Connector API

Authors

Person	Role	Partner	Contribution
Frank Asseg	DEV	FIZ	
Matthias Hahn	DEV	FIZ	

Distribution

Person	Role	Partner
PT Group		

Revision History

Version	Status	Author	Date	Changes
0.9		Frank Asseg	2012-04-10	Final draft
1.0		Matthias Hahn	2012-04-19	
1.1		Frank Asseg	2013-7-02	Added parent ids

Table of Contents

1 Introduction

The Connector API's purpose in the SCAPE platform is to integrate different repositories with the various SCAPE components. As such it sits on top of the content repository and exposes well defined services via HTTP used by clients to access the repository content. The proposed Connector API can be used by clients to access not only content but preservation plans as well.

When preserving collections with mainly small individual content, the overhead to request binary content via HTTP can be neglected, but when requesting large binary content from the repository via HTTP the overhead gets significant in respect to the request duration. Therefore two storage strategies are introduced that handle binary content differently: One is responsible for the whole lifecycle of a binary object, the second storage strategy only handles references to binary content, while an outside agent is responsible for content manipulation.

The Connector API is required to acommodate the various use cases in a SCAPE platform arising from the different API clients. The computation cluster as a client of the Connector API must be able to perform CRUD[1] operations on digital objects, while preservation plan management requires CRUD operations on preservation plans, and the discorvery of digital objects via SRU[2] searches is required by e.g. plan experimenting.

The Connector API is, aside from the Report API, one of the two APIs a repository has to implement in order to be employed in a SCAPE platform. Therefore arbitrary repository systems can be used in a SCAPE environment, if and only if they implement the Connector and the Report API.

Since well defined requests have to be used by clients of the Connector API the format of the objects is part of the API definitions. The data model definitions for

[1] http://en.wikipedia.org/wiki/Create,_read,_update_and_delete

[2] http://www.loc.gov/standards/sru/

the SCAPE platform can be found in the document “SCAPE Digital Object Model” [3]by the PT.WP.5 work package members. This document introduces Intellectual Entities, Representations, Files and Bitstreams as the core elements of the SCAPE data model.

2 Use Cases of the Connector API

2.1 Loader application for batch ingest

The Loader Application will be a client application of the repository that enables the administrator to ingest data into the repository in a batch process. The Loader application takes care of validation, error logging and retry functionality. The details of the requirements will be described in a separate document. The loader application requires a HTTP endpoint for ingesting Intellectual Entities in the repository as described in Chapter 5.4.4.

2.2 Request Intellectual Entities by the computation cluster

In order to run actions on Intellectual Entities for characterization, identification and other tasks and depending on the Storage Strategy as described in Chapter 3 the data or their references are needed for task execution. References to the data can be extracted from Intellectual Entities requested as described in Chapter 5.4.1. Content can also be requested directly from the repository by using an Identifier as described in Chapter 5.4.8 and depending on the Storage Strategy the repository returns the content directly or uses HTTP redirection to route the request to the content's location.

2.3 Update Intellectual Entities with provenance information

To enable preservation actions executed on the computation cluster updating provenance metadata of Representations an interface is needed. Preservation actions have to employ the HTTP endpoint to update Intellectual Entities, although it may be possible to expose an endpoint for updating single representations in the future. If e.g. the computation cluster updated an image file and provenance information had to be written into the corresponding Representation, the information had to be updated in the repository by using the HTTP endpoint to update the whole Intellectual Entities as described in Chapter 5.4.5.

2.4 Partially fetching large scale files

A repository may hold Files such as ARC containers large in size. Since preservation tasks may only be concerned with a subset of the File, fetching the whole container would be uneconomic and have negative impact on performance. Therefore the requirement to partially request Files arises. In order to request partial Files from the repository the endpoint as described in Chapter Error: Reference source not found can be used or in the case of named bit streams the endpoint as described in Chapter 5.4.9.

---

[3] https://portal.ait.ac.at/sites/Scape/PT/Shared%20Documents/PT.WP.5%20Repository%20Integration/SCAPE_Dgital_Object_Model.doc

3 Storage Strategy

The heterogeneity of collections in regard to size poses a challenge for a well performing implementation: while fetching small collections through an API exposed via HTTP for
workflow execution is a practical approach, the performance to fetch very large collections will be poor. Therefore the following two Storrage Strategies are available, letting stakeholders choose the more fitting alterantive:

1. *Managed Content:* When using this strategy Files are accessible only via the Connector API, so that any operations involving Files the binary data requires the clients to use the Connector API to retrieve or update the binary data from the repository. This functionality is exposed to the clients by an HTTP enpoint as described in Chapter 5. This strategy is not very well suited for large amounts of data or geographically separated storage and computation clusters, because of the necessary I/O overhead for data retrieval by the computation cluster..
2. *Referenced Content:* When using this strategy the repository is integrated with the different components by storing Files in a file system directly accessible by the SCAPE platform, and only referencing Files in the repository by an URI. This enables the platform to handle large files without having to move them in between machines before computation can take place.

