Implementation Guidelines for the Open Archives Initiative Protocol for Metadata Harvesting

Guidelines for Repository Implementers

Protocol Version 2.0 of 2002-06-14
Document Version 2005/01/19T19:27:00Z
http://www.openarchives.org/OAI/2.0/guidelines-repository.htm

Editors

Carl Lagoze (OAI Executive; Cornell University - Computer Science)
Herbert Van de Sompel (OAI Executive; Los Alamos National Laboratory - Research Library)
Michael Nelson (Old Dominion University - Computer Science)
Simeon Warner (Cornell University - Computer Science)

This document is one part of the Implementation Guidelines that accompany the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH).

Table of Contents

1. Introduction

The OAI-PMH is designed to provide a low barrier to implementation for repositories. As such, there are many constructs that are optional in the OAI-PMH. This document first lists the requirements for a minimal repository implementation and then describes best practices relevant for repository implementers and maintainers.

2. Minimal Repository Implementation

It is important to stress that there are many optional concepts in the OAI-PMH. The aim is to allow for high fidelity communication between repositories and harvesters when available and desirable. However, they remain optional for situations when these constructs are not needed or implementation would be too difficult. A list and discussion of optional features of the OAI-PMH is given below. Repository implementers may implement all, some or none of these features.

2.1 Dublin Core and Other Metadata Formats

Dublin Core (DC) is the resource discovery *lingua franca* for metadata. Since all DC fields are optional and repeatable, most repositories should have no trouble creating at least a minimal mapping of their native metadata to unqualified DC. Repositories are not required to store their metadata in DC; DC is something that is frequently "converted to", rather than "stored in". Many repositories store their metadata in some other format, and dynamically convert to DC in response to harvester requests. Repositories may, of course, choose to store DC directly if it is desirable in a particular implementation.

Although repositories are strongly encouraged to expose richer, possibly community-specific metadata formats, there is no requirement to do so. Repository implementers should consider exporting DC the first and most important step toward OAI-PMH interoperability. The addition of facilities to export other formats may then be added at a later date.

2.2 `<about>` Containers

Records in OAI-PMH are particular disseminations of metadata from items in the repository. `<about>` containers allow the repository to include self-referential metadata about the records themselves. While this may prove to be an invaluable tool for encoding rights and provenance statements, it is not required. It is recommended that implementers defer the use of `<about>` containers if they are uncertain of their use or ramifications.

2.3 Sets

Sets provide a method of exposing a partitioning of a repository's contents to harvesters. While this allows for sophisticated harvesting, not all harvesters will exploit this capability. Like non-DC metadata formats, sets are most likely to be useful within specific communities. It is recommended that implementers defer the implementation of sets until there is a particular community-specific situation or deployment scenario that needs sets. Implementation of sets is optional; repository implementers may choose not to implement sets (harvesters may also ignore sets).

2.4 Response Compression

The ability to send compressed responses to harvesters can be a significant performance optimization. However, this depends on: 1) harvesters being able to accept compressed responses, and 2) sufficiently lengthy responses are available to warrant the additional time required to compress and uncompress the responses by the repository and harvester, respectively. It is recommended that repositories defer the implementation of response compression unless they have large collections (e.g., several thousand records) and until the time when harvesters widely adopt the compression mechanism.

All repositories must support `identity` (uncompressed) encoding and must reply with that unless some other recognized encoding is parsed from the `Accept-Encoding:` header of the request.

2.5 Flow Control

Flow control, through the use of incomplete list responses and `resumptionToken` elements, is a powerful mechanism that allows a repository to throttle incoming requests from harvesters. However, depending on the repository, this can be one of the more complex parts of the OAI-PMH to implement. While there are no definite guidelines regarding the size of a list response that should result in an incomplete-list responses, it is reasonable that repositories with less than 1000 items reply with a single complete-list responses (i.e. do not use `resumptionToken` elements).

