve, and the `metadataPrefix` argument specifies the format of the metadata record section.

In the response below, the metadata record section is contained within the `<metadata>` tags. All other parts of the response are specified by the OAI-PMH:

```
<?xml version="1.0" encoding="UTF-8"?>
<OAI-PMH xmlns="http://www.openarchives.org/OAI/2.0/"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.openarchives.org/OAI/2.0/
         http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
  <responseDate>2002-02-08T08:55:46Z</responseDate>
  <request verb="GetRecord" identifier="oai:arXiv.org:cs/0112017"
           metadataPrefix="oai_dc">http://arXiv.org/oai2</request>
  <GetRecord>
   <record>
    <header>
      <identifier>oai:arXiv.org:cs/0112017</identifier>
      <datestamp>2001-12-14</datestamp>
      <setSpec>cs</setSpec>
      <setSpec>math</setSpec>
    </header>
    <metadata>
      <oai_dc:dc
        xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/"
        xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=
         "http://www.openarchives.org/OAI/2.0/oai_dc/
         http://www.openarchives.org/OAI/2.0/oai_dc.xsd">
        <dc:title>
          Using Structural Metadata to Localize Experience of
          Digital Content</dc:title>
        <dc:creator>Dushay, Naomi</dc:creator>
        <dc:subject>Digital Libraries</dc:subject>
        <dc:description>With the increasing technical
          sophistication of both information consumers and
          providers, there is increasing demand for more
          meaningful experiences of digital information. We
          present a framework that separates digital object
          experience, or rendering, from digital object storage
```

```
          and manipulation, so the rendering can be tailored to
          particular communities of users.
        </dc:description>
        <dc:description>Comment: 23 pages including 2 appendices,
            8 figures</dc:description>
        <dc:date>2001-12-14</dc:date>
      </oai_dc:dc>
    </metadata>
  </record>
 </GetRecord>
</OAI-PMH>
```

Repositories are only required to support one metadata format, unqualified Dublin Core (oai_dc), but may optionally support any number of additional formats. The formats that a repository supports can be queried using the `ListMetadataFormats` verb.

Sets

Sets are an optional OAI-PMH concept that allow repositories to group items. Sets are defined by three elements:

- `setSpec` - the unique identifier of a set.
- `setName` - a short string naming the set.
- `setDescription` - an optional description of the set.

Sets that are exposed by the repository may be listed using the `ListSets` request. They can also be used to limit the responses to the `ListIdentifiers` and `ListRecords` requests using the `set` argument.

Sets may be organized in a hierarchy. Hierarchical organization is specified by the syntax of the `setSpec` element. Each hierarchical item in the set is separated in the `setSpec` by a colon. Take, for example, the sets:

setName	setSpec
Institutions	`institution`
Oceanside University of Nebraska	`institution:nebraska`
Valley View University of Florida	`institution:florida`
Subjects	`subject`
Existential Kinesiology	`subject:kinesiology`
Quantum Psychology	`subject:quantum`

Here we have two main sets, institution and subject, each with two subsets.

The advantages of hierarchical sets are:

- Large sets can be partitioned into logically smaller groups.
- All subsets are returned in response to `ListIdentifiers` and `ListRecords` requests using the set argument containing a parent `setSpec`.

The `setSpec` element must only contain alphanumeric characters, colons or the characters: - _ . ! ~ * ' ( )

Other resources

Additional information about the OAI-PMH can be obtained from a number of sources, including:

- http://www.openarchives.org/pmh
- http://www.openarchives.org/OAI/openarchivesprotocol.html
- http://www.oaforum.org/tutorial/
- http://en.wikipedia.org/wiki/Open_Archives_Initiative_Protocol_for_Metadata_Harvesting

SECTION 2

The IMu OAI-PMH web service

The IMu OAI-PMH web service implements the OAI-PMH and uses an instance of the EMu Collections Management System (CMS) as its data store. Collectively, the web service and the EMu installation act as an OAI-PMH repository. The web service is implemented in the PHP programming language and uses the IMu toolset to harvest data from the EMu installation.

Requirements

The IMu OAI-PMH web service uses the PHP programming language. The service requires:

- PHP version 5.3.0 or later.
- The libxml PHP extension. The extension is usually available by default with the required PHP version, although some systems may maintain it separately.
- The XMLWriter extension. The extension is usually available by default with the required PHP version, although some systems may maintain it separately.

