Reference implementation of DOR with interfaces to preservation components, workflows, and execution

Authors
Matthias Hahn, Frank Asseg (FIZ Karlsruhe), Sebastien Leroux (KEEP Solutions), Donal Fellows (University of Manchester)

September 2014

This work was partially supported by the SCAPE Project. The SCAPE project is co-funded by the European Union under FP7 ICT-2009.4.1 (Grant Agreement number 270137).

Executive Summary

The SCAPE ecosystem presents a modular infrastructure divided across five main entities, Automated Watch, Automated Planning, Component Management, the Execution Platform and the Digital Object Repository. The Digital Object Repository integrates with almost all aforementioned entities. The SCAPE project delivered two repositories, Fedora 4 and RODA, as reference implementations for the SCAPE ecosystem. These integrate with the SCAPE tools and services via SCAPE APIs such as the Preservation Watch Service, SCOUT, the Preservation Planning Tool, PLATO, the SCAPE Loader Application and the Plan Management GUI. This document describes the two repositories from a technical perspective and how they integrate with the SCAPE tools and services.

On the one hand SCAPE took part in the Fedora 4 development and implemented the SCAPE APIs on top of this well-known repository – although Fedora 4 is in a beta stage at the time of this writing. On the other hand RODA [SW6], developed by KEEPS, becomes a reference implementation due to its tied integration with Preservation Planning and Watch and it is a mature repository based on a former release of Fedora. Both repositories, RODA and Fedora are open source are able to interact with several SCAPE services like Scout [SW1], PLATO [SW3] or the Workflow Execution Engine based on Taverna Server [SW9] via the Plan Management GUI (to manage and execute Preservation Plans). A detailed description of these tools and services is not part of this document but we provide a short overview in section 2. The following sections describe the installation and usage of RODA and Fedora 4 along with other information we think is worthwhile knowing in order to get started with the Digital Object Repository for the SCAPE ecosystem.

Table of Contents

1 Introduction

This document describes the reference implementations of the Digital Object Repository for the SCAPE ecosystem. The repositories that have been used within the SCAPE project are RODA, eSciDoc, Fedora 4, DOMS and Rosetta. Despite Rosetta, which is a commercial implementation of a repository used for long term preservation, the other aforementioned repositories are based on Fedora, which is a well-known open source repository maintained and developed by DuraSpace [1]. The repositories are not built on top of the same version of Fedora however; RODA is based on Fedora 2, whereas DOMS and eSciDoc are based on Fedora 3. During the project's lifetime the development of Fedora 4 started with the promise of better scalability than Fedora 3. An alpha release of Fedora 4 has been chosen for this reason as one reference implementation for SCAPE.

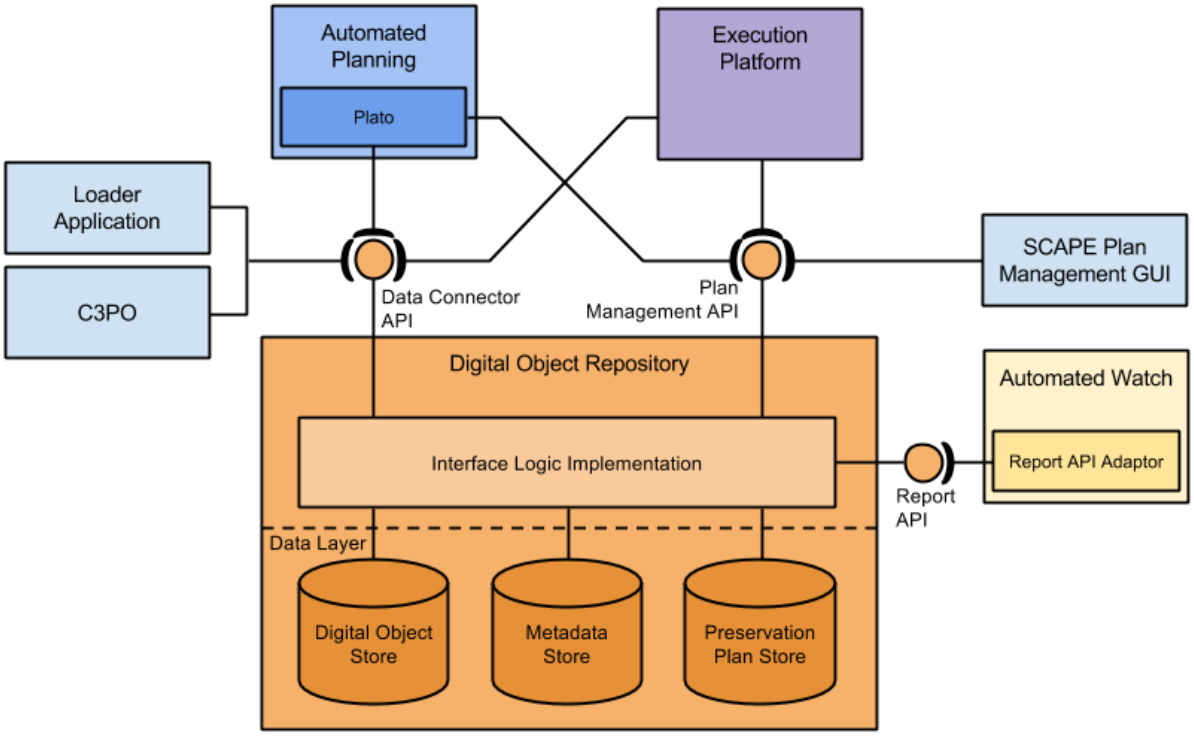

Figure 1-1: Digital Object Repository architecture and interactions

As illustrated in figure 1.1, the SCAPE ecosystem consists of several services such as the Execution Platform, Planning and Watch Services (C3PO [SW2], Automated Watch [SW1] and Plato [SW3]) and the Digital Object Repository. The DOR Interface Logic is an abstract entity that contains repository-specific implementations of the necessary logic to enable the functionality provided by three APIs outlined below. The three conceptual stores in the data layer, Digital Object Store, Metadata Store and Preservation Plan Store, do not have to be separate stores, and serve to illustrate that the Digital Object Repository stores three different types of information:

- Digital Object Store: This conceptual entity is responsible for storing and making accessible digital object content.

