

Technical Specification– Sherpa DP

Document Details

Project name: Sherpa DP
Document type: technical
Author: Bodhimage/Hedges
Status: Version
Version: 1.0
Issue Date: 15/09/2006

Change history

Date	Version	Description	Author
15/09/2006	1.0	Development document	KB/MCH

Intended Audience

This document is written for use by AHDS Executive staff and SHERPA DP project partners.

Table Of Contents

1.	Introduction.....	4
1.1	Purpose	4
1.2	Scope	4
1.3	Expected Changes	4
1.4	References	4
1.5	Abbreviations used.....	4
2.	High-level Design	6
2.1	Fedora and the Fedora Service Framework.....	6
2.2	Overall system structure	6
2.3	Data storage.....	8
2.4	Hardware and software environment.....	8
2.5	Other design considerations	9
2.5.1	Security and Access	9
2.5.2	Ingest Workflow Automation	9
2.5.3	Scheduled Activities	10
2.5.4	Configuration Parameters	10
2.5.5	Future technological enhancements.....	10
2.5.6	Requirements resolution	
3.	Preservation Metadata	12
3.1	Overview	12
3.2	Object Metadata	13
3.3	Event Metadata.....	15
3.4	Relationship Metadata.....	19
3.5	Format-Specific Metadata	23
3.6	Audit Trails	23
4.	Component Descriptions – Sherpa DP Services.....	24
4.1	Introduction	24
4.2	Access and User Management Use Cases	24
4.2.1	User Logon.....	24
4.2.2	Add IR.....	24
4.2.3	Add User	25
4.3	Institutional Repository Use Cases.....	25
4.3.1	Request Item Deletion.....	25
4.3.2	Request DIP	26
4.3.3	Request Statistics Report	26
4.3.4	View Reports	27

4.3.5	View Holdings	27
4.4	Ingest Use Cases	27
4.4.1	Harvest Metadata	30
4.4.2	Process Harvest.....	30
4.4.3	Create SIP	31
4.4.4	Update AIP Metadata.....	32
4.4.5	Check SIP Integrity	33
4.4.6	Generate DC Metadata.....	34
4.4.7	Generate Technical Metadata for SIP	34
4.4.8	Generate Technical Metadata for File.....	35
4.4.9	Normalise Datastreams for SIP	37
4.4.10	Normalise Datastream.....	37
4.4.11	Generate Technical Metadata for Normalised Files.....	38
4.4.12	Generate Fedora Ingest Package	39
4.4.13	Ingest AIP	39
4.4.14	Generate Ingest Report.....	40
4.5	Post-Ingest Use Cases.....	41
4.5.1	Check Repository Integrity	42
4.5.2	Check Integrity of AIP	42
4.5.3	Check Format Obsolescence.....	43
4.5.4	Migrate Datastream.....	44
4.5.5	Delete AIP.....	44
4.5.6	Generate DIP	45
4.5.7	Generate Dissemination Report	46
4.5.8	Generate Statistics Report.....	47
5.	Component Descriptions – Fedora	48
5.1	Content Modelling Options	48
5.2	Expressing Relationships in Fedora	48
5.2.1	RELS-EXT.....	49
5.2.2	RELS-INT	50
5.3	E-print Versioning	50
5.4	Correspondence with OAIS Model	51
5.5	Fedora Ingest METS packages	52
5.5.1	Option 1: single Fedora object per e-print	53
5.5.2	Option 2: multiple Fedora objects per e-print.....	54
6.	Database specification	57
6.1	Logical database design.....	57
6.1.1	User.....	57
6.1.2	Role.....	57
6.1.3	Collection.....	57
6.1.4	Harvest.....	58
6.1.5	Eprint	58
6.1.6	File	58
6.1.7	Request.....	58
6.1.8	Event	59

1. Introduction

1.1 Purpose

This document describes the proposed technical solution for the Sherpa DP preservation repository. It currently comprises:

- a high-level overview of the proposed system;
- a more detailed specification of the main components.

1.2 Scope

The purpose of this document is threefold:

- to communicate the design of the system to AHDS staff and the project partners;
- to serve as a basis for the development of the Sherpa DP system;
- to document the system design and facilitate maintenance of the system.

1.3 Expected Changes

As system development progresses, the design information contained in this document will be updated and enhanced. A final version will be issued at the end of the project.