Using the `-m` switch with the command line interface to PHP displays a list of the available extensions. This list should include `libxml` and `xmlwriter`:

```
$ php -m
[PHP Modules]
Core
...
libxml
...
xmlwriter
...
```

PHP configuration

The following PHP configuration options should be set in the PHP configuration file (php.ini) for the service to run correctly:

Name	Description
`date.timezone`	Specifies the default time zone used by all PHP date / time functions. The OAI-PMH web service has to generate and parse a number of dates and times as part of its regular operation. For example: `date.timezone = Australia/Melbourne`
`default_socket_timeout`	Specifies the default timeout (in seconds) for socket based streams. The OAI-PMH web service uses a socket based stream to connect with the EMu Collections Management System via the IMu toolset. If the value is not large enough, the connection may be closed before a response is received. As the default value (60 seconds) may sometimes not be large enough, the value should be increased. For example: `default_socket_timeout = 600`

Configuration

The IMu OAI-PMH web service comprises:

- An implementation of the OAI-PMH.
- The implementations of the metadata formats that the IMu OAI-PMH web service supports.
- An interface to the EMu CMS via the IMu toolset.

Each of these is configured separately.

Localization

The setup of the IMu OAI-PMH repository configuration is such that it is possible to override or add to existing configurations. There are three levels of localization:

Level	Description
Default settings (`common`)	KE provides some default configuration settings and metadata formats (page 19) that are common to all clients. In practice there are not many configuration options that are useful at this level because most options are, at a minimum, client specific.
Client specific settings (`client`)	KE can also provide configuration and metadata formats that are specific to each client. These are set up, maintained and distributed by KE with the OAI-PMH repository code for the client.  Settings at this level override those at the `common` level.
Site specific settings (`local`)	Configurations at this level are set up and maintained by the customer. This allows customers to add to or modify configurations and metadata formats supplied by KE.  Settings at this level override those at the `common` and `client` level.

 Configuration files at the `common` and `client` levels are maintained and distributed by KE software with the IMu OAI-PMH web service. Clients must not modify these files as any changes they make will be lost when the service is upgraded. Changes or additions to the service not made by KE Software should be maintained at the `local` level. Files maintained here will be preserved when the service is upgraded.

Settings

Configuration files for the implementation of the OAI-PMH and for the interface to the EMu Collections Management System are located under the `config` directory:

Level	Location
Default settings (`common`)	`oai-pmh/ws/common/config`
Client specific settings (`client`)	`oai-pmh/ws/client/config`
Site specific settings (`local`)	`oai-pmh/ws/local/config`

Configuration files are specified in the INI file format. The allowed configuration files under the `config` directory are:

- `ws.ini`
- `oai.ini`
- `filter.ini`
- `sets.ini`

Web service

Web service settings are specified in the `ws.ini` file. This file has options for the interface to the EMu Collections Management System and options related to the web service that are not specifically related to the OAI-PMH. The options are:

Name	Description	Default Value	Mandatory	Multiple Allowed
`dateFormat`	The date format used by the EMu CMS.	`d#m#Y`	No	No
`fetchLimit`	The maximum number of records returned at one time by the EMu CMS.	`1000`	No	No
`host`	The host name of the server running the IMu server.	`localhost`	No	No
`port`	The port number (`main-port`) used by the IMu server.	`40000`	No	No
`setMode`	Specifies the repository support for sets. See Set Settings (page 17).	`off`	No	No
`supportedFormats`	The metadata formats, other than oai_dc that this EMu OAI-PMH service supports. See Metadata Formats (page 19).	`oai_dc`	No	Yes
`timeFormat`	The time format used by the EMu CMS.	`H#i`	No	No
`traceFile`	The path to a file for writing trace (debugging) information. The file must already exist and have permissions such that it can be written to by the web server (e.g. apache) process.	`(None)`	No	No
`traceLevel`	The level of detail to put in trace (debugging) information. The higher the number, the more detail.	`1`	No	No
`version`	The version of the KE IMu OAI-PMH web service library files to use. Currently only version 2.0 is supported.	`2.0`	No	No

OAI-PMH

OAI-PMH settings are specified in the `oai.ini` file. This file has options related to the OAI-PMH itself. The options are:

Name	Description	Default Value	Mandatory	Multiple Allowed
`adminEmail`	The email address of an administrator of the repository. Part of the response to the `Identify` request.	(None)	Yes	Yes
`earliestDatestamp`	A UTC datetime that is the guaranteed lower limit of all date stamps recording changes, modifications or deletions in the repository. Part of the response to the `Identify` request.	`1970-01-01T00:00:00Z`	No	No
`identifierPrefix`	The leading part of the `identifier` value returned with an OAI-PMH `header` response.	(None)	Yes	No
`repositoryName`	A human readable name for the repository. Part of the response to the `Identify` request.	`KE OAI-PMH Repository`	No	No

Filters

With filters it is possible to globally restrict the records returned from the EMu CMS.