- Metadata Store: This conceptual entity is responsible for storing and making accessible metadata information relating to the digital objects stored in the Digital Object Store.

- Preservation Plan Store: This conceptual entity is responsible for storing Preservation Plans executable on the SCAPE Execution Platform.

To integrate a repository with these services three APIs have been specified and implemented. The APIs are described in detail and the specification is publicly available on Github [17].

In short, the following APIs have been defined:

Data Connector API: Interface to create, retrieve, search and update digital objects within a repository. To implement the Data connector API, knowledge about the SCAPE Digital Object Model [SD5] is also needed (see Section 3.1).

Report API: Interface to retrieve information about events that take place on a repository, e.g. ingest, access, and preservation operations. A reference implementation of the Report API is available for the RODA repository [SW7].

Plan Management API: Interface to manage and execute preservation plans. The implementation of the Plan Management API can use a Workflow engine to actively perform preservation operations as defined by a preservation action plan.

Section 2 provides a short overview of the tools and services able to interact with the SCAPE Digital Object Repository via these APIs. This is followed, in section 3, with a description of the installation and usage of RODA and Fedora 4 along with other information we think is worthwhile knowing in order to get started with the Digital Object Repository.

2 SCAPE Tools and Services that interact with the DOR

The following tools and services are used in the SCAPE ecosystem, and interact with the SCAPE Digital Object Repository via the aforementioned SCAPE APIs. A detailed description can be found in the Architectural Report [SD1] .

Automated Watch – Scout
Scout [SW1] is an implementation for monitoring and reasoning over sources of information to detect preservation risks and opportunities. Scout implements a Report API Adaptor that interacts with the DOR Report API. The adapter provides the ability to gather and record user interaction data from a repository, e.g. ingest events, user access events, or plan execution events. Scout is a Java Web Application and can be deployed on a Java Servlet Container like Tomcat.

Content Profiling – C3PO
A content profiling tool [SW2] that processes metadata extracted from digital objects into an aggregated form suitable for Preservation Planning and Watch analysis.

Automated Planning – Plato
Plato [SW3] is a tool applying a methodical approach to generating, testing and evaluating

preservation plans. Plato is a JBOSS application packaged as Java Enterprise Application Resource.

Ingest – Loader Application
The Loader Application [SW11] provides a way to ingest and monitor the ingest process of data into the digital object repository.

Workflow Execution – Taverna Server
A system to execute a Taverna [SW9] workflow. This system is used to run a preservation action plan and works together with the Plan Management GUI to retrieve a preservation plan from the repository.

Plan Management GUI
To interact with the Plan Management API and to execute a preservation plan a graphical user interface [SW10] has been developed and can be used by any repository that exposes the Plan Management API. Appendix A of this document describes the installation and configuration of the Plan Management GUI.

The following figure illustrates the interactions between the SCAPE tools and services that directly interact with the DOR described above.

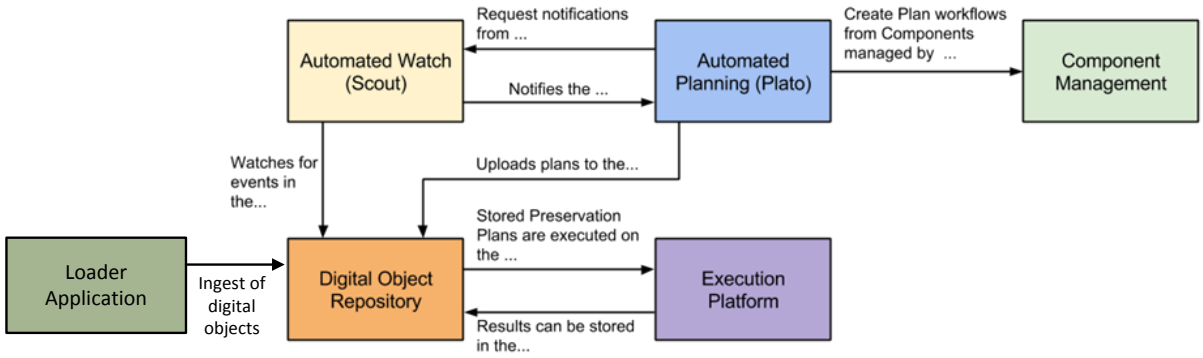

Figure 2-1: Effective interactions between SCAPE entities

3 The SCAPE Repository Reference Implementations

Four digital object repositories, Fedora 4, RODA, DOMS and Rosetta are part of the SCAPE project. RODA and Fedora 4 are the reference implementations for SCAPE. RODA has implemented all three SCAPE APIs and provides therefore a full integration into the whole ecosystem. Fedora 4 currently provides only the Data Connector API and the Plan Management API, the implementation of the Report API is at the time of this writing in progress. Rosetta and DOMS expose only the Data Connector API, so the whole Preservation Planning & Watch services of SCAPE cannot be used. A summary of the implementation status of the APIs of all repositories are illustrated in the following table:

Repository	Connector API	Plan Management API	Report API
RODA	done	done	Done
Fedora 4	done	done	-[1]
Rosetta[2]	done	-	-
DOMS[3]	done	-	Planned

Table 1: Implementation of the SCAPE APIs of the SCAPE repositories

3.1 Digital Object Model

The SCAPE Digital Object Model [SD5] provides a unified model, based on PREMIS [14] and METS [13], enabling any repository's Data Connector API implementation to consume and deliver information in this format with other entities in the SCAPE ecosystem. Any information about digital objects should therefore be described with the SCAPE Digital Object Model.
An Object in PREMIS has three subtypes: Representation, File and Bitstream.