1.4 References

Sherpa DP Documents:

- [1] WP 4.4 – Preservation Metadata
- [2] WP 6.5 – Migration review
- [3] WP 5.6 - Fedora Software Review
- [4] WP 5.5 - Storage Assessment and Synchronisation Mechanism Review

Other documents:

- [5] PREMIS Data Dictionary

1.5 Abbreviations used

AHDS – Arts and Humanities Data Service
AIP – Archival Information Package
DIP – Dissemination Information Package
IR – Institutional Repository
JMS – Java Message Service
PID – Persistent Identifier
SIP – Submission Information Package
SRB – Storage Resource Broker

2. High-level Design

This section gives a high-level overview of the proposed system.

2.1 Fedora and the Fedora Service Framework

The kernel of the Sherpa DP preservation repository will be the Fedora repository management system, which will be used to store and manage the repository data and metadata. Fedora will supply the core functionality of the system, but it does not implement the specific ingest and preservation services required for Sherpa DP.

The core Fedora repository service exposes its functionality via a number of APIs, which are exposed as web service interfaces belonging to the main Fedora web application running in a dedicated instance of Tomcat. These core APIs comprise a repository management interface (API-M), a repository access interface (API-A), a basic repository search interface, and an RDF-based Resource Index search interface (for querying relational data).

From version 2.1, the core Fedora repository service incorporates the Fedora Service Framework, which provides a means for developers to build and integrate new services around a Fedora repository using a service-oriented architecture (SOA) approach. These services run as stand-alone web applications that are independent of the Fedora repository, but which can interact with the core repository service or with other services, providing additional functionality that facilitates the integration of the basic repository kernel into a broader application environment.

It is proposed to use this approach to developing the suite of services associated with the preservation repository, such as ingest services and preservation services, which are described in more detail in Section 4. These functional units will be developed as atomic, modular services, which interact with the Fedora repository as well as with other services, but which are only loosely coupled with the repository. They can be independently plugged in and replaced with other implementations to cope with enhancements or technological developments.

Currently there are no available guidelines for developing services that can be integrated into the service framework, although such guidelines are under development. The design patterns of framework services that have already been developed, such as the DirIngest and OAIPProvider services, should be used as models.

2.2 Overall system structure

Fig. 1 shows the overall structure of the system, together with the main components and dataflows.