Filters are specified in the `filter.ini` configuration file. The `name` of the filter, which is only used for diagnostic purposes, is specified as a section name with the other set details included in that section. Multiple sections can be specified. If multiple sections are specified, then **all** conditions must be met for a record to be returned. The options allowed in a section are:

Name	Description	Mandatory	Multiple Allowed

`term[]`	A two part option that specifies an EMu `column` and a `value`, in that order. Only records that have `value` in `column` will be included with the records returned from the EMu CMS. For example,
	`term[] = "Category_tab"`
	`term[] = "Music"`
	The trailing open and close square brackets `[]` are required for this option.
		Yes	Required

We can, for example, specify two filters with the names `department` and `music` respectively:

```
[department]
term[] = "SecDepartment_tab"
term[] = "Art"

[category]
term[] = "Category_tab"
term[] = "Music"
```

The columns used in the `filter.ini` file must exist in **all** of the modules from which you are retrieving data. See Metadata formats (page 19).

Sets

Sets allow repository items to be partitioned arbitrarily into groups. The IMu OAI-PMH web service supports three set modes. The set mode is specified in the `ws.ini` configuration file.

The modes are:

`off`

The `off` setting disables set support. This mode should be used if a repository does not require sets. In this mode the correct response is generated for the `ListSets` request; the correct response is also generated with the `ListIdentifiers` or `ListRecords` requests if the `set` argument is included. This is the default setting.

`on`

The `on` setting requires that the `setSpec` and `setName` values are stored in specialized EMu record columns, unlike the `legacy` mode which requires set details to be stored in a configuration file. This has a number of advantages over the `legacy` setting:

- Sets can be specified across multiple modules.
- Hierarchical sets are supported, which allows for a much finer grained specification of set membership.
- The performance of the `ListIdentifiers` and `ListRecords` requests is improved.
- Users can see, add or modify set membership (if they have the correct permissions) when viewing records in EMu.

The `on` setting requires additional database support. Please contact KE Software for details.

legacy

The `legacy` setting emulates the set support of the previous version of the IMu OAI-PMH web service. In this mode, sets are defined in the `sets.ini` file. The `setSpec` value is specified as a section name with the other set details included in that section. The options allowed in a section are:

Name	Description	Mandatory	Multiple Allowed
`name`	A short, human-readable string naming the set.	No	No
`description`	A description of the set. See the Collection and Set Descriptions section of the OAI-PMH Guidelines for Repository Implementers for more details. Only the Dublin Core `description` element scheme is supported.	No	No
`module`	The name of the EMu module to which the `term[]` option applies. For example: `module = "ecatalogue"`	Yes	No

`term[]`	A two part option that specifies an EMu `column` and a `value`, in that order. Any EMu record in `module` (see above) that matches this term (i.e. where `column` has `value`) is considered part of the set. For example:
	`term[] = "Category_tab"`
	`term[] = "Music"`
	The trailing open and close square brackets `[]` are required for this option.
		Yes	Required

We can, for example, specify two sets with a `setSpec` value of `music` and `video` respectively:

```
[music]
name = "Music Collection"
description = "Our music collection"
module = "ecatalogue"
term[] = "Category_tab"
term[] = "Music"
[video]
name = "Video Collection"
module = "ecatalogue"
term[] = "Category_tab"
term[] = "Video"
```

The `legacy` setting is deprecated and may be removed in a later version of the IMu OAI-PMH web service.

Metadata formats

Metadata formats specify the structure of the XML metadata record section of the responses to OAI-PMH `GetRecord` or `ListRecords` requests. A specific metadata format can be selected as part of these requests.

The metadata formats supported by a repository can be listed using the OAI-PMH `ListMetadataFormats` request. Support for the Dublin Core (oai_dc) metadata format is a requirement of the OAI-PMH.