3. Datestamp granularity: `datestamp` and `responseDate` values

The OAI-PMH permits two `datestamp` granularities: days and seconds. All datestamps must be expressed in UTC and therefore the day-to-day transition occurs at midnight UTC and not, in general, at the local midnight of the repository. It is expected that most repositories will use the seconds granularity. However, repositories with particular update schedules or based on existing software may choose to use day granularity if that reflects the underlying processes.

Regardless of the granularity used, metadata must be available from new or modified items during the period of the item's new `datestamp`. If this is not the case then that update may be missed by an incremental harvest. For example, if an item is added to a repository with `datestamp` `2002-03-01` and but this shows up in an index created at `2002-04-01T02:00Z`. Then then an incremental harvest which runs at `2002-04-01T01:00Z` will miss the update from the previous day, and the follow-on harvest specifying `from=2002-04-01` will also miss the update.

The `datestamp` values used for OAI are for the purpose of incremental harvesting. They should be changed to reflect any change in the item which includes any change to any of the disseminations. For example, even if the underlying master-record within a repository is not changed, a change to the way that some formats are generated on the fly should result in updates to the datestamps of all items for which that format may be disseminated. One way that this may be implemented internally when building a `datestamp` index is:

```
if (internalItemDatestamp > disseminationInterfaceDatestamp) {
  datestamp = internalItemDatestamp
} else {
  datestamp = disseminationInterfaceDatestamp
}
```

Harvesters may use the `responseDate` to obtain the time (expressed in UTC) from the repository's clock corresponding to the start of a list request as a way to implement incremental harvesting. By using the time returned by the repository, harvesters avoid problems with clock skew. It is thus recommended that the `responseDate` reflect the time of the repository's clock at the start of any database query or search function necessary to answer the list request, rather than when the output is written. This will ensure that, even if the query takes some time, the next incremental harvest will catch any updates which occurred during the query.

4. Sets

Sets provide optional support for selective harvesting. Repositories need not implement them (see Minimal Implementation: Sets) and harvesters may ignore any set structure that is exposed.

4.1 Set Hierarchy

Use of the colon (`:`) character in `setSpec` values implies a hierarchy of sets. Repositories that do not implement a hierarchy of sets must not include colons in their `setSpec` values.

Example `ListSets` response:

```
<ListSets ...>
  <set>
    <setSpec>A</setSpec>
    <setName>set A</setName>
  </set>
  <set>
    <setSpec>A:B</setSpec>
    <setName>set A:B</setName>
  </set>
  <set>
    <setSpec>B</setSpec>
    <setName>set B</setName>
  </set>
  <set>
    <setSpec>B:C</setSpec>
    <setName>set B:C</setName>
  </set>
  <set>
    <setSpec>B:D</setSpec>
    <setName>set B:D</setName>
  </set>
  <set>
    <setSpec>B:D:E</setSpec>
    <setName>set B:D:E</setName>
  </set>
</ListSets>
```

This response implies the following hierarchy of sets:

```
A
A:B
B
B:C
B:D
B:D:E
```

The sets `A:B` and `B` are distinct, however the repeated use of node name `B` is not recommended because it may lead to confusion. If the header for `item1` is:

```
  <header>
    <identifier>item1</identifier>
    <setSpec>A:B</setSpec>
    ...
  </header>
```

then `item1` is organized in sets `A` and `A:B`. If the header for `item2` is:

```
  <header>
    <identifier>item2</identifier>
    <setSpec>A</setSpec>
    <setSpec>B:D:E</setSpec>
    ...
  </header>
```

then `item2` is organized in sets `A`, `B`, `B:D` and `B:D:E`.

4.2 Collection and Set Descriptions

Set descriptions may be included in the `setDescription` of the `ListSets` response. If sets correspond to some notion of "collections" then this allows collection description. If the whole repository represents one collection then it may be more appropriate to use the `description` container in the `Identify` response to describe the collection.