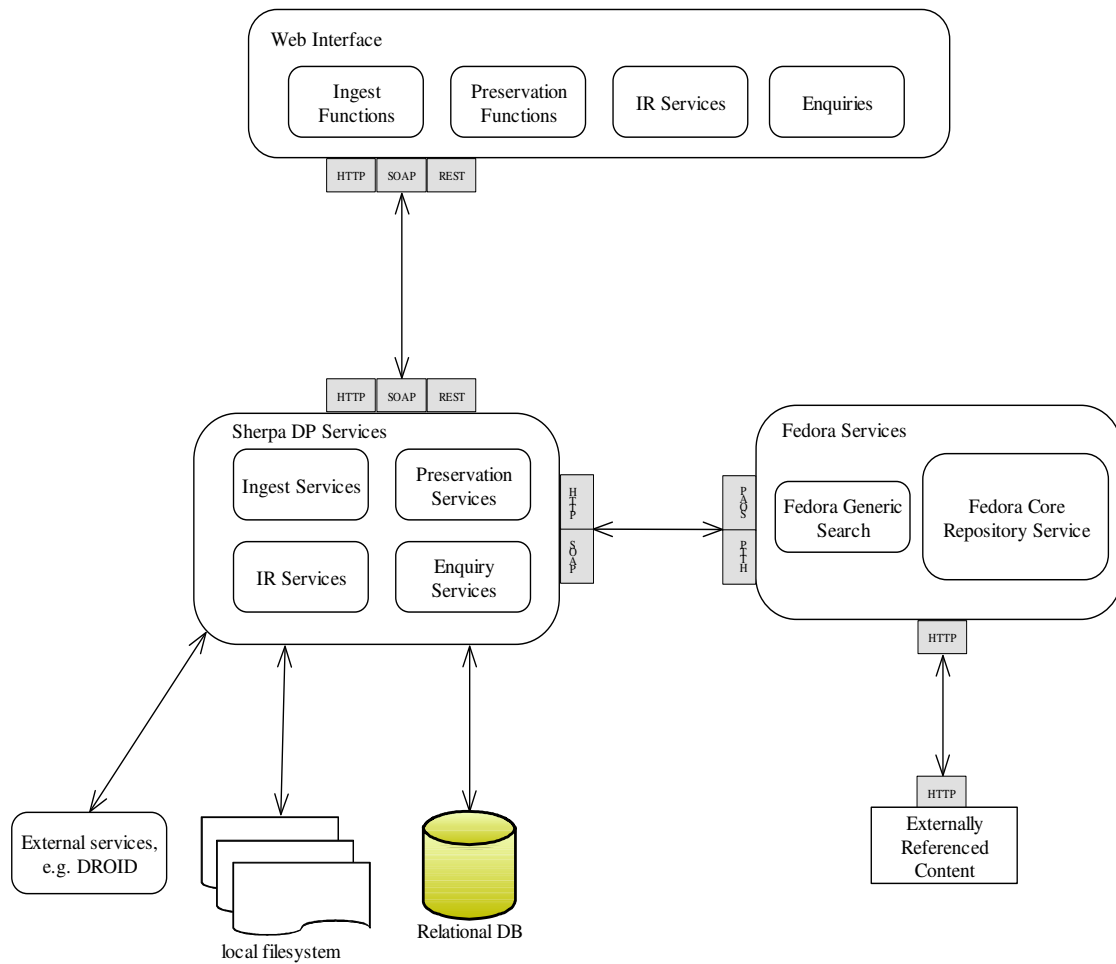


Fig. 1 - Overall System Structure

Web interface: users at both the institutional and the preservation repositories will access Sherpa DP services using a web interface, through which they will enter the required service parameters and receive service output and error messages. This will be a lightweight interface, and the bulk of the processing will be performed by individual web services, with which the interface will communicate using SOAP. Note: preservation repository users with appropriate authorisation will also be able to access the Fedora repository directly using one of the standard Fedora clients, although they should not update the repository in this way.

Sherpa DP web services: individual functions (described in detail in Section 4) will be implemented as modular web services, in accordance with the Fedora Service Framework. These services will communicate with the core Fedora repository (via its standard web service interfaces - see Section 2.1), with the Fedora Generic Search Service, with external services such as DROID, and with each other. They will also read and write data from/to a relational database and to the local file system.

Fedora services: In addition to the core repository services, the system will use the Generic Search Service, which allows any metadata datastream in a Fedora object to be indexed and searched.

2.3 Data storage

For the most part, information about an individual e-print or about one of its constituent datastreams is held as metadata within a Fedora object, once the e-print has actually been ingested into Fedora. This ensures that the data is closely linked to the object with which it is associated.

In the following cases, however, data is held either temporarily or permanently outside Fedora:

1) Certain data is captured or generated during the ingest process (see Section 4.4), before the e-print is ingested into Fedora, for example harvested metadata, captured data files or generated preservation metadata. Such data will be stored on the file system until the object is ingested. In the case of metadata, the information will be stored as XML files that will be included as sections of the METS documents used for ingest into Fedora. Once ingest is complete, the copies stored directly on the file system will be deleted to avoid duplication of data.

2) A relational database will be used to hold data that either does not relate to specific objects (e.g. about metadata harvests), or is not considered sufficiently important to be recorded within the Fedora objects, but which should nevertheless be stored for the purposes of audit or processing control. It will also be used to record the temporary locations of files described in (1). This information is not deleted, although old data will be archived periodically. The database is described in Section 6.

3) E-print content files may be stored on the AHDS preservation system, and ingested into Fedora as “externally referenced content”.

2.4 Hardware and software environment

The Sherpa DP system will run on a dedicated (possibly virtual) server running Linux. The AHDS preservation server, based around a CX300 SAN system, will be used for preservation storage. This preservation storage will be mirrored to an external site using SRB.

The implementation will use following off-the-shelf software components and associated libraries:

- Fedora Server (initially version 2.1.1).
- FedoraGSearch generic search plug-in (currently a Beta version, although it will in future be bundled with the Fedora server)
- MySQL database server (initially version 5.0).
- Elated web interface to Fedora. This will not be used as it stands, but parts of it will be re-used in the Sherpa DP web interface.
- JHOVE (initially version 1.1)

- DROID
- SRB

2.5 Other design considerations

2.5.1 Security and Access

Both institutional and preservation repository users will access the Sherpa DP repository using a bespoke web interface. They will authenticate themselves by username and password, which are checked against the user table in the database. Authorisation to individual functions will be defined by roles, although initially two roles (one corresponding to institutional users and one to AHDS preservation users) will suffice. The individual web services will ensure that institutional users only have access data associated with their home IR.