Each metadata format that the IMu OAI-PMH web service defines is implemented as a PHP class. The class files are located under the `format` directory and can be localized in the same way as configuration files:

Level	Location
Default settings (`common`)	`oai-pmh/ws/common/format`
Client specific settings (`client`)	`oai-pmh/ws/client/format`
Site specific settings (`local`)	`oai-pmh/ws/local/format`

Each class must specify:

- The `metadataPrefix`, `schema` and `metadataNamespace` that identify the format. These values are displayed using the OAI-PMH `ListMetadataFormats` request. It is also necessary to specify the `version` of the metadata format that is implemented by this class.
- The modules and columns to retrieve from the EMu CMS.
- The XML metadata record section of the responses to OAI-PMH `GetRecord` or `ListRecords` requests.

Each class must also extend a provided class called `BaseMetadataFormat` (page 29). This means that the metadata format class inherits some already defined functionality specified in the `BaseMetadataFormat` class that is required by the IMu OAI-PMH web service.

The metadata formats supported by your repository must be specified in the `ws.ini` file `supportedFormats` option.

Creating a new metadata format

The steps to create a new metadata format class are:

1. Create a new file in the `local` (or `client` for KE supplied metadata formats) format directory. Name the file after the `metadataPrefix` of the metadata format that it implements and use a `.php` extension.
   For example: `oai_dc.php`
2. Open the file with your favourite editor. As the metadata format is written in the PHP programming language we must include PHP opening and closing tags at the beginning and end of the file:

```
<?php
…
?>
```

Make sure that you do not have any whitespace leading or trailing the PHP tags. PHP can include the whitespace with the output of the web service which can result in invalid XML.

3. Each class **must** extend the `BaseMetadataFormat` class so we have to specify the location to the file that contains that class:

```
require_once WS::$shared . '/BaseMetadataFormat.php';
```

4. Begin the class definition.
   We must specify the class name in a variable `$class` and then specify the class itself. The class name should be the name of the `metadataPrefix` followed by an underscore followed by the name of the format directory, i.e. `local` or `client`. We **must** extend the `BaseMetadataFormat` class.

```
$class = 'oai_dc_local';
class oai_dc_local extends BaseMetadataFormat
{
…
}
```

You should append the format directory name to the class name, for example `oai_dc_local` (or `oai_dc_client` for KE supplied metadata formats).

5. Define the class constructor.
   In the constructor you must call the parent (`BaseMetadataFormat`) constructor with four parameters, the `metadataPrefix`, `schema` and `metadataNamespace` that identify the format and the `version` of the format that you are implementing:

```
public function __construct()
{
  parent::__construct('oai_dc',
    'http://www.openarchives.org/OAI/2.0/oai_dc.xsd',
    'http://www.openarchives.org/OAI/2.0/oai_dc/',
    '2.0');
}
```

6. Define the `getColumnsByModule` function.
This specifies the modules and columns to be retrieved from the EMu CMS. The function must return an associative array that has module names as keys and an array of column names as values:

```
public function getColumnsByModule()
{
  $columnsByModule = array
  (
      'emultimedia' => array
      (
          'DetSubject_tab',
          'DetPublisher',
          'MulDescription',
          'MulMimeFormat',
          'MulMimeType',
          'MulTitle',
      ),
  );
  return($columnsByModule);
}
```

The columns `irn`, `AdmDateInserted`, `AdmTimeInserted`, `AdmDateModified` and `AdmTimeModified` are automatically added for each module so you do not have to specify them.

See *Specifying columns* (page 24) for more details.