Communities are able to develop their own collection description XML schemas for use within the `description` container and `setDescription` elements. If all that is desired is the ability to include an unstructured textual description then it is recommended that repositories use the Dublin Core `description` element. Seven existing schemes are:

Dublin Core	The 15 DCMES elements (unqualified)	http://www.dublincore.org/documents/dces/ OAI-PMH: Dubin Core, `oai_dc`
Encoded Archival Description (EAD)	A standard for encoding archival finding aids. EAD instances used effectively for this purpose would probably be limited to information about the collection as a whole and significant high-level components.	http://www.loc.gov/ead/
The `eprints` schema	A schema for describing eprint archives.	OAI-PMH Guidelines: `eprints`
RSLP collection description schema	An application of qualified Dublin Core built on an underlying model of collections and their descriptions.	http://www.ukoln.ac.uk/metadata/rslp/schema/
UDDI/WSDL	The Universal Description Discovery and Integration specification provides a mechanism for describing services on the Internet.	http://www.uddi.org/
MARC21	Commonly used in library catalogs to describe entire collections in a single record.	http://www.loc.gov/marc/ OAI-PMH Guidelines: `oai_marc`
The `branding` schema	A schema for including branding information: icon, link etc.	OAI-PMH Guidelines: `branding`

Note that appropriate XML schema may not yet exist for EAD, RSLP and UDDI. Also note that mappings between the DC, EAD, RSLP and MARC21 schemas are available.

The OAI community is encouraged to track the work of the DCMI Collection Description working group, who are planning to develop a simple collection description schema based on qualified DC.

5. Incomplete-List Responses and the Use of `resumptionToken` Elements

The OAI-PMH does not specify the syntax or even suggest an implementation strategy for `resumptionToken` elements. This is to allow maximum flexibility for individual repositories to adapt this mechanism to their own deployment profile. Harvesters are required to treat `resumptionToken` values as opaque data objects. A repository may choose to make the syntax of a `resumptionToken` obvious, such as:

```
<resumptionToken>resultSet=157&nextRange=1001-2000</resumptionToken>
```

or may choose to encode the `resumptionToken` so the syntax is not visible, such as:

```
  <resumptionToken>9023A210CD007</resumptionToken>
```

The OAI-PMH was designed to provide a mechanism whereby a harvester can incrementally harvest from a repository and not miss any changes (additions, modifications, and deleted reported with `deleted` status). The implementation of incomplete list responses and `resumptionToken` elements should not change this. Whether a change is harvested during a particular harvest or on the subsequent harvest is probably not critically important, provided it is not missed in the sequence of harvests. For example, consider the case of an item modified during the sequence of requests necessary to complete a `ListRecords` request. The new `datestamp` will correspond to the time after the start, but before the end, of the sequence of requests. It is quite reasonable for this change to be picked up by the current harvest *or* for it not to be picked up until the following harvest.

There are two main implementation strategies that a repository may choose when implementing incomplete-list responses and `resumptionToken` elements: to encode the state in the `resumptionToken`; or to cache the result set. These approaches are described in the next two sections.

5.1 Encoding State in the `resumptionToken`

By encoding all state in the `resumptionToken`, a repository can remain stateless. This implementation strategy trades the management overhead required to cache results for possible increased computation time in regenerating the state necessary for the next response. Consider the following example:

```
  http://an.oai.org/?verb=ListRecords&set=227&from=1999-02-03
```

A `resumptionToken` may be similar to (wrapped for clarity):

```
  <resumptionToken>set=227&from=1999-02-03&until=2002-04-01
    &range=751-1500&metadataPrefix=oai_dc</resumptionToken>
```

This example is for a `ListRecords` request and so the `resumptionToken` includes the `metadataPrefix` as part of the state; the `metadataPrefix` part would not be required for a `ListIdentifiers` `resumptionToken`. Harvesters may omit either or both of the `from` and `until` arguments to say `from the earliest datestamp' and/or `until the last datestamp'. It is recommended that any `resumptionToken` constucted from a request such as this have at least the part that corresponds to the `until` argument filled in with the current repository datestamp. While this will not avoid the incomplete-list requests including new items with the same datestamp as the `until` value (likely if day granularity is used), it will avoid the incomplete-list requests including new or modified items with later datestamps.