For repository management purposes, preservation users with appropriate authorisation will also be able to access the repository using one of the standard Fedora clients. However, as far as possible the repository will be updated only using the bespoke Sherpa DP functions.

The Fedora server runs inside Tomcat, and basic authentication to access repository is provided by the Tomcat's server container, which can be configured to use simple a user/password file or LDAP for authentication. In addition, the Fedora server repository exposes web service APIs for accessing and managing the repository, and Fedora can be configured to use SSL authentication for either or both of these APIs.

In the future, as Shibboleth becomes more widely adopted by UK higher education and research institutions, it is likely that a Shibboleth-based access management system will be used to manage access of staff to Sherpa DP functions and to secure the Fedora repository. Shibboleth can be used to restrict access both to individual functions and to specific repository objects.

Physical security will be ensured by locating the hardware running the system in the AHDS' secure machine room. Access to this room is limited to designated key holders. Data will be secured by tape backup and mirroring to offsite SRB storage.

2.5.2 Ingest Workflow Automation

The stages of the ingest process (see Section 4.4) are linked by dependency relationships, whereby one operation cannot be performed until certain other operations are complete, and data generated by one operation flows into the next (see Fig. 3). In other words, if we ignore error conditions, these operations form a DAG (Directed Acyclic Graph).

Ideally, the operations would be combined to form a workflow, which, once activated, would run to completion without user intervention. At present, some operations, such as format recognition and metadata generation, will use tools that are still under development, and will require user input before processing can continue. The use case descriptions indicate which operations are likely to require intervention, and which ones can be automatically chained together.

The dependencies between operations and the automatic chaining of operations may in future be implemented using one of the several currently available workflow tools. In the initial Sherpa DP system, these functions will be implemented in a control layer above the individual service implementations. Dependency checking can be implemented by inspecting event logs or maintaining status information.

2.5.3 Scheduled Activities

A number of activities will be performed regularly on the preservation repository, without user intervention. These activities are indicated in the use cases by an actor of “Timer”. They will be scheduled at configurable intervals, using a suitable mechanism such as the Unix cron utility, Java Timer Task, or the Quartz job scheduler. Each such activity will have a separate activation period, configured in the application’s config.properties file.

2.5.4 Configuration Parameters

A number of Sherpa DP system parameters will be configurable in the config.properties file, including the following:

1. Database connectivity information.
2. Fedora configuration parameters.
3. Mail server parameters (for sending mails to IR staff).
4. File system directories used for holding harvested metadata, SIPs, reports and DIPs. Also archive directories if it is decided to configure Fedora to use externally referenced content.
5. Checksum algorithm (SHA or MD5) to be used, by default using MD5
6. SRB configuration parameters (in future)
7. Flags to indicate whether or not to delete DIPs after dissemination, ingest related files when ingest complete, reports after download.
8. Indexing fields for Lucene indexer.
9. Obsolescence check period.
10. Integrity check period.

2.5.5 Future technological enhancements

A number of technological developments were considered to be relevant to the Sherpa DP project, but were not included, either because they are not at present sufficiently developed to be included in the system, or because it was not possible to investigate and assess them in the available timescales. They are listed here and may be incorporated into the system as future upgrades:

- Fedora preservation services. The Fedora development team are planning a number of plug-in services that implement preservation-related functions, analogous to some of the services specified in this document. Details are not available at present, but the services will be developed using the Fedora Services Framework (see Section 2.1).
- Workflow management tools. There are a number of tools that enable web services to be combined in an automated fashion, by representing dependencies between services and allowing data produced by one service to be piped into another. Such tools would be of

significant utility, particularly during the ingest process, and they are currently being assessed by the AHDS as part of another project. In the initial system, similar (but limited and system-specific) functionality will be provided within the web application.