7. Define the `writeMetadata` function.
The function actually outputs the XML metadata record section. The function must accept two parameters, an EMu module name `$name` and an array `$row`. The parameter `$name` is the name of a module specified in the `getColumnsByModule` function. The parameter `$row` is an associative array of column names to column values for a single EMu record.

```
public function writeMetadata($name, array $row)
{
  $this->writer->startElement('oai_dc:dc');

  $this->writer->writeAttribute('xmlns:oai_dc',
    'http://www.openarchives.org/OAI/2.0/oai_dc/');

  $this->writer->writeAttribute('xmlns:dc',
      'http://purl.org/dc/elements/1.1/');

  $this->writer->writeAttribute('xmlns:xsi',
    'http://www.w3.org/2001/XMLSchema-instance');

  $this->writer->writeAttribute('xsi:schemaLocation',
    'http://www.openarchives.org/OAI/2.0/oai_dc/' .
```

```
    'http://www.openarchives.org/OAI/2.0/oai_dc.xsd');

  $date = $this->getDateInserted($row);
  $this->writer->writeElement('dc:date', $date);

  if ($this->hasValue($row, 'MulDescription'))
  {
    $this->writer->writeElement('dc:description',
    $row['MulDescription']);
  }
  if ($this->hasValue($row, 'MulMimeFormat'))
  {
    $this->writer->writeElement('dc:format',
      $row['MulMimeFormat']);
  }
  if ($this->hasValue($row, 'DetSubject_tab'))
  {
    foreach ($row['DetSubject_tab'] as $subject)
    {
      $this->writer->writeElement('dc:subject',
        $subject);
    }
  }
  if ($this->hasValue($row, 'MulTitle'))
  {
    $this->writer->writeElement('dc:title',
      $row['MulTitle']);
  }
  if ($this->hasValue($row, 'MulMimeType'))
  {
    $this->writer->writeElement('dc:type',
      $row['MulMimeType']);
  }
  $this->writer->endElement();
}
```

See *Writing metadata* (page 26) for more details.

You should always check that a value for a column in the `$row` variable exists before trying to output it as XML, otherwise your output may be invalid. The `BaseMetadataFormat` class provides the `hasValue` method for this purpose. See *Reference* (page 29) for details on the methods provided by the `BaseMetadataFormat` class.

8. Add the `metadataPrefix` of the new metadata format to the `ws.ini` configuration file `supportedFormats` option.

See *Appendices* (page 35) for metadata format examples.

Extending an existing metadata format

The steps to modify an existing metadata format class are:

1. Create a new file. As for *Creating a new metadata format* (page 20).
2. Open the file. As for *Creating a new metadata format* (page 20).
3. As we are extending an existing metadata format, we have to specify the location of that class:
```
require_once WS::$base . '/common/formats/oai_dc.php';
```
4. Begin the class definition. As for *Creating a new metadata format* (page 20).
5. Unless we need to redefine the `schema` and `metadataNamespace` that identifies the format or the `version` of the metadata format we are extending, there is no need to define a class constructor.

If you want to redefine the `metadataPrefix` then you should be *Creating a new metadata format* (page 20).

6. Define the `getColumnsByModule` function. As for *Creating a new metadata format* (page 20).
7. Define the `writeMetadata` function. As for *Creating a new metadata format* (page 20).

Specifying columns

The columns specified in the `getColumnsByModule` function can be more than simple column names. It is also possible to:

- Retrieve columns from modules to which the current record attaches.

  For example, suppose that the Catalog module documents the creator of an object as an attachment (to a record in the Parties module) in a column called `CatCreatorRef`. If the Catalog module is searched, it is possible to get the creator's last name for each Catalog record in the result set as follows:

  ```
  'CatCreatorRef.NamLast'
  ```

  This technique can be extended to retrieve data for more than one column:

  ```
  'CatCreatorRef.(NamTitle,NamLast,NamFirst)'
  ```

- Retrieve columns from records that attach to the current record.

  For example, to retrieve information from a set of Catalog records which attach to the current Parties record via the Catalog's `CatCreatorRef` column, specify:

  ```
  '<ecatalogue:CatCreatorRef>.(irn,TitMainTitle)'
  ```

- Rename columns.

  It is possible to rename any column when it is returned by adding the new name in front of the real column being requested, followed by an equals sign. For example, to request data from the `NamLast` column but rename it to `last_name`, specify:

  ```
  'last_name=NamLast'
  ```

  The `$row` variable will contain an element called `last_name` rather than `NamLast`.

- Group nested table columns.

  Grouping allows an association between columns to be reflected in the structure of the data returned. To group the columns, surround them with square brackets:

  ```
  '[DocIdentifier_tab,DocMimeType_tab,DocMimeFormat_tab]'
  ```

  The `$row` variable will contain an element, called `group1`, where each corresponding row of the tables specified in the group is listed together:

```
[group1] => Array
    (
        [0] => Array
            (
                [DocMimeFormat_tab] => tiff
                [DocMimeType_tab] => image
                [DocIdentifier_tab] => image.tif
            )

        [1] => Array
            (
                [DocMimeFormat_tab] => jpeg
                [DocMimeType_tab] => image
                [DocIdentifier_tab] => image.thumb.jpg
            )
    )
```

In the following example the elements are not grouped:

```
[DocMimeType_tab] => Array
    (
        [0] => image
        [1] => image
    )

[DocIdentifier_tab] => Array
    (
        [0] => image.tif
        [1] => image.thumb.jpg
    )

[DocMimeFormat_tab] => Array
    (
        [0] => tiff
        [1] => jpeg
    )
```

By default, the group is given a name of `group1`, `group2` and so on, which can be changed easily enough:

```
'multimedia=[DocIdentifier_tab,DocMimeType_tab,DocMimeFormat_tab]'
```

More details can be found in the document Using KE IMu's PHP API.