The first approach enables preservation actions to be performed on an external computation cluster, which is in use for a limited time only. The export of the data to the cluster is performed via the Connector API. In this scenario the computation cluster and storage are physical and/or geographical distinct entities. Drawbacks to this approach are that an institutions policy might prevent exporting the data to an untrusted third party and it introduces a bottleneck by exporting the data over a network - a worst case scenario would be to move the entire content - to a computation cluster.
The second approach is more eligible for running a Hadoop cluster with access to the same HDFS file system the Files are stored in. Computation Cluster and Storage are not distinct instances in this scenario, requiring the hardware to act as a storage and computation platform for the preserved Intellectual Entities.

4 Schematic views

4.1 Fetching an Intellectual Entity

To fetch an Intellectual Entity from the repository via the Connector API the following workflow has been identified:

1. A taverna workflow[4] has been scheduled for execution on the computation cluster
2. A process on the computation cluster requests collection metadata from the repository via the Connector API

[4] http://www.taverna.org.uk/

3. The repository gathers the Intellectual Entities' metadata from the metadata storage
4. The repository answers the process' Connector API request by returning the XML representation for the Intellectual Entity.
5. If referenced content is used the computation cluster can access the binary data by resolving URIs in the collection profile, otherwise the computation cluster requests the binary data from the repository via the Connector API.

The following sequence diagram illustrates this workflow to retrieve an Intellectual Entity from the repository.

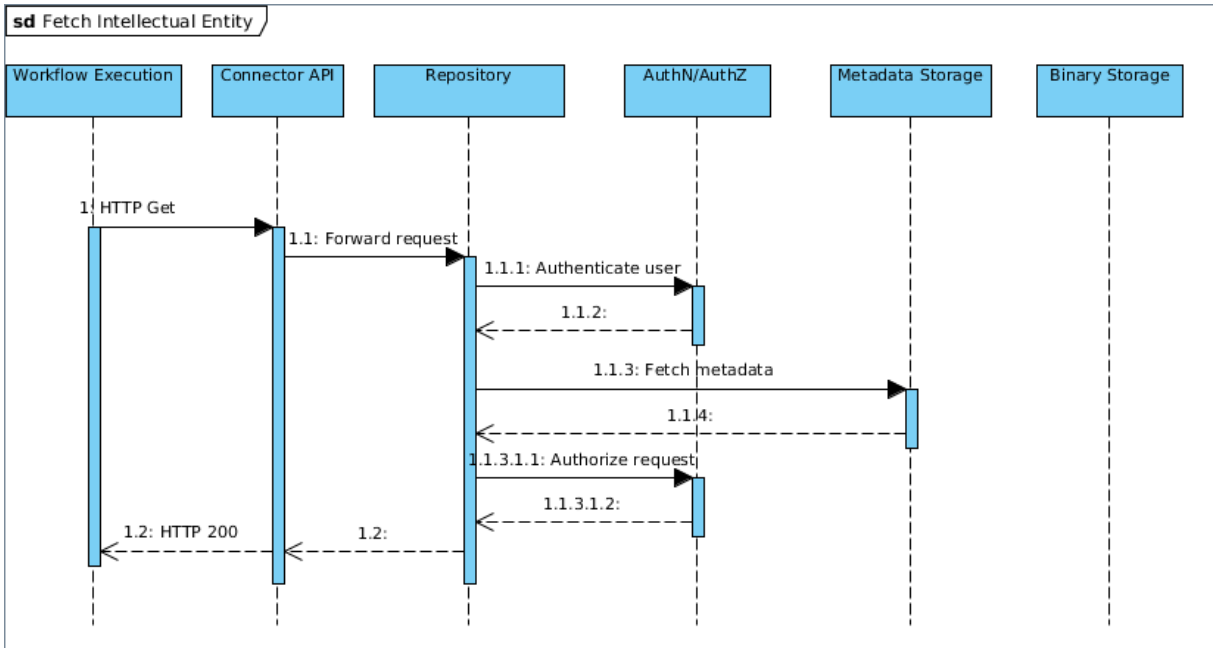

Figure 4-1: Sequence Diagram illustrating the workflow to fetch an Intellectual Entity

4.2 Fetching a File

This two sequence diagrams illustrate the workflow to fetch a File from the repository in a scenario where the files are handled as referenced content and in the second diagram within a managed content scenario.