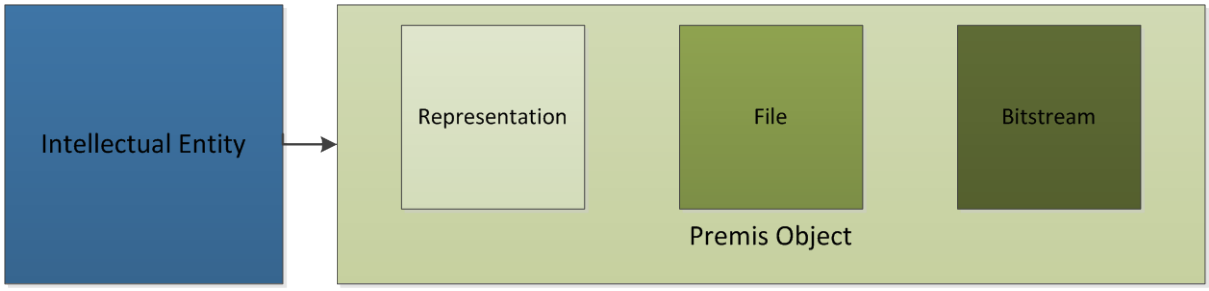

Figure 3-2: The PREMIS Data Model with Intellectual Entity and the Object with its subtypes: Representation, File and Bitstream.

The Intellectual Entity is, e.g., a book, a map, photograph or database, etc. The Representation of an IE is a set of Files including structural Metadata (e.g. described by METS). There can be more than one representation for the same Intellectual Entity. For example, a journal article may be complete in one PDF file and this single file will then constitute the representation. However, another journal article may consist of one SGML file and two image files. In this case, these three files will constitute the representation. A File is a named and ordered sequence of bytes known by the operating system that can be written, read and copied. Bitstreams represent data within a file, e.g., a jpeg in a PDF document, audio data within a WAVE file or graphics within a Word document. In the outline of this document we will refer to this model and we will use these terms in the section about the usage of the APIs.

A Java implementation of the SCAPE Digital Object Model has been developed to help developers serialize and de-serialize Intellectual Entities to and from METS [SW8]. This may help developers to implement the Data Connector API on top of their own repositories.

In the following section we will focus on Fedora 4 and RODA in more detail.

[1] The implementation of the Report API is, at the time of writing, in progress
[2] Rosetta is developed by ExLibris: http://www.exlibrisgroup.com/category/RosettaOverview
[3] DOMS is an institutional repository implementation by the Statsbiblioteket Denmark, Aarhus.

3.2 Fedora 4

Fedora is a robust, modular, open source repository system for the management and dissemination of digital content. It is especially suited for digital libraries and archives, both for access and preservation. It is also used to provide specialized access to very large and complex digital collections of historic and cultural materials as well as scientific data. Fedora has a worldwide installed user base that includes academic and cultural heritage organizations, universities, research institutions, university libraries, national libraries, and government agencies. The Fedora community is supported by the stewardship of the DuraSpace organization [1].

The decision to use Fedora 4 within SCAPE in addition to the other open source repositories based on Fedora 3 and 2 has been motivated by the clustering feature. However, currently the state of the clustering feature is not sufficient to run a performant distributed Fedora 4 Repository although the single installation performance competes with Fedora 3. This feature has been postponed by the Fedora Steering Group [2] and will not be achieved by the end of the SCAPE project life. Additionally, at the time of writing, the OAI-PMH implementation [21] is not finished and impacts on the implementation of the Report API, but work is in progress. The other SCAPE APIs, such as the Data Connector API and the Plan Management API, have been implemented and their usage is described in this section.

Additional information can be found at:

- Wiki https://wiki.duraspace.org/display/FF/Fedora+Repository+Home
- Roadmap https://wiki.duraspace.org/display/FF/Roadmap
- Google Group https://groups.google.com/forum/?hl=de#!forum/fedora-tech

The SCAPE specific API implementations can be found at:

- Fedora 4 Connector API implementation:
  https://github.com/openplanets/scape-fcrepo4-connector
- Fedora 4 Plan Management API implementation:
  https://github.com/openplanets/scape-planmanagement-webapp

3.2.1 Installation and Usage of Fedora 4

This section describes the usage and installation of Fedora 4. The war file of Fedora 4 with SCAPE APIs implemented can be downloaded from Github (see below). You need Tomcat 7 and a Java Runtime Environment based on Java 7 to be able to deploy Fedora 4 successfully.

3.2.1.1 Creating a WAR file from sources

Since Fedora 4 is in active development and therefore the APIs used are not yet finalized, this project might not run on an arbitrary version of Fedora 4. A version fcrepo-4.0.0-scape has therefore been tagged which is compatible with the current implementations:
https://github.com/futures/fcrepo4/tree/fcrepo-4.0.0-scape

3.2.1.2 Pre-packaged WAR

A pre-packaged Fedora 4 WAR file which includes the SCAPE Data Connector API and the SCAPE Plan Management API is available via http://scape.opf-labs.org/fedora.

Deploy Fedora 4 by dropping the war file into the webapps folder of the tomcat directory.

3.2.1.3 Usage of the Data Connector API

This section lists some examples of how to use the SCAPE Data Connector API with Fedora 4. Further examples can be found in the integration tests at [SW12].

The examples assume a local installation of Fedora 4, so the main URL "localhost" with port 8080 may be replaced by your specific configuration or setup of your web server, e.g. Apache. Part of the Data Connector API implementation is a file named "entity-minimal.xml" which can be found in the local source folder where you have checked out the source code of the Data Connector API implementation. This path is called *CONNECTOR_FOLDER* in the examples. The following examples are based on curl for simplicity to give you a hint of how to use the Connector API.