Writing metadata

XMLWriter

The `BaseMetadataFormat` class property `writer` is an instance of the PHP `XMLWriter` class and can be accessed by any class that `extends` the `BaseMetadataFormat` class.

When defining a metadata format's `writeMetadata` function, the `writer` property can be used to output any arbitrary XML. The `XMLWriter` class defines a number of methods for writing XML:

- `startElement`: start an XML element tag.
- `writeAttribute`: write an XML attribute name and value.
- `text`: write an XML text node.
- `endElement`: end an XML attributes tag.
- `writeElement`: start an XML element tag, write a text node and end the XML element tag.

For example, the following methods:

```
$this->writer->startElement('start');
$this->writer->writeAttribute('name', 'value');
$this->writer->text('this is some text');
$this->writer->endElement();
```

will output:

```
<start name="value">this is some text</start>
```

and the following method:

```
$this->writer->writeElement('start', 'this is some more text');
```

will output

```
<start>this is some more text</start>
```

It is your responsibility to produce valid XML output, although the `XMLWriter` will automatically escape predefined entities in any text output.

See the PHP XMLWriter documentation for more details.

Output

As it is possible to use all of the facilities of the PHP programming language, the options for writing XML are very flexible. It is possible to output static (hard coded) text or mix static and dynamic text:

```
$this->writer->writeElement('dc:publisher',
  'The National Museum');

$this->writer->writeElement('dc:identifier',
  'http://nationalmusuem.com/collection?object=' .
  $row['irn']);
```

We can conditionally output text:

```
if ($this->hasValue($row, 'DetPublisher') &&
    $row['DetPublisher'] == 'NM')
{
  $this->writer->writeElement('dc:publisher',
    'The National Museum');
}
```

We can also output entirely dynamic content:

```
if ($this->hasValue($row, 'DetPublisher'))
{
  $this->writer->writeElement('dc:publisher',
    $row['DetPublisher']);
}
```

SECTION 3

Reference

Class BaseMetadataFormat

The abstract parent class for all metadata formats.

Constructor

```
public __construct(string $metadataPrefix, string $schema,
        string $metadataNamespace, string $version)
```

Instantiate the object. Must be called from the constructor of the child class.

Parameters

`string $metadataPrefix`

A string that uniquely identifies the metadata format within the repository.

`string $schema`

A URL string that specifies the XML schema of the metadata format.

`string $metadataNamespace`

A URL string that specifies the XML namespace of the metadata format.

`string $version`

A string that specifies the version of the metadata format.

Properties

`string $metadataNamespace`	The `metadataNamespace` parameter specified in the constructor.
`string $metadataPrefix`	The `metadataPrefix` parameter specified in the constructor.
`string $schema`	The `schema` parameter specified in the constructor.
`string $version`	The `version` parameter specified in the constructor.
`XMLWriter $writer`	An instance of the `XMLWriter` class.

Abstract methods

Abstract methods must be implemented by any class that extends this class.

```
public abstract array getColumnsByModule()
```

Get an associative array of modules and columns to retrieve from the EMu CMS.

`Returns` An associative array with module names as keys and arrays of columns as values.

```
public abstract void writeMetadata(string $module, array $row)
```

Write a single metadata element.

Parameters

`string $module` The name of an EMu module.

`array $row` The data from a single row (or record) from the module specified in `$module`.

Methods

```
protected string getDate(string $date, string $time = null)
```

Take a date and, optionally, a time string and return a date/time string in ISO8601 format.

Parameters

`string $date` A date string. Must be in the format specified by the `ws.ini dateFormat` option.

`string $time` A time string. Must be in the format specified by the `ws.ini timeFormat` option.

`Returns` A date/time string in ISO8601 format.

```
protected string getId(string $module, array $row)
```

Utility function that takes a module name and module row (e.g. as supplied to the `writeMetadata` function) and returns a string that uniquely identifiers that row.