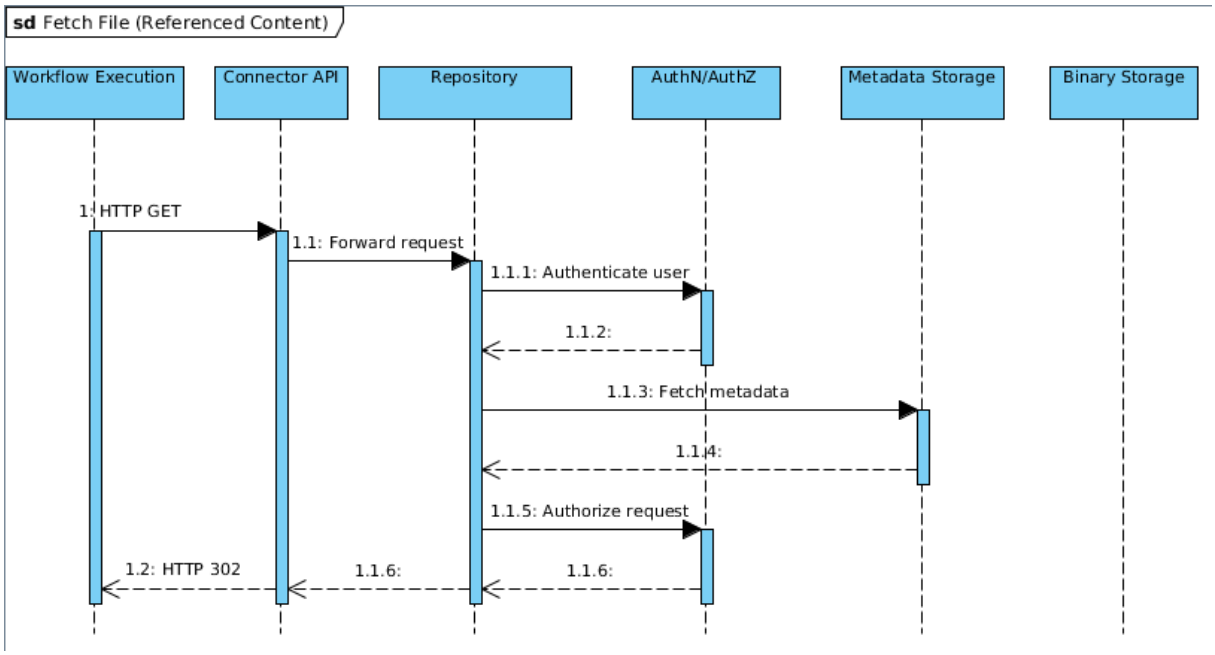

Figure 4-2: Sequence Diagram illustrating the workflow to fetch an File (as referenced content)

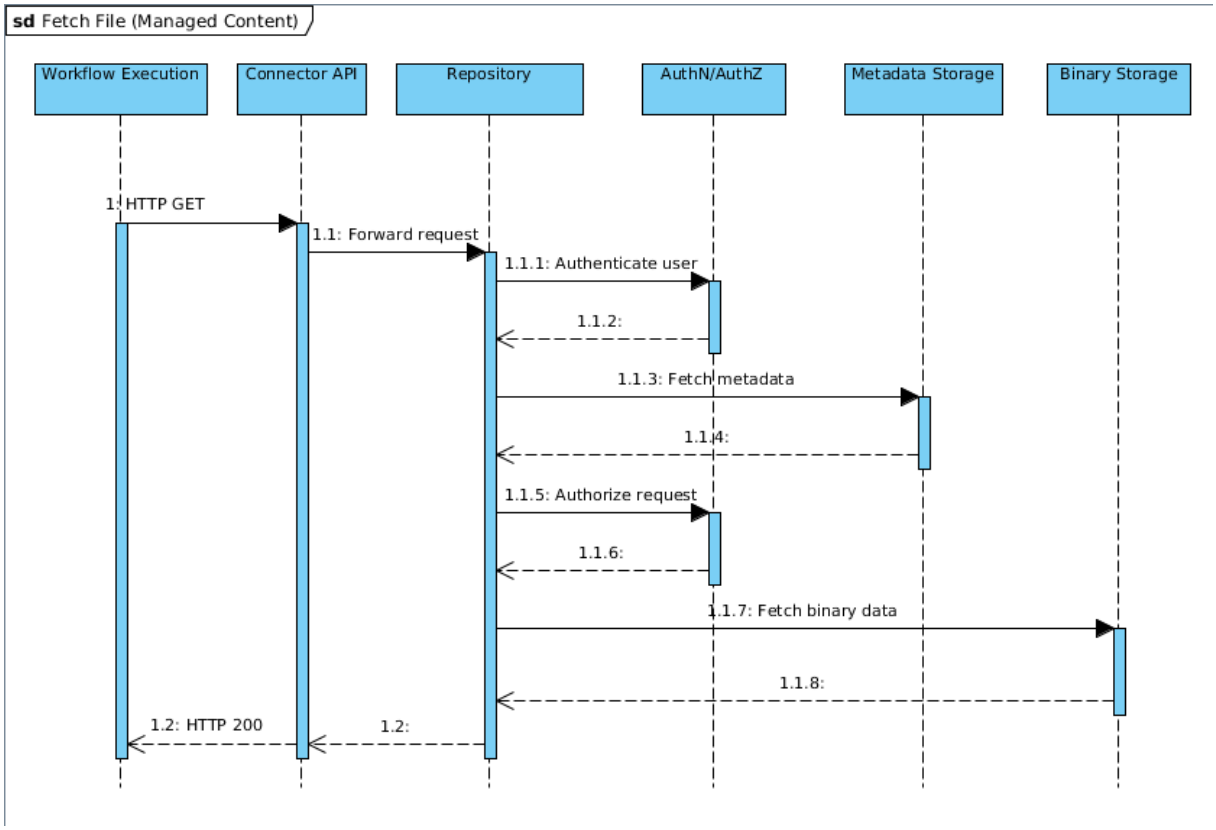

Figure 4-3: Sequence Diagram illustrating the workflow to fetch an Intellectual Entity (as managed content)