One way to implement this form of `resumptionToken` is to encode the original arguments of the list request and some offset information in the `resumptionToken`. When the repository later receives the `resumptionToken` it can then repeat the original query to regenerate the state. This may be computationally expensive. However, there is no cache management overhead associated with the `resumptionToken` values. For this reason alone, it is recommended that unless implementers are comfortable with managing a cache, repositories should use a scheme whereby all necessary state is encoded in the `resumptionToken` values.

Note that some additional information may need to be encoded in the `resumptionToken` to cope with the case where items move outside the `datestamp` range of the initial request before the list request sequence is complete.

5.2 Caching the Result Set

A repository may cache the results of a harvester's list request and then return segments of this as incomplete list responses (accessible through further requests with the appropriate `resumptionToken` values). A typical solution would be to encode both a session identifier and a cursor in the `resumptionToken`. If harvester issued a request:

```
  http://an.oai.org/?verb=ListRecords&set=227&from=1999-02-03
```

A `resumptionToken` in the response could look similar to:

```
<resumptionToken>searchId=4234&range=751-1500</resumptionToken>
```

Subsequent requests from the harvester would involve the repository only sending the next 750 records (for example) from the cached result set. A repository could also check the result set cache for duplicate searches and if two different harvesters issued the same request, they could be served from the same cached result set. Depending on the nature of the repository and the community it serves, this scenario could turn out to be common, and caching results could be a significant optimization for the repository.

A downside of this approach is that the repository has the additional responsibility of maintaining and purging the cache. Given the limited amount of knowledge the OAI community currently has regarding harvesting profiles, it is not known which cache replacement algorithm would be best for a given repository. The wide range of repository deployment profiles may make a general recommendation impossible.

Repositories implementing this sort of `resumptionToken` should use the `expirationDate` attribute to indicate how long the `resumptionToken` values will remain valid. If only the identifiers are cached for `ListRecords` and `ListIdentifers` requests then it may also be necessary to check that items have not been deleted or that `datestamps` have not moved out of the range of the request by the time that the metadata is actually disseminated.

5.3 Idempotency of List Requests

The OAI-PMH requires a weak form of idempotency for both complete list requests and individual incomplete list requests using `resumptionToken` arguments:

- If there are no changes in the repository then two identical list requests **must** return the same result.
- If there are no changes in the repository then re-issuing the most recent incomplete list request (using the same, non-expired `resumptionToken`) **must** return the same result. A session identifier cannot be used as a `resumptionToken` without additional state information because it must be possible to re-use a `resumptionToken` to re-issue an incomplete list request. It is, however, permitted for repositories to expire `resumptionToken` values; the time at which expiration occurs may be specified in the `expirationDate` attribute.
- If there are changes in the repository between two identical list or incomplete list requests then both responses must include all elements that have not changed. The implication of this is that individual incomplete list requests must have the same degree of idempotency as the complete list request.
- Only the `resumptionToken` most recently received by a harvester may be re-issued. The implication for repositories is that they must support the above idempotency requirement for the most recent incomplete list request that was answered. They must accept both the most recent `resumptionToken` issued and the previous `resumptionToken` issued.

There are scenarios where different semantics may result from the two implementation strategies described above. If a repository changes rapidly compared with the rate at which a harvester issues follow-on requests with `resumptionToken` arguments, the two different implementations can yield different results. Consider a repository that adds *N* new records midway through a harvester's session, and these records match the criteria specified in the initial request (`set`, `timestamp`, etc.). Implementations relying on state stored by the repository are likely to give the results relative to the time of the initial harvest. Implementations transferring state in the `resumptionToken` values are likely to give the results relative to the time of the last request (of some intermediate time). Sophisticated implementations of either scheme could accommodate for this difference, but it is recommended that implementers consider the anticipated accession rate of the repository and set the optional `resumptionToken` attribute `expirationDate` accordingly.