Parameters

`string $module` The name of a module.

`array $row` A row (record) from the module specified in the `$module` parameter.

`Returns` A string that uniquely identifies the module row.

`protected string getDateInserted(array $row)`

Utility function that takes a module row (e.g. as supplied to the `writeMetadata` function) and returns a date/time string in ISO8601 format specifying when `$row` was inserted.

Parameters

`string $row` A module row (record).

`Returns` A date/time string in ISO8601 format specifying when `$row` was inserted.

`protected string getDateModified(array $row)`

Utility function that takes a module row (e.g. as supplied to the `writeMetadata` function) and returns a date/time string in ISO8601 format specifying when `$row` was last modified.

Parameters

`array $row` A module row (record).

`Returns` A date/time string in ISO8601 format specifying when `$row` was last modified.

`protected boolean hasValue(array $row, string $column)`

Utility function that takes a module row and a column name and returns a boolean value indicating whether the module row contains a value for the specified column.

Parameters

`array $row` A module row (record).

`string $column` A column name.

`Returns` A boolean indicating whether `$row` contains a value for `$column`.

SECTION 4

Glossary

Harvesting	Collecting and conglomerating data from multiple sources into a single store.
Header	The part of certain OAI-PMH responses that contains information that can be used for selective harvesting.
IMu OAI-PMH web service	KE Software's implementation of an OAI-PMH repository.
KE EMu	KE Software's Electronic Museum management system.
KE IMu	IMu, or Internet Museum, broadly describes KE Software's strategy and toolset for distributing data held within EMu via the Internet.
Metadata	The data component, or payload, of certain OAI-PMH responses.
Metadata format	The XML structure of the data component, or payload, of an OAI-PMH response. This term is also used to refer to the components of the IMu OAI-PMH web service that generate the payload.
metadataNamespace	Part of the specification of a metadata format. It specifies the URL of an XML namespace for the metadata format.
metadataPrefix	Part of the specification of a metadata format. It specifies a string that uniquely identifies the metadata format.
Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)	A relatively simple protocol for harvesting metadata.
PHP	A general-purpose server-side scripting language.
Protocol	A specification of message formats and rules for exchanging information between computer systems.
Repository	A web service that implements the OAI-PMH.
schema	Part of the specification of a metadata format. It specifies the URL of an XML schema for the metadata format.

Selective Harvesting	Limit requests to portions of the metadata available from a repository.
Set	An OAI-PMH concept for grouping repository items.
Verb	The name of an OAI-PMH request type. Included as part of the OAI-PMH request.
Web service	A method of communication between two electronic devices over the Web.

SECTION 5

Appendices

Appendix A: Base Darwin Core metadata format

```
<?php

require_once IMu::$lib . '/Trace.php';
require_once WS::$shared . '/BaseMetadataFormat.php';

$class = 'oai_dc';
class oai_dc extends BaseMetadataFormat
{
  public function __construct()
  {
      parent::__construct('oai_dc',
          'http://www.openarchives.org/OAI/2.0/oai_dc.xsd',
          'http://www.openarchives.org/OAI/2.0/oai_dc/',
          '2.0');
  }

  public function getColumnsByModule()
  {
      $columnsByModule = array
      (
          'emultimedia' => array
          (
              'ChaMediaForm',
              'DetContributor_tab',
              'DetCoverage',
              'DetLanguage_tab',
              'DetPublisher',
              'DetRelation_tab',
              'DetResourceType',
              'DetRights',
              'DetSource',
              'DetSubject_tab',
              'MulCreator_tab',
              'MulDescription',
              'MulIdentifier',
              'MulMimeFormat',
              'MulMimeType',
              'MulTitle',
          ),
      );
      return($columnsByModule);
  }
```

```
  public function writeMetadata($name, array $row)
  {
      $this->writer->startElement('oai_dc:dc');
      $this->writer->writeAttribute('xmlns:oai_dc',
          'http://www.openarchives.org/OAI/2.0/oai_dc/');
      $this->writer->writeAttribute('xmlns:dc',
          'http://purl.org/dc/elements/1.1/');
      $this->writer->writeAttribute('xmlns:xsi',
          'http://www.w3.org/2001/XMLSchema-instance');
      $this->writer->writeAttribute('xsi:schemaLocation',
          'http://www.openarchives.org/OAI/2.0/oai_dc/' .
         ' http://www.openarchives.org/OAI/2.0/oai_dc.xsd');

      $this->writeElement('dc:contributor', $row,
          'DetContributor_tab');
      $this->writeElement('dc:coverage', $row, 'DetCoverage');
      $this->writeElement('dc:creator', $row, 'MulCreator_tab');

      $this->writer->writeElement('dc:date',
$this->getDateInserted($row));

      $this->writeElement('dc:description', $row, 'MulDescription');
      $this->writeElement('dc:format', $row, array('MulMimeFormat',
          'ChaMediaForm'));
      $this->writeElement('dc:language', $row, 'DetLanguage_tab');
      $this->writeElement('dc:publisher', $row, 'DetPublisher');
      $this->writeElement('dc:relation', $row, 'DetRelation_tab');
      $this->writeElement('dc:rights', $row, 'DetRights');
      $this->writeElement('dc:source', $row, 'DetSource');
      $this->writeElement('dc:subject', $row, 'DetSubject_tab');
      $this->writeElement('dc:title', $row, 'MulTitle');
      $this->writeElement('dc:type', $row, array('MulMimeType',
          'DetResourceType'));

      $this->writer->endElement();
  }

  protected function writeElement($field, $row, $columns)
  {
      if (! is_array($columns))
      {
          $columns = array($columns);
      }
      foreach ($columns as $column)
      {
          if (! $this->hasValue($row, $column))
          {
              continue;
          }
          $values = $row[$column];
          if (! is_array($values))
          {
              $values = array($values);
          }
```

```
            foreach ($values as $value)
            {
                $this->writer->writeElement($field, $value);
            }
        }
    }
}
?>
```