4.3 Ingesting or Updating an Intellectual Entity

To ingest or to update an Intellectual Entity in a repository via the Connector API the following workflow has been identified:

1. A Taverna workflow has executed successfully and the resulting Intellectual Entities have to be written back to the repository.
2. A process on the computation cluster sends the updated Intellectual Entity via the Data Connector API to the repository, and references the updated binary data in the request.
3. The repository updates the Intellectual Entities' metadata, and if using managed content the updated binary data gets written from the referenced location to the binary storage.
4. The repository answers the process' request and informs about any errors that might have occurred while the operation was performed.

The following sequence diagrams illustrates this workflow for two storage scenarios of managed and referenced content:

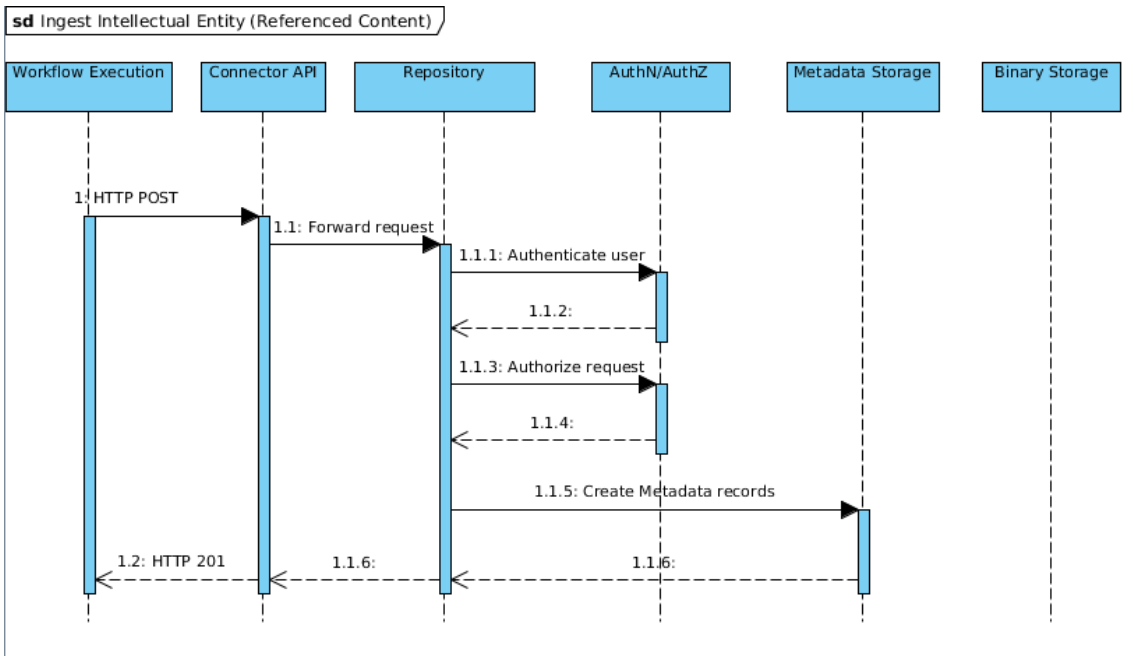

Figure 4-4: Sequence Diagram illustrating the workflow to ingest an Intellectual Entity (as referenced content)