- Storage Resource Broker (SRB). SDSC has modified Fedora's Java I/O interface to enable access to storage managed by SRB, and the modification is available as a plug-in from Fedora. Further investigation is needed to assess this. In the initial system, the mechanisms used currently within the AHDS will be used to replicate the archived e-prints in SRB storage.
- Shibboleth-managed access. In the future, as Shibboleth becomes more widely adopted by UK higher education and research institutions, it is likely that a Shibboleth-based access management system will be used to manage access of staff to Sherpa DP functions and to secure the Fedora repository.

3. Preservation Metadata

3.1 Overview

The preservation repository will hold preservation metadata for the digital objects stored within it. This metadata will follow the proposals in the PREMIS Data Dictionary [5]. The subset of metadata elements to be used for Sherpa DP is defined in WP 4.4 [1], although the following modifies slightly the proposals in that document.

This section describes at an abstract level the metadata stored and the values assigned to the metadata elements. The mapping between abstract metadata and the Fedora METS structures is described in Section 52.

The PREMIS proposal differentiates between object metadata and event metadata, where object metadata includes the description of relationships with other objects. Here, relationship metadata is treated separately.

In addition to the generic PREMIS preservation metadata, in some cases it will be necessary to store format-specific metadata. In particular, this applies to image files, in which case MIX metadata will be stored.

3.2 Object Metadata

Semantic Unit(s)	Semantic Unit	Value	Applies to
objectIdentifier	objectIdentifierValue	[Fedora object PID or datastream id (e.g. PID/DS-ID)]	e-print or file
objectIdentifier	objectIdentifierType	[Implicit within objectIdentifier]	e-print or file
preservationLevel		Full ¹	file
objectCharacteristics	compositionLevel	[1 if JHOVE output contains a repInfo/properties/property/property element with sub-element name = "Encryption", 0 otherwise] ²	file
objectCharacteristics	size	[JHOVE: repInfo/size]	file
objectCharacteristics/format/formatDesignation	formatName	[JHOVE: repInfo/mimetype]	file
objectCharacteristics/format/formatDesignation	formatVersion	[JHOVE: repInfo/version]	file
objectCharacteristics/format/formatDesignation	significantProperties	[JHOVE: from repInfo/properties/property/property elements] ³	file
objectCharacteristics/inhibitors	inhibitorType	[If JHOVE output contains an encryption property (see	file

¹ The default value is "Full" (see WP 4.4, Section 2.2). The preservation user will have the option of changing this (to a value within a controlled vocabulary) via the web interface, if requested by the IR. In future, this element could be added to the harvested metadata and the required value incorporated automatically.

² objectCharacteristics.compositionalLevel represents the number of processes of additional encoding (e.g. encryption) or bundling (e.g. compression) to which a file has been subjected. During the ingest process, the software will attempt to determine whether each datastream has been processed in this way. For Sherpa DP, we may assume that a file may be encrypted, but will not be compressed. Enhanced processing will need to be implemented for future extensions.

³ Some of the properties generated by JHOVE may be regarded as significant by the preservation repository administrators, others not significant. In addition, other properties (not included in the JHOVE output) may be regarded as significant. A possible approach: configure, for each expected file format, a list of JHOVE properties that may be significant. Include these in the metadata automatically, then allow the preservation user to review the list, and possibly add other properties manually, before proceeding. The properties will be stored as name/value pairs.

		objectCharacteristics), set to the value of its sub-element of name = "Algorithm", otherwise N/A] ⁴	
objectCharacteristics/inhibitors	inhibitorTarget	[Not clear how to extract this from the JHOVE metadata, although it may be possible if better JHOVE documentation could be found. Otherwise, enter manually, using a controlled vocabulary]	file
objectCharacteristics/inhibitors	inhibitorKey	[Cannot be generated automatically. If required, this will have to be supplied by the IR]	file
objectCharacteristics/fixity	messageDigestAlgorithm	MD5	
objectCharacteristics/fixity	messageDigest	[JHOVE: repInfo/checksums/checksum (TYPE="MD5")] ⁵	file
objectCharacteristics/fixity	messageDigestOriginator	[Identifies the IR, using the controlled vocabulary from WP 4.4, Section 2.1]	file
objectCharacteristics/format/formatRegistry	formatRegistryName	PRONOM ⁶	file
objectCharacteristics/format/formatRegistry	formatRegistryKey	[PRONOM PUID]	file
objectCharacteristics/format/formatRegistry	formatRegistryRole	[N/A, but see footnote]	file

⁴ The simplest approach, which is indicated in the table, would be to use the values output by JHOVE, assuming that these belong to a controlled vocabulary whose literal values will persist in later releases of JHOVE (although the vocabulary may be extended). It is not entirely clear from the JHOVE documentation that this is the case. An alternative would be to create our own controlled vocabulary, along the lines of that in WP 4.4, Section 2.4.4.1, and create a mapping to this vocabulary from the JHOVE values. This mapping can be changed for subsequent versions of JHOVE, if necessary, so that the literal values of our vocabulary remain the same.