Ingest an Intellectual Entity
To ingest an intellectual entity a POST request will be consumed by the repository. In this example a minimal METS file, called "entity-minimal.xml" will be ingested. The file is part of the source code of the Connector API implementation.

```
$ curl -X POST http://localhost:8080/fcrepo/rest/scape/entity -d
@${CONNECTOR_FOLDER}/src/test/resources/entity-minimal.xml
```

Ingest an Intellectual Entity asynchronously
The "entity-async" POST method is useful to ingest Intellectual entities asynchronously into the repository. This method is useful if you want to ingest lots of files and you don't care when they will be stored persistently in the repository.

```
$ curl -X POST http://localhost:8080/fcrepo/rest/scape/entity-async -d
@${CONNECTOR_FOLDER}/src/test/resources/entity-minimal.xml
```

Retrieve an Intellectual Entity
To retrieve an Intellectual Entity a GET request with the ID of the Intellectual entity must be given. In this example "entity-1" is the ID. It depends on your repository which identifier can be used to retrieve an entity.

```
$ curl -X GET http://localhost:8080/fcrepo/rest/scape/entity/${entityID}
```

Retrieve a distinct Intellectual Entity version
It is possible to retrieve a distinct version of an Intellectual entity. If no version number is given, the most recent version will be delivered.

```
$ curl -X GET http://localhost:8080/fcrepo/rest/scape/entity/${entityID}/${version number}
```

Retrieve an Intellectual Entity using references for child elements
The parameter "useReferences" controls whether 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.

```
$ curl -X GET http://localhost:8080/fcrepo/rest/scape/entity/${entityID}?useReferences=[yes|no]
```

Retrieve an Intellectual Entity list

A list of Intellectual entities can be retrieved by sending a POST request with a list of entity IDs, e.g. entity-1. In this example only two references to entities ("entity-1" and "entity-2") in a list (as a separate file named "entity_list.txt") is sent to the repository.

```
$ curl -H "Content-Type:text/uri-list" -X POST http://localhost:8080/fcrepo/rest/scape/entity-list -d @${path to a directory}/entity_list.txt

Note entity_list.txt is a text file containing a list of links to objects:
http://localhost:8080/fcrepo/rest/scape/entity/entity-1
http://localhost:8080/fcrepo/rest/scape/entity/entity-2
```

Retrieve a Representation

To retrieve a representation only, and not the entire Intellectual entity, extend the GET request for the Intellectual entity with the ID of the representation. Again, the ID depends on your repository implementation.

```
$ curl -X GET http://localhost:8080/fcrepo/rest/scape/representation/${entityID}/${representationID}
```

Retrieve a binary File

To retrieve just a binary file as a part of an Intellectual entity and representation, just extend the GET request with the ID of the file.

```
$ curl -X GET http://localhost:8080/fcrepo/rest/scape/file/${entityID}/${representationID}/${fileID}
```

Retrieve a Bitstream

It is also possible to retrieve a bitstream (e.g. file inside a zip file), by extending the GET request with the bitstream ID (in this example "bitstream-1").

```
$ curl -X GET http://localhost:8080/fcrepo/rest/scape/bitstream/${entityID}/${representationID}/${fileID}/${bitstream ID}
```

Retrieve the lifecycle state of an Intellectual Entity

To retrieve the lifecycle state of an Intellectual entity a GET request with the ID of the intellectual entity is available. The following lifecycle states are defined: INGESTED, INGEST_FAILED, OTHER.

```
$ curl -X GET http://localhost:8080/fcrepo/rest/scape/lifecycle/${entityID}
```

Retrieve the descriptive metadata of an Intellectual entity

Retrieval of single metadata records of entities is done via a GET request. "entityID" is the ID of the entity and the term "DESCRIPTIVE" refers to the descriptive metadata set of the entity.

```
$ curl -X GET http://localhost:8080/fcrepo/rest/scape/metadata/${entityID}/DESCRIPTIVE
```

Retrieve the source/rights/provenance or technical metadata of a Representation
Source, rights, provenance and technical metadata of a representation of an entity can be retrieved via a GET request.

```
$ curl -X GET
http://localhost:8080/fcrepo/rest/scape/metadata/${entityID}/${representationID}/SOURCE

$ curl -X GET
http://localhost:8080/fcrepo/rest/scape/metadata/${entityID}/${representationID}/RIGHTS

$ curl -X GET
http://localhost:8080/fcrepo/rest/scape/metadata/${entityID}/${representationID}/PROVENANCE

$ curl -X GET
http://localhost:8080/fcrepo/rest/scape/metadata/${entityID}/${representationID}/TECHNICAL
```

Retrieve the technical metadata of a File
Technical metadata of a file can be retrieved via a GET request. The entity ID, the representation ID and the file ID must be given.

```
$ curl -X GET
http://localhost:8080/fcrepo/rest/scape/metadata/${entityID}/${representationID}/${fileID}/TECHNICAL
```

Retrieve the technical metadata of a bitstream
To retrieve the technical metadata of a bitstream, the bitstream ID must be appended to the GET request (along with the entity ID, representation ID and file ID).

```
$ curl -X GET
http://localhost:8080/fcrepo/rest/scape/metadata/${entityID}/$representationID}/${fileID}/${bitstreamI
D}/TECHNICAL
```

Retrieve the Version List of an Intellectual Entity
The version list of an Intellectual entity can be retrieved by the “entity-version-list” GET method. The ID of the Intellectual entity must be given.

```
$ curl -X GET http://localhost:8080/fcrepo/rest/scape/entity-version-list/${entityID}
```

Update an Intellectual Entity
In order to allow updating of Intellectual Entities the implementation exposes a PUT endpoint. The request must include the updated METS representation (e.g. entity-minimal.xml) of the entity in the request body.

```
$ curl -H "Content-Type:text/xml" -X PUT http://localhost:8080/fcrepo/rest/scape/entity/${entityID} -d
@${CONNECTOR_FOLDER}/src/test/resources/entity-minimal.xml
```

Update Metadata of an Intellectual Entity

To allow updating of the descriptive metadata of an Intellectual entity a PUT endpoint is exposed. The request must include the updated METS representation of the entity in the request body.

```
$ curl -H "Content-Type:text/xml" -X PUT
http://localhost:8080/fcrepo/rest/scape/metadata/${entityID}/DESCRIPTIVE -d '<dc:dublin-core
xmlns:dc="http://purl.org/dc/elements/1.1/"><dc:title>foo</dc:title></dc:dublin-core>'
```