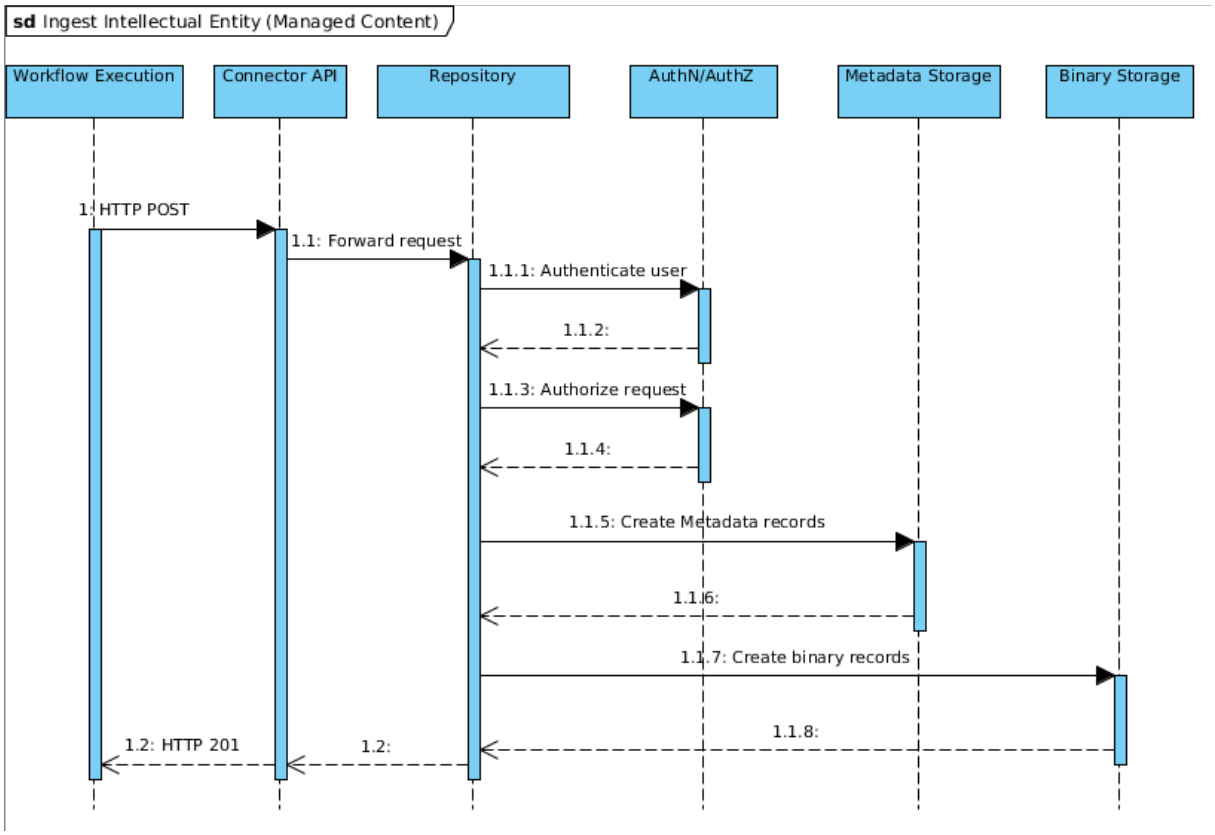

Figure 4-5: Sequence Diagram illustrating the workflow to ingest an Intellectual Entity (as managed content)

5 Specification

The content repository exposes a RESTful[5] API to the workflow execution layer on top of a Hadoop computation cluster, for operations on the metadata of Intellectual Entities. The API consists of various HTTP endpoints defined in the following section.

The preserved Intellectual Entities consist of administrative, structural, rights and descriptive metadata and associated Files and are themselves versioned. Existing and well established standards for encoding the different aspects of metadata are used for wrapping the metadata of Intellectual Entities.

[5]R.T. Fielding, 2000,"Architectural Styles and the Design of Network-based Software Architectures"

5.1 Authentication & Authorization

Following the REST recommendations authentication is done by using Basic and Digest Access Authentication mechanism, also known as HTTP Basic Authentication, which is well established and supported throughout the otherwise heterogeneous IT landscape. This implies that every HTTP request has to have a BASE64 encoded user name and password string in the Authentication Header. Therefore encryption by using HTTP over SSL/TLS is strongly recommended. Authorization is done by the repository depending on the current user and the individual Representations' associated rights and permission metadata.

5.2 Life cycle states

The life cycle state of an Intellectual Entity contains the information describing the state of an Intellectual Entity in the preservation lifecycle. Currently three states are defined: INGESTED, INGEST_FAILED, OTHER. Each of these states also has details associated with it, so the repository can supply additional information. These states result from the following use case: When ingesting SIPs asynchronously a user has to be able to get information about the ingestion process: Whether ingestion succeeded, failed or is currently in process. In order to supply information about currently running ingestion processes (e.g. virus checking) “OTHER” can be used with it’s details describing the process of the ingestion workflow the SIP is currently in.

```
<lifecyclestate id=”entity-42” state=”INGESTED”>
      <details>Ingest finished successfully at 6/27/2012 13:34:57</details>
</lifecyclestate>
```

Example 1: XML Representation of the life cycle state of an ingested Intellectual Entity

5.3 HTTP Status codes

The existing HTTP status codes with their individual semantics are used in the context of the connector API.

- **200 OK** This indicates success.
- **201 Created** means that an object has been created in the repository.
- **401 Unauthorized** The request requires authentication. The user should authenticate properly against the repository and repeat the request.
- **403 Forbidden** The server refuses to fulfill the request. Authorization will not help and the request should not be repeated.
- **404 Not Found** The requested resource cannot be found.
- **415 Unsupported Media Type** The media type sent with the requested was not valid**.**
- **500 Internal Server Error** In the case of runtime errors that might be happening while requesting a resource: e.g. Disk full.