Idempotency of `resumptionTokens` will aid harvesters in recovering from the unavoidable error conditions that can occur when harvesting large repositories (e.g., lost responses, crashed harvesters, etc.).

5.4 Expiration and `expirationDate`

The `expirationDate` attribute may be used to indicate when a `resumptionToken` expires. It is recommended that repositories use `resumptionToken` values that remain valid for at least tens of minutes.

If a repository uses `resumptionToken` values that do not expire, then the `expirationDate` attribute should not be used.

If, for some reason, a repository cannot continue to complete a request then it may issue a `badResumptionToken` (with suitable text explanation for human readers) to stop the sequence of requests. A harvester would be expected to start again from the initial list request.

5.5 The `completeListSize` and `cursor` attributes

The `completeListSize` and `cursor` attributes of the `resumptionToken` element may optionally be included to provide extra information about the size of the result set and the number of results so far disseminated.

The `cursor` attribute is simply the number of results returned so far in the complete list response, thus it is always "0" in the first incomplete list response. It should only be specified if it is consistently used in all responses.

The `completeListSize` attribute provides a place where the estimated number of results in the complete list response may be announced. This is likely to be used for status monitoring by harvesting software and implementation is recommended especially in repositories with large numbers of records. The value of `completeListSize` can be reliably accurate only in the case of a system where the result set is cached. In other cases, it is permissible for repositories to revise the estimate during a list request sequence.

6. Flow Control and Load Balancing

The related but orthogonal concepts of *flow control* and *load balancing* allow a repository to throttle incoming requests. Flow control, as available through incomplete lists and `resumptionToken` elements, can be viewed as an *intra-repository* technique, and the semantics of flow control are defined in the OAI-PMH. Load balancing can be viewed as an *inter-repository* technique to redirect the harvesting load from busy repositories to lightly loaded repositories with the same contents. Load balancing semantics are not addressed in the OAI-PMH and are expected to be handled at the transport layer.

Both concepts are optional, and both may be used independently or in combination. It is easy to imagine a scenario in which a harvester is redirected to a lightly loaded "mirror" repository, and the resulting requests and responses between that repository and the harvester are throttled through the use of `resumptionToken` elements.

6.1 Flow Control

The semantics of `resumptionToken` elements are defined in the OAI-PMH, and possible implementation strategies are discussed above. The rest of this section addresses transport-level flow control methods.

The HTTP protocol defines a status code of 503 - "Service Unavailable". If a repository is heavily loaded, instead of processing the request (and returning HTTP status code 200), it can issue a HTTP 503 response with a "Retry-After" header. This mechanism might also be used to delay requests during database updates or other short outages.

If a repository wants to slow a sequence of requests then it can issue a HTTP 503 response, which informs the harvester that its request was not processed and it should wait the specified amount of time before retrying. At the repository's discretion, it may issue HTTP 403 "Forbidden" responses to harvesters that do not adhere to the delay specified in the `Retry-After` header of the 503 response. A 403 is more punitive than a 503, and should be issued only in extreme situations.

It is recommended that a waterfall model of flow control be adopted, with repositories moving to the next level only after a harvester appears unresponsive at the current level:

1. HTTP status code 200; response to OAI-PMH request, perhaps with a `resumptionToken`.
2. HTTP status code 503; with the `Retry-After` header set to an appropriate value if subsequent request follows too quickly or if the server is heavily loaded.
3. HTTP status code 403; with an appropriate reason specified if subsequent requests do not adhere to `Retry-After` delays.

6.2 Load Balancing

Although not addressed in the OAI-PMH, it is possible to build hierarchies of cooperating repositories. Especially popular or heavily loaded repositories might set up mirror repositories. Load balancing can then be accomplished through the use of HTTP 302 or 303 status codes, or even through the use of CNAMES in DNS. Load balancing is powerful, yet subtle. It is recommended that repositories implement load balancing only when: 1) the canonical repository is clearly overloaded, and 2) they are comfortable with details of OAI-PMH.