Update Metadata of a Representation of an Intellectual Entity

To allow updating of the source metadata of a representation a PUT endpoint is exposed. For updating a Representation of an Intellectual entity without sending a METS representation of the Intellectual Entity the PUT endpoint consumes the XML representation of the metadata of the Representation. In the example below the descriptive metadata represented by DC is used. The repository *has to* create a new Version of the Intellectual Entity with the updated Representation

```
$ curl -H "Content-Type:text/xml" -X PUT
http://localhost:8080/fcrepo/rest/scape/metadata/${entityID}/${representationID}/SOURCE -d
'<dc:dublin-core xmlns:dc="http://purl.org/dc/elements/1.1/">
<dc:title>foo</dc:title>
</dc:dublin-core>'
```

Update Metadata of a File

To allow updating of the technical metadata of a file a PUT endpoint is exposed. The request must include the updated XML representation of the file metadata in the request body. In the example below we used textMD.

```
$ curl -H "Content-Type:text/xml" -X PUT
http://localhost:8080/fcrepo/rest/scape/metadata/${entityID}/$representationID}/${fileID}/TECHNICAL -
d '<textmd:textMD xmlns:textmd="info:lc/xmlns/textmd-v3">
<textmd:encoding><textmd:encoding_platform linebreak="LF">
</textmd:encoding_platform>
</textmd:encoding></textmd:textMD>'
```

Update Metadata of a Bitstream

To allow updating of the technical metadata of a representation a PUT endpoint is exposed. The request must include the updated XML representation of the bitstream metadata in the request body. In the example below we used DC.

```
$ curl -H "Content-Type:text/xml" -X PUT http://localhost:8080/fcrepo/rest/scape/metadata/entity-
1/representation-1/file-1/bitstream-1/TECHNICAL -d '<dc:dublin-core
xmlns:dc="http://purl.org/dc/elements/1.1/">
<dc:title>foo</dc:title></dc:dublin-core>'
```

Search Intellectual Entities, Representations and Files

For Intellectual Entity discovery the implementation exposes a SRU search endpoint. The endpoint implements the SRU specifications by the Library of Congress for Internet Search queries [15], utilizing CQL [16], 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” (please refer to the SRU specifications [16] for an explanation and usage of the request parameters). The query parameter contains the search term (wildcards are allowed). The search takes place in the metadata of Intellectual Entities, Representations and Files.

```
Search in metadata of intellectual entities:
$ curl -X GET
http://localhost:8080/fcrepo/rest/scape/sru/entities?version=1&operation=searchRetrieve&query=${sea
rchterm}

Search in metadata of representations:
$ curl -X GET
http://localhost:8080/fcrepo/rest/scape/sru/representations?version=1&operation=searchRetrieve&que
ry=${searchterm}

Search in metadata of files:
$ curl -X GET
http://localhost:8080/fcrepo/rest/scape/sru/files?version=1&operation=searchRetrieve&query=${search
term}
```

3.2.1.4 Usage of Plan Management API

The Plan Management API is a set of HTTP endpoints for serving content in a SCAPE environment. Its purpose is to integrate the various components in the SCAPE platform that handle preservation plans. Firstly, plans in use by the SCAPE platform are part of a digital object’s provenance and therefore the need to persist these arises, so the repository acts as a storage system for preservation plans. Secondly the Plan Management API acts as a Bridge in between the Workflow Execution Environment and a planning agent, relaying execution of plans as requested by the agent and supplying information about preservation plan state to the agent.

Reserve a Plan identifier:

In order to create new Plans the agent has to be aware of the identifier to use for the plan. Therefore a facility to request a reserved identifier is required, which is used when the preservation plan gets deployed on the repository.

```
$ curl -X GET http://localhost:8080/fcrepo/rest/scape/plan-id/reserve
```

Deploy a new Plan:

To deploy a new plan, a PUT endpoint is exposed. ${planID} in this example is the placeholder of reserved identifier for the plan that gets deployed.

```
$ curl -X PUT http://localhost:8080/fcrepo/rest/scape/plan/${planID} -d @${path to plan xml}
```

Retrieve a list of deployed Plans:

To retrieve a list of deployed plans inside of a repository a GET endpoint is exposed.

```
$ curl -X GET http://localhost:8080/fcrepo/rest/scape/plan-list
```

Search deployed Plans

In order to be able to search plans based on their significant properties an endpoint for SRU searching is exposed by the repository. The endpoint implements the SRU specifications by the Library of Congress for Internet Search queries [15], utilizing CQL, a standard syntax for representing queries [16], and exposes this functionality via a HTTP GET endpoint.

```
$ curl -X GET
http://localhost:8080/fcrepo/rest/scape/plan/sru?version=1&operation=searchRetrieve&query=*
```

Retrieve a Plan

An endpoint for plan retrieval is exposed by the repository. Plans are returned according to the Plato Version 4 XML schema [SW3]. The ID of the plan "planID" must be given.

```
$ curl -X GET http://localhost:8080/fcrepo/rest/scape/plan/${planID}
```

Retrieve Plan life cycle state

An agent requires the possibility to retrieve the life cycle state of a preservation plan. The state of the plan can be one of the two states "ENABLED" or "DISABLED".

```
$ curl -X GET  http://localhost:8080/fcrepo/rest/scape/plan-state/${planID}
```

Update Plan life cycle state

An agent requires the possibility to change the life cycle state of a preservation plan to enable and disable specific preservation plans. Only enabled plans can be executed on the Workflow Execution Environment. The new state of the plan can be one of the two states "ENABLED" or "DISABLED". The default state is dependent on the implementation of the API.

```
$ curl -X PUT  http://localhost:8080/fcrepo/rest/scape/plan-state/${planID}/DISABLED
```

Retrieve Plan execution state

In order to supply an Agent with feedback about the execution of preservation plans, the repository exposes and Endpoint for retrieving lists of execution states. The request produces a XML representation of all the execution states associated with a preservation plan. The execution state consists of a timestamp, a description and one of the three following values:

- EXECUTION_IN_PROGRESS
  The preservation plan is currently being executed on the Workflow Environment
- EXECUTION_SUCCESS
  The preservation plan is successfully executed on the Workflow Execution Environment
- EXECUTION_FAIL
  The execution of the preservation plan failed and the plan is not available for execution until an agent changes this state back to ENABLED

A detailed description can be found in the document "Plan Management API" [18].

```
$ curl -X GET http://localhost:8080/fcrepo/rest/scape/plan-execution-state/${planID}
```

Add a new Plan execution state
The workflow execution environment needs the possibility to update the execution state of preservation plans in order for this state to be available to the Plan Management GUI user. The request consumes a XML representation of the execution state.

```
$ curl -X POST http://localhost:8080/fcrepo/rest/scape/plan-exceution-state/${planID} -d
@/path/to/executionstate
```

3.3 RODA

RODA is an open source digital repository with long-term preservation and authenticity as its primary objectives. Created by the Portuguese National Archives in partnership with the University of Minho, it was designed to support the most recent archival standards and become a trustworthy digital repository.
RODA has been developed to be a complete digital repository providing functionality for all the main units that compose the OAIS reference model.