5.4 HTTP endpoints

Following is a specification of the HTTP endpoints. The implementation of the endpoints has to be done by each repository in order to be deployed in a SCAPE Platform environment.

The XML Schemas of the XML responses are defined by the Scape Digital Object Model and by metadata frameworks like Dublin Core and VideoMD. The schemas for object representations as defined in the Digital Object Model are generated by the JAX-B implementation’s schemagen tool. The schemas for existing

metadata frameworks employed by the SCAPE platform are readily available on the web.

5.4.1 Retrieve an Intellectual Entity

Retrieval of entities is done via a GET request. Since Intellectual Entities can have multiple versions there is an optional version identifier, which when omitted defaults to the most current version of the Intellectual Entity. When successful the response body is a METS representation of the Intellectual Entity. The parameter *useReferences* controls wether the response is created using references to the metadata via <mdRef> elements or if the metadata should be wrapped inside <mdWrap> elements in the METS document.

Path:

/entity/<entity-id>/<version-id>?useReferences=[yes|no]

Method:

HTTP/1.1 GET

Parameters:

entity-id: *the id of the requested Intellectual Entity*

version-id: t*he version of the requested entity (optional)*

useReferences: *Wether to wrap metadata inside <mdWrap> elements or to reference the metadata using<mdref> elements. Defaults to yes.*

Produces:

A XML representation of the requested Intellectual Entity version

Content-Type:

text/xml

5.4.2 Retrieve a metadata record

Retrieval of single metadata records of entities is done via a GET request. Since Intellectual Entities can have multiple versions there is an optional version identifier, which when omitted defaults to the most current version of the Intellectual Entity. When successful the response body is a XML representation of the corresponding metadata record.

Path:

/metadata/<entity-id>/<rep-id>/<file-id>/<bitstream-id>/<version-id>/<md-id>

Method:

HTTP/1.1 GET

Parameters:

entity-id: *the id of the Intellectual Entity*

rep-id: *the id of the Representation (optional)*

file-id: *the id of the File (optional)*

bitstream-id: *the id of the requested binary content (optional)*

md-id: the id of the metadata to retrieve

version-id: t*he version of the requested bit stream's parent Intellectual Entity (optional)*

Produces:

A XML representation of the requested metadata record according to the corresponding metadata’s schema

Content-Type:

text/xml

5.4.3 Retrieve a set of Intellectual Entities

In order to make fetching a whole set of entities feasible this GET method consumes a list of URIs sent with the request. It resolves the URIs to Intellectual Entities and creates a response consisting of the corresponding METS representations. If at least one URI could not be resolved the implementation returns a HTTP 404 Not Found status message.

Path:

/entity-list

Method:

HTTP/1.1 POST

Consumes:

A text/uri-list of the entities to be retrieved

Produces:

METS representations of the requested entities.

Content-Type:

multipart

5.4.4 Ingest an Intellectual Entity

Ingestion of digital objects is done by sending a METS representation of an Intellectual Entity in the body of a HTTP POST request, which gets validated and persisted in the repository. If validation does not succeed the implementation returns a HTTP 415 “Unsupported Media Type” status message. When successful the response body is a plain text document consisting of the ingested entity's identifier.

Path

/entity

Method

HTTP/1.1 POST

Parameters

Consumes

A XML representation of the entity

Produces

The Intellectual Entity identifier

Content-Type

text/plain

5.4.5 Ingest an Intellectual Entity asynchronously

Ingestion is done by sending a SIP to this endpoint. The method returns instantly and supplies the User with an ID which can be used to request the status of the ingestion.

Path

/entity-async

Method

HTTP/1.1 POST

Parameters

Consumes

A XML representation of the entity

Produces

An Identifier which can be used to request the lifecycle status of the digital object ingested.

Content-Type

text/plain

5.4.6 Update an Intellectual Entity

In order to allow updating of Intellectual Entities the implementation exposes this HTTP PUT endpoint. The mandatory parameter <id> tells the repository which Intellectual Entity is to be updated. The request must include the updated METS representation of the entity in the request body.

Path:

/entity/<id>

Parameters

Id: *the id of the Intellectual Entity to update*

Method

HTTP/1.1 PUT

Consumes

A digital object's XML representation.

5.4.7 Retrieve a version list for an Intellectual Entity

In order to get a list of all versions of an Intellectual Entity a plain GET request can be sent to the implementation with the <id> parameter indicating which entity's versions to list. If successful the response consists of the Intellectual Entity's version identifiers in a XML representation

Path:

/entity-version-list/<entity-id>

Parameters

entity-id: *the id of the Intellectual Entity*

Method

HTTP/1.1 GET

Produces

A XML representation of all the entities version ids.

Content-Type
text/xml

5.4.8 Retrieve a File

For fetching Files associated with Intellectual Entities the implementation exposes a HTTP GET endpoint. Requests sent to this endpoint must have a <id> parameter indicating which File to fetch. The parameter <version-id> indicating the version to fetch is optional and defaults to the most current version of the File. Depending on the Storage Strategy the response body is the binary file with the corresponding Content-Type set by the repository or a HTTP 302 redirect in the case of referenced content.