Appendix B: Extended Darwin Core metadata format

```
<?php

require_once IMu::$lib . '/Trace.php';
require_once WS::$base . '/common/formats/oai_dc.php';

$class = 'oai_dc_client';
class oai_dc_client extends oai_dc
{
  public function getColumnsByModule()
  {
    $columnsByModule = array
    (
      'emultimedia' => array
      (
        'ChaMediaForm',
        'DetContributor_tab',
        'DetCoverage',
        'DetLanguage_tab',
        'DetPublisher',
        'DetRelation_tab',
        'DetResourceType',
        'DetRights',
        'DetSource',
        'DetSubject_tab',
        'MulCreator_tab',
        'MulDescription',
        'MulIdentifier',
        'MulMimeFormat',
        'MulMimeType',
        'MulTitle',
      ),
    );
    return($columnsByModule);
  }

  public function writeMetadata($module, array $row)
  {
    $this->writer->startElement('oai_dc:dc');
    $this->writer->writeAttribute('xmlns:oai_dc',
      'http://www.openarchives.org/OAI/2.0/oai_dc/');
    $this->writer->writeAttribute('xmlns:dc',
      'http://purl.org/dc/elements/1.1/');
    $this->writer->writeAttribute('xmlns:xsi',
      'http://www.w3.org/2001/XMLSchema-instance');
    $this->writer->writeAttribute('xsi:schemaLocation',
      'http://www.openarchives.org/OAI/2.0/oai_dc/' .
     ' http://www.openarchives.org/OAI/2.0/oai_dc.xsd');

    /* Using the writeElement method inherited from the oai_dc class.
     */
```

```
    $this->writeElement('dc:contributor', $row,
      'DetContributor_tab');
    $this->writeElement('dc:coverage', $row, 'DetCoverage');
    $this->writeElement('dc:creator', $row, 'MulCreator_tab');

    $this->writer->writeElement('dc:date',
$this->getDateInserted($row));

    $this->writeElement('dc:description', $row, 'MulDescription');
    $this->writeElement('dc:format', $row, array('MulMimeFormat',
      'ChaMediaForm'));
    $this->writeElement('dc:language', $row, 'DetLanguage_tab');
    $this->writeElement('dc:publisher', $row, 'DetPublisher');
    $this->writeElement('dc:relation', $row, 'DetRelation_tab');
    $this->writeElement('dc:rights', $row, 'DetRights');
    $this->writeElement('dc:source', $row, 'DetSource');
    $this->writeElement('dc:subject', $row, 'DetSubject_tab');
    $this->writeElement('dc:title', $row, 'MulTitle');
    $this->writeElement('dc:type', $row, array('MulMimeType',
      'DetResourceType'));

    $this->writer->endElement();
  }
}
?>
```

Index