Additional information can be found at:

- RODA Community web page http://www.roda-community.org
- RODA GitHub Repository https://github.com/keeps/roda
- RODA-SCAPE GitHub Repository https://github.com/openplanets/roda

The SCAPE specific API implementations can be found at [SW7].

3.3.1 Installation and Usage of RODA

This section describes the usage and installation of RODA. An installer can be built from the source code available on GitHub [SW6].

3.3.1.1 Installation from source

The prerequisites to build RODA are:

- Git client
- Apache Maven
- Apache Ant
- OpenJDK 6 JDK

To install all dependencies in Debian based systems, execute:

```
$ sudo apt-get install git maven ant openjdk-6-jdk
```

To download the latest RODA source, execute:

```
$ git clone https://github.com/openplanets/roda.git roda
```

To compile, go to the RODA sources folder and execute the command:

```
$ mvn clean install -Pcreate-installer
```

After a successful compile, the installer will be available at *roda-installer/roda-installer.zip*. To install, uncompress the archive and follow the instructions on the file INSTALL.txt that is inside.

3.3.1.2 Usage of the Data Connector API

This paragraph shows some examples of how to use the Data Connector API with RODA. Please refer also the SCAPE Digital Object Model [SD5] and the SCAPE Data Connector API documents [17]. The usage of the Data Connector API with Fedora has been described in detail above, so the following only provides two examples with RODA. Please note the difference in the URLs, the methods are of course the same and the full Data Connector API is implemented within RODA.

Ingest an Intellectual Entity
To ingest an Intellectual entity into RODA, you must provide the credentials (username and password), the URL to the RODA repository (rodaURL) and the path to the METS file.

```
$ curl  -X POST ${username}:${password}@${rodaURL}/roda-core/rest/entity –d
${pathToIntellectualEntityXMLFile}
```

Retrieve an Intellectual Entity
To retrieve an Intellectual entity from RODA, you must provide the credentials (username and password), the URL to the RODA repository (“rodaURL”) and the ID of the Intellectual Entity (“entityID”).

```
$ curl -X GET ${username}:${password}@${rodaURL}/roda-core/rest/entity/${entityID}
```

3.3.1.3 Usage of the Plan Management API

This paragraph shows some examples of how to use the Plan Management API with RODA. Please refer also the SCAPE Digital Object Model [SD5] and the SCAPE Plan Management API [17] documents. Not all methods are shown, but just two of all the aforementioned REST endpoints.

Deploy a new plan
To deploy a new plan, used the exposed PUT endpoint.

```
$ curl  -X POST ${username}:${password}@${rodaURL}/roda-core/rest/plan/${planID} –d
${pathToPlanXmlRepresentation}
```

Note: the planID can be obtained via GET request at the URL ${rodaURL}/roda-core/rest/plan-id/reserve

Retrieve plan execution states
The plan execution state can be requested through the GET endpoint. The execution state consists of a timestamp, a description and a value (EXECUTION_IN_PROGRESS, EXECUTION_SUCCESS, or EXECUTION_FAIL).

```
$ curl -X GET ${username}:${password}@${rodaURL}/roda-core/rest/plan-execution-state/${planID}
```

3.3.1.4 Usage of the Report API

This paragraph shows some examples of how to use the Report API with RODA. Please refer also the SCAPE Digital Object Model [SD5] and the SCAPE Report API documents [17].

Identify the repository
This endpoint retrieves information about the repository. It produces an XML representation according to OAI-PMH specification [21] for Identify response [22].

```
$ curl -X GET ${username}:${password}@${rodaURL}/roda-core/report?verb=Identify
```

Retrieve a list of events (e.g. for PREMIS events)
This endpoint retrieves a list of events. The type of the events must be given in the metadataPrefix request parameter. In the example we used “premis-events-v2”. It produces XML according to the OAI-PMH specification for ListRecords response [23].

```
$ curl -X GET ${username}:${password}@${rodaURL}/roda-
core/report?verb=ListRecords&metadataPrefix=premis-event-v2
```

4 Summary

The SCAPE project delivered two repositories, Fedora 4 and RODA, as reference implementations for the SCAPE ecosystem, both of which integrate with the SCAPE tools and services. The document described the two repositories, focussing on how they integrate with the SCAPE tools and services in the SCAPE ecosystem. A description of the usage and installation of each repository and the usage (with examples) of their APIs has also been provided. The expectation is that this guidance will be of assistance in helping individuals work with SCAPE’s Fedora 4 or RODA repositories, or to those wishing to enhance their own repository for use within the SCAPE ecosystem.

5 References

[1] DuraSpace, http://www.duraspace.org/

[2] Fedora Steering Group, http://www.fedora-commons.org/steering-group

[3] Git revision control, http://git-scm.com/.

[4] GitHub, https://github.com/.

[5] MyExperiment, http://www.myexperiment.org.

[6] Scout REST API, http://projects.opf-labs.org/scape/scout/apidocs/eu/scape_project/watch/rest/WatchClient.html

[7] Apache Tomcat, http://tomcat.apache.org/.

[8] Plato, http://www.ifs.tuwien.ac.at/dp/plato/intro.html.

[9] Taverna Workbench, http://www.taverna.org.uk/download/workbench/.