Path:

/file/<entity-id>/<representation-id>/<file-id>/<version-id>

Method

HTTP/1.1 GET

Parameters

entity-id: *the id of the Intellectual Entity*

representation-id: *the id of the Representation*

file-id: *the id of the File*

version-id: t*he version of the requested File's parent Intellectual Entity (optional)*

Produces

the file requested or a redirect to the file when using referenced content.

Content-Type

depends on File's metadata, but defaults to application/octet-stream.

5.4.9 Retrieve named bit streams

For fetching a named subset of Files, such as an entry in an ARC container, the implementation exposes a HTTP GET method. The mandatory parameter <id> is the identifier of the requested bit stream in the Intellectual Entity. Depending on the Storage Strategy the implementation returns the bit stream directly in the response body, or it redirects the request using HTTP 302 to the referenced content. This requires special care when using Referenced Content as a Storage Strategy since the implementation is only able to redirect to referenced bit streams, making the redirect target responsible for answering the request properly.

Path:

/bitstream/<entity-id>/<rep-id>/<file-id>/<bitstream-id>/<version-id>

Method

HTTP/1.1 GET

Parameters

entity-id: *the id of the Intellectual Entity*

rep-id: *the id of the Representation*

file-id: *the id of the File*

bitstream-id: *the id of the requested binary content*

version-id: t*he version of the requested bit stream's parent Intellectual Entity (optional)*

Produces

the binary content associated requested or a redirect to the binary content.

Content-Type

depends on content's type, but defaults to application/octet-stream.

5.4.10 Search Intellectual Entities in a collection

For digital object discovery the implementation exposes a SRU search endpoint. The endpoint implements the SRU specifications by the Library of Congress for Internet Search queries, utilizing CQL, a standard syntax for representing queries, and exposes this functionality via a HTTP GET endpoint. Pagination is done via the SRU parameters *startRecord* and *maximumRecords*[6]

Path

/sru/entities

Parameters

see SRU specification[7]

Method

HTTP/1.1 GET

Produces

A XML representation as specified by SRU

Content-Type

text/xml

5.4.11 Search Representations in a collection

For discovering Representations the implementation exposes a SRU search endpoint. The endpoint implements the SRU specifications by the Library of

---

6 http://www.loc.gov/standards/sru/specs/search-retrieve.html

7 http://www.loc.gov/standards/sru/

Congress for Internet Search queries, utilizing CQL, a standard syntax for representing queries, and exposes this functionality via a HTTP GET endpoint. Pagination is done via the SRU parameters *startRecord* and *maximumRecords*

Path

/sru/representations

Parameters

see SRU specification

Method

HTTP/1.1 GET

Produces

A XML representation as specified by SRU

Content-Type

text/xml

5.4.12 Search Files in a collection

For discovering Files the implementation exposes a SRU search endpoint. The endpoint implements the SRU specifications by the Library of Congress for Internet Search queries, utilizing CQL, a standard syntax for representing queries, and exposes this functionality via a HTTP GET endpoint. Pagination is done via the SRU parameters *startRecord* and *maximumRecords*

Path

/sru/files

Parameters

see SRU specification

Method

HTTP/1.1 GET

Produces

A XML representation as specified by SRU

Content-Type

text/xml

5.4.13 Retrieve the lifecycle status of an entity

In order to access the life cycle state of an Intellectual Entity without having to fetch the whole METS representation an endpoint for retrieving this significant property is exposed by the repository.

Path

/lifecycle/<entity-id>

Method

HTTP/1.1 GET

Parameters:

entity-id: *the id of the Intellectual Entity to update*

Produces

A XML representation of the lifecycle status

Content-Type

text/xml

5.4.14 Retrieve a Representation

For fetching Representations without having to retrieve the METS representation of the whole Intellectual Entity a dedicated endpoint is exposed by the repository

Path:

/representation/<entity-id>/<representation-id>

Method:

HTTP/1.1 GET

Parameters:

entity-id: *the id of the Intellectual Entity to update*

representation-id: *the id of the Representation to update*

Produces:

A XML representation of the requested Representation

Content-Type:

text/xml

5.4.15 Update a Representation of an Intellectual Entity

For updating a Representation of an Intellectual entity without sending a METS representation of the Intellectual Entity an endpoint is exposed by the repository. The repository *has to* create a new Version of the Intellectual Entity with the updated Representation.

Path:

/representation/<entity-id>/<representation-id>

Parameters

entity-id: *the id of the Intellectual Entity to update*

representation-id: *the id of the Representation to update*

Method

HTTP/1.1 PUT

Consumes

A Representations' XML representation.