6.2.1 HTTP-Based Load Balancing

For example, consider the request:

```
  http://an.oai.org/?verb=ListRecords&set=227&from=1999-02-03
```

If `an.oai.org` is especially busy, it may choose to issue a 302 with the `Location` header of:

```
http://another.oai.org/?verb=ListRecords&set=227&from=1999-02-03
```

Which implies that `another.oai.org` is a mirror of (or "slave" to) the canonical `an.oai.org`. If this request generates an incomplete list, the successive harvesting requests will be relative to `another.oai.org`. However, all future harvesting requests must be directed to `an.oai.org`. To this end, if an `Identify` request is issued to `another.oai.org`, the response must contain `an.oai.org` in the `baseURL`. Similarly, the contents of the `request` element in the header of the response must be the `baseURL` (`an.oai.org`).

Note that `another.oai.org` is free to load balance as well, issuing a 302 with the `Location` header of:

```
http://yetanother.oai.org/?verb=ListRecords&set=227&from=1999-02-03
```

It is up to the implementers to ensure there are no circular redirections for the harvester.

6.2.2 DNS-Based Load Balancing

A simpler, but slightly less powerful load balancing strategy is to use a DNS rotor. In this case, `an.oai.org` might resolve to one of: `a.oai.org`, `b.oai.org` or `c.oai.org`. Which presumably all map to physically separate machines. The harvester resolving the name `an.oai.org` would randomly receive one of the addresses and use that address for the harvesting requests. Repositories wishing to implement this form of load balancing must ensure tight synchronization between the machines answering the requests. They must also use `an.oai.org` for `baseURL` (in `Identify` responses) and `request` (in response headers).

7. Response Compression

Compression in the OAI-PMH is handled at the HTTP level, with the added restriction that both repositories and harvesters must support the `identity` encoding (no compression). Repositories **should** advertise the compression they support in `compression` element(s) in the `Identify` response.

Response from `Identify`:

```
<Identify ...>
  ...
  <compression>gzip</compression>
  ...
</Identify>
```

Next request includes:

```
Accept-encoding: gzip;q=1.0, identity;q=0.5
```

Response then `gzip` encoded with header:

```
Content-Encoding: gzip
```

While compression is optional, it is encouraged to minimize network traffic involved in lengthy harvests. It is recommended that repositories offer at least one of the compression types defined in HTTP: RFC 2616 section 3.5 "Content Codings": `gzip`, `compress`, or `deflate`.

8. Error Handling

Errors are reported using one or more `error` elements in the response. Although not required, it is strongly recommended that repository implementers include detailed and helpful error messages (content of the `error` elements) in addition to the mandatory `code` attributes.

Repositories are permitted to include just a single `error` element even if multiple error conditions are identified. However, to aid debugging, it is recommended that repositories report as many errors as they identify. Multiple `error` elements may be used and it is preferable to separate even multiple errors with the same `code` into separate `error` elements. For example,

```
  <error code="badArgument">Illegal argument 'arg1'</error>
  <error code="badArgument">Illegal argument 'arg2'</error>
```

is to be preferred over

```
  <error code="badArgument">Illegal arguments 'arg1', 'arg2'</error>
```

Acknowledgements

Support for the development of the OAI-PMH and for other Open Archives Initiative activities comes from the Digital Library Federation, the Coalition for Networked Information, and from the National Science Foundation through Grant No. IIS-9817416. Individuals who have played a significant role in the development of OAI-PMH version 2.0 are acknowledged in the protocol document.

Document History

2005-01-19: HTML fixes and added Table of Contents.
2002-05-13: Changed to reflect day/second granularities in protocol.
2002-03-31: Release of initial version of OAI-PMH v2.0 guidelines documents.