[10] Taverna Server, http://www.taverna.org.uk/download/server/.

[11] MyExperiment REST API, http://wiki.myexperiment.org/index.php/Developer:API.

[12] MyExperiment Components API, http://wiki.myexperiment.org/index.php/Developer:Components.

[13] METS, http://www.loc.gov/standards/mets/.http://www.loc.gov/standards/mets/

[14] PREMIS, http://www.loc.gov/standards/premis/.http://www.loc.gov/standards/premis/

[15] Search/Retrieve via URL (SRU), http://www.loc.gov/standards/sru/.

[16] SRU Contextual Query language (CQL), http://www.loc.gov/standards/sru/specs/cql.htmlhttp://www.loc.gov/standards/sru/specs/cql.html.

[17] “SCAPE APIs”, SCAPE, 2012, https://github.com/openplanets/scape-apis

[18] "Plan Management API", F. Asseg, M. Hahn, 2012, SCAPE, https://github.com/openplanets/scape-apis/blob/master/Plan%20Management%20API_V1.0.pdf

[19] "Report API Specification", R. Castro, M. Ferreira, L. Faria, F. Asseg, P. Petrov, 2012, SCAPE, https://github.com/openplanets/scape-apis/blob/master/ReportAPI_V1.0.pdf.

[20] “Data Connector API”, F. Asseg, M. Hahn, 2012, SCAPE, https://github.com/openplanets/scape-apis/blob/master/Data_Connector-API_V1.0.pdf

[21] OAI-PMH, http://www.openarchives.org/OAI/openarchivesprotocol.htmlhttp://www.openarchives.org/OAI/openarchivesprotocol.html.

[22] OAI-PMH Identify http://www.openarchives.org/OAI/openarchivesprotocol.html#Identify

[23] OAI-PMH ListRecords http://www.openarchives.org/OAI/openarchivesprotocol.html#ListRecords

[24] Fedora Repository Wiki home, https://wiki.duraspace.org/display/FF/Fedora+Repository+Home.

[25] RODA - Repository of Authentic Digital Objects, http://www.roda-community.org/.

[26] Document Object Management System (DOMS), http://wiki.statsbiblioteket.dk/domswiki/.

[27] Hadoop API, https://hadoop.apache.org/docs/r2.2.0/api/.

[28] Apache Oozie Web Services API, https://oozie.apache.org/docs/4.0.0/WebServicesAPI.html

[29] Apache Hadoop, http://hadoop.apache.org/.http://hadoop.apache.org/

[30] Taverna Server REST API, http://dev.mygrid.org.uk/wiki/display/taverna/REST+API

5.1 SCAPE Deliverables

The following links provide references to SCAPE specific deliverables referenced within this report. All deliverables are (or will shortly be) available from:
http://www.scape-project.eu/category/deliverable.

[SD1] "D2.3 Technical Architecture Report v2", P. May, C. Wilson, 14-04-2014
[SD2] "D4.2: Final Platform Release", R. Schmidt, M. Rella, 31-01-2014, v1.0
[SD3] "D5.1: Guidelines for deploying preservation tools and environments", R. Schmidt, D. Tarrant, R. Castro, M. Ferreira, H. Silva, 2012,
[SD4] "D7.2: Workflow modelling environment", D. Fellows, 31-01-2014, v1.0
[SD5] "D8.1: Recommendations for Preservation-aware Digital Object Model", M. Hahn, F. Asseg, N. Sherwinter, R. Castro, 31-01-2014, v1.0

5.2 SCAPE Software References

The following links provide references to software tools and services developed in SCAPE and described or referenced in this report (i.e. this is not a complete list of SCAPE software).

[SW1] Scout
https://github.com/openplanets/scout
[SW2] C3PO
https://github.com/openplanets/c3po
[SW3] Plato
https://github.com/openplanets/plato
[SW4] Data Connector API Reference Implementation
https://github.com/openplanets/scape-fcrepo4-connector
[SW5] Plan Management API Reference Implementation
https://github.com/openplanets/scape-fcrepo4-planmanagement
[SW6] RODA
https://github.com/openplanets/roda
[SW7] SCAPE specific RODA
https://github.com/openplanets/roda/tree/master/roda-core/roda-core-services/src/main/java/eu/scape_project/roda/core
[SW8] SCAPE Digital Object Model
https://github.com/openplanets/scape-platform-datamodel
[SW9] Taverna Server with SCAPE Execution Service API
https://github.com/myGrid/taverna-server/tree/scape-execution-interface
[SW10] SCAPE Plan Management Web Application
https://github.com/openplanets/scape-planmanagement-webapp
[SW11] SCAPE Loader Application
https://github.com/openplanets/loader-app

[SW12] Integration Tests Data Connector API
https://github.com/openplanets/scape-fcrepo4-connector/blob/master/src/test/java/eu/scapeproject/fcrepo/integration/IntellectualEntitiesIT.java

6 Glossary

The following terms and abbreviations are used throughout this report:

Table 2: Terms and Definitions

Term/ Abbreviation	Description
AIP	Archival Information Package
API	Application Programming Interface
Automated Planning	A systematic and semi-automatic process that provides the ability to assess the impact of influencers and specify actionable preservation plans that define concrete courses of actions and the directives governing their execution. This is the operative management of obsolescence and maximizing expected value with minimal costs.
Automated Watch	A systematic and semi-automatic process that provides the ability to monitor external and internal entities for changes having a potential impact on preservation and to provide notification. The Automated Watch entity denotes the architectural software component that supports the Automated Watch process.
(SCAPE) Components	SCAPE Components are Taverna Components, identified by the SCAPE Preservation Components sub-project, that conform to the general SCAPE requirements for having annotation of their behaviour, inputs and outputs. SCAPE components may be stored in the SCAPE Component Catalogue.
Component Catalogue	The Component Catalogue is a searchable repository for the definitions of SCAPE Components, Component Families and Component Profiles. The Component Catalogue is implemented by the myExperiment service [20] and implements the Component Service API [36].
Digital Object Model	A data exchange model, based on METS and PREMIS, to encapsulate digital objects and ensure a consistent and well-understood information exchange between SCAPE entities.
Digital Object Repository	An OAIS Compliant repository that provides a data management solution for storing content and metadata about digital objects, as well as Preservation Plans. DORs implement three interfaces (see Section 1): Plan Management API; Data Connector API; and the Report API.
DIP	Dissemination Information Package
DOM	Digital Object Model
DOR	Digital Object Repository