5.4.16 Update the metadata of an Intellectual Entity

When updating only the metadata of an Intellectual Entity validity on binary files can be omitted, thereby saving cpu cycles. An endpoint is exposed to clients for updating the metadata of an Intellectual entity, that consumes a METS representations of an Intellectual Entity.

Path:

/metadata/<entity-id>/<metadata-id>

Parameters

entity-id: *the id of the Intellectual Entity to update*

metadata-id: *the Id of the metadata set to update*

Method

HTTP/1.1 PUT

Consumes

An metadata's XML representation.

6 Technology Compatibility Kit

A Technology Compatibility Kit will be developed to enable repository system developers to test their repository's implementation of the Connector API. The TCK consists of a series of integration tests covering the various endpoints of the Connector API, mocking an implementation client, creating, retrieving and updating Intellectual Entities and preservation plans in the repository as described by the various HTTP methods in Chapter 5.4. The TCK should test for the various error conditions possible and validate the responses of the implementation.

The TCK will consist of a HTTP Client sending requests to an existing repository implementation. It will for example check if it is possible to ingest an Intellectual Entity via a HTTP POST as described in 5.3.3 and check if the ingested entity can be fetched by the HTTP GET method described in 5.3.1.

7 Glossary

Apache Hadoop is a software framework that supports data-intensive distributed applications under a free license. It enables applications to work with thousands of computational independent computers and petabytes of data. Hadoop was derived from Google's MapReduce and Google File System (GFS) papers.

A **Bitstream** is a contiguous or non-contiguous data within a file that has meaningful common properties for preservation purposes. Generally speaking, a bitstream cannot be transformed into a standalone file without the addition of file structure (headers, etc.) and/or reformatting the bitstream to comply with some particular file format. This definition is derived from the data model outlined in Introduction and Supporting Materials from PREMIS Data Dictionary, p. 7, illustrated by the example of a TIFF file that contains embedded bitstreams representing raster images together with header that presents some information about the file. The authors of the PREMIS definition note that their definition is limited to sets of bits embedded within a file and they call attention to an alternate usage that defines *bitstream* as an entity that could span more than one file.[8]

[8] http://www.digitizationguidelines.gov/term.php?term=bitstream

A **File** file is a named and ordered sequence of bytes that is known by an operating system. A file can be zero or more bytes and has a file format, access permissions, and file system characteristics such as size and last modification date . . . . Files can be read, written, and copied. Files have names and formats." Introduction and Supporting Materials from PREMIS Data Dictionary (p. 7)[9]

A **Intellectual entity** is a set of content that is considered a single intellectual unit for purposes of management and description: for example, a particular book, map, photograph, or database. An Intellectual Entity can include other Intellectual Entities; for example, a Web site can include a Web page; a Web page can include an image. An Intellectual Entity may have one or more digital representations. From Introduction and Supporting Materials from PREMIS Data Dictionary, p. 6.[10]

A **Representation** is a set of files, including structural metadata, needed for a complete and reasonable rendition of an Intellectual Entity. For example, a journal article may be complete in one PDF file; this single file constitutes the representation. Another journal article may consist of one SGML file and two image files; these three files constitute the representation. A third article may be represented by one TIFF image for each of 12 pages plus an XML file of structural metadata showing the order of the pages; these 13 files constitute the representation. From Introduction and Supporting Materials from PREMIS Data Dictionary, p. 7.[11]

A **Technology Compatibility Kit (TCK)** is a suite of tests that at least nominally checks a particular alleged implementation for compliance.

The **Metadata Encoding and Transmission Standard (METS)** is a metadata standard for encoding descriptive, administrative, and structural metadata regarding objects within a digital library, expressed using the XML schema language of the World Wide Web Consortium. The standard is maintained in the Network Development and MARC Standards Office of the Library of Congress, and is being developed as an initiative of the Digital Library Federation.

Representational state transfer (REST) is a style of software architecture for distributed hypermedia systems such as theWorld Wide Web. The term

[9] http://www.digitizationguidelines.gov/term.php?term=digitalfile

[10] http://www.digitizationguidelines.gov/term.php?term=intellectualentity

[11] http://www.digitizationguidelines.gov/term.php?term=representation

representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. Fielding is one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification versions 1.0 and 1.1.

Contextual Query Language (CQL), previously known as Common Query Language is a formal language for representing queries to information retrieval systems such as search engines, bibliographic catalogs and museum collection information. Based on the semantics of Z39.50, its design objective is that queries be human readable and writable, and that the language be intuitive while maintaining the expressiveness of more complex query languages. It is being developed and maintained by the Z39.50 Maintenance Agency, part of the Library of Congress.

Search/Retrieve via URL(SRU) is a standard search protocol for Internet search queries, utilizing Contextual Query Language(CQL), a standard query syntax for representing queries.

8 List of Figures

9 Changes

Update 1.1

In order to minimize requests needed to the back end store (e.g. Fedora) the Identifiers of the parent have been added to the request paths. For e.g. retrieval of a File the corresponding Representation and Intellectual Entity Identifiers are supplied via path variables.