Entity (architectural)	SCAPE architectural elements, e.g. the Execution Platform, the Digital Object Repository, etc. This term is used to avoid overloading the term "components", distinguishing "architectural components" from "SCAPE Components".
Execution Environment	An abstract layer of the Execution Platform which provides a placeholder representing functionality to be fulfilled by a specific technology. The Execution Environment provides the physical infrastructure to perform computation. An example might be the nodes of a Hadoop cluster.
Execution Platform	An infrastructure that provides the computational resources to enact a Preservation workflow and execute Preservation actions. Abstracted into three layers: the Execution Environment; the Job Execution Service and the Job Submission Service API.
File	A 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.
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
Intellectual Entity	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 may have one or more digital representations.
Job Execution Service	An abstract layer of the Execution Platform which provides a placeholder representing functionality to be fulfilled by a specific technology. The Job Execution Service provides job scheduling functionality, allocating computing tasks amongst the available hardware resources available within the Execution Environment. An example might be Taverna-Server [10] or Hadoop [29].
Job Submission Service API	An abstract layer of the Execution Platform which provides a placeholder representing functionality to be fulfilled by a specific technology. Provides the entry point to the Execution Platform, implementing a remotely accessible interface to enable a user or client application to schedule and execute workflows (jobs) on the Execution Environment. The exact interface depends on the underlying Job Execution Service and Execution Platform, but typical examples would be the Hadoop API provided over a SSH connection, or the Taverna-Server REST API over HTTP.
JSON	JavaScript Object Notation
METS	Metadata Encoding and Transmission Standard
OAIS	Open Archival Information System

PREMIS	PREservation Metadata: Implementation Strategies
Preservation Plan	A preservation plan is a live document that defines a series of preservation actions to be taken by a responsible institution due to an identified risk for a set of digital objects or records (called a collection). It is defined by Plato and stored in a Plan Management Service.
Preservation Action Plan	A Preservation Action Plan is part of a Preservation Plan and describes a set of digital objects, an operation (typically a transformation) to apply to each of them, and a rule that allows the determination of whether the operation on a particular digital object was successful. Success is determined on the basis of characteristics measured on the instantiation of the digital object, what it was transformed into, or the comparison of what it was and what it became. A Preservation Action Plan does not describe how to instantiate the DO, where to archive successful transformations, or where to report the outcome of applying the Preservation Action Plan.
REST	REpresentational State Transfer
SCAPE	SCAlable Preservation Environments
SIP	Submission Information Package
SRU	Search/Retrieve via URL
SSH	Secure SHell
Taverna Component	Taverna components are Taverna workflow fragments that are stored independently of the workflows that they are used in, and that are semantically annotated with information about what the behaviour of the workflow fragment is. They are logically related to a programming language shared library, though the mechanisms involved differ. Taverna components are stored in a component repository. This can either be a local directory, or a remote service that supports the Taverna Component API (e.g., the SCAPE Component Catalogue). Only components that are stored in a publicly accessible service can be used by a Taverna workflow that has been sent to a system that was not originally used to create it.
Taverna Server	Taverna Server is a multi-user service that can execute Taverna workflows. Clients do not need to understand those workflows in order to execute them.
Taverna Workbench	The Taverna Workbench is a desktop application for creating, editing and executing Taverna workflows.
Taverna Workflow	A Taverna workflow is a parallel data-processing program that can be executed by Taverna Workbench or Taverna Server. It is stored as an XML file, and has a graphical rendering.

URI	Uniform Resource Identifier
XML	eXtendable Markup Language

Appendix A – Installation and Usage of the Plan Management GUI

The Plan Management GUI can be used to manage plans inside of the DOR, display their execution states and submit them for execution.

Source Code & Deployment:

The Plan Management GUI is a JavaScript web application. Nevertheless a pre-packaged war file is available on Github [SW10] to provide a convenient way to deploy the webapp in Tomcat. Basically just a webserver such as Apache is needed to run the web application.

Configuration:

The configuration of the web application can be done via the `scape.properties` file (in Java Properties file format), which resides in the directory `src/main/webapp/WEB-INF/` (or in the `WEB-INF` directory on deployment):

```
# SCAPE Plan Management User Interface configuration
#
# Alter *.delegateURL, *.username and *.password to configure the service
# to which you delegate a URL. The repository.* properties configure where
# plans are stored, and the execute.* properties configure where plans are
# enacted.
#
# The *.servletURL.RE properties probably don't need to change unless you
# deploy the servlet as a root servlet or with a multi-directory path.
# Neither are usual (or recommended) configurations.

# Execution Delegate
execute.servletURL.RE: ^/[^/]*/execute
execute.delegateURL: https://localhost:8443/execute/scape
execute.username: FOO
execute.password: BAR

# Repository Delegate
repository.servletURL.RE: ^/[^/]*/repository
repository.delegateURL: http://localhost:8080/fcrepo/rest/scape
repository.username: FOO
repository.password: BAR
```

The properties intended for modification are:

- `execute.delegateURL`: the location of the Taverna plan execution REST API.
- `execute.username` and `execute.password`: the credentials for accessing the plan execution API of Taverna.
- `repository.delegateURL`: the location of the repository containing the digital objects, which will support the SCAPE Data Connector API.
- `repository.username` and `repository.password`: the credentials for accessing the digital object repository.

Note that clients will be unable to see the contents of the configuration file via the Plan Management GUI; it is entirely shrouded from users in order to ensure that deployments are secure (hidden username and password).

The `*.servletURL.RE` properties are not intended to be edited by users and will require changes to the GUI's HTML files if modified. (They are regular expressions used to match the part of the path that needs modifying during the internal site delegation processing.)