⁵ This will also be contained in the original metadata (participant repositories are required to supply MD5 checksums). By the processing stage in which technical metadata is generated, it will have been verified that the JHOVE value is the same as the value supplied by the IR.

⁶ Currently, PRONOM is the only registry of practical utility, and initially will be the only one used by Sherpa DP. Other registries, such as FRED or GDFR, are likely to be useful in the future, and the domains of formatRegistryName and formatRegistryKey will change as appropriate. Indeed several registries may be used for different purposes, and in such cases, formatRegistry will be a repeating element, with formatRegistryRole indicating the purpose or use.

Table 1 - Object Metadata

3.3 Event Metadata

In the PREMIS data dictionary, the following semantic units are associated with an event:

Semantic Unit	Semantic Unit	Value	Where stored
eventIdentifier	eventIdentifierType	ahds.ac.uk/sherpadp/event	implicit ⁷
eventIdentifier	eventIdentifierValue	[A positive integer, unique within the Sherpa DP system]	event metadata
eventType		[Depends on event – see Table 3]	event metadata
eventDateTime		[The time the event occurred, in the format defined in WP 4.4, Section 2.5.3. If an event corresponds to a process that takes a significant period of time, a date range (i.e. start time and end time) may be recorded (or just the completed time, or 2 events for start and end ???).]	event metadata
eventDetail		[Depends on event – see Table 3]	event metadata
eventOutcomeInformation	eventOutcome	[Depends on event – see Table 3]	event metadata
eventOutcomeInformation	eventOutcomeDetail	[Depends on event – see Table 3]	event metadata
linkingAgentIdentifier	linkingAgentIdentifierType	[Domain within which linkingAgentIdentifierValue is unique]	event metadata
linkingAgentIdentifier	linkingAgentIdentifierValue	[Either: (i) Identification of the user whose action resulted in the event, if any; (ii) Identification of the software tool or service that was involved in the	event metadata

⁷ eventIdentifierType identifies the domain in which eventIdentifierValue is unique. Assuming that all the events recorded are generated within the AHDS preservation system, this will be set to a fixed value, for example ahds.ac.uk/sherpadp/event. If event logging is extended to include events raised by an IR prior to an e-print's capture at the AHDS, eventIdentifierType will identify the IR (using the controlled vocabulary in WP 4.4 Section ???) and eventIdentifierValue will be a value determined by the IR.

		generation of the event, if any (we may want to distinguish between local tools and remote services)] ⁸	
linkingAgentIdentifier	linkingAgentRole	[Identifies the “type” of linkingAgentIdentifierValue: select from a controlled vocabulary, say “user”, “tool”, “service”]	event metadata
linkingObjectIdentifier	linkingObjectIdentifierType	[The objectIdentifierType and objectIdentifierValue of the object associated with the event]	Implicit – the event is associated with an object by the Fedora METS structure.
linkingObjectIdentifier	linkingObjectIdentifierValue		

Table 2 - Event Metadata

The following table shows the event types and associated event details that will be recorded, together with the associated linking object (which corresponds to the linkingObjectIdentifier semantic unit).

eventType	eventDetail	Comment	Linking Object
capture	E-print captured	Initial capture from IR, at start of ingest process.	e-print
capture	Updated metadata captured	Update of the original metadata at the request of the IR (see Section 4.4.3)	e-print
message digest calculation	Message digest calculated	Details of the message digest calculation (algorithm used etc.) are included in the messageDigest element.	file
fixity check	Compare values of the [date] and [date] message digests.	Discrepancy between the two message digests is recorded in the	file

⁸ linkingAgentIdentifier is repeating, so both of these can occur.

		eventOutcome element.	
viruscheck	Virus check performed using [software application]		file
validation	Validated to [standard] through use of [tool or online service].	e.g. when JHOVE is used to check that a file is of the alleged format.	file
normalization ⁹	Normalised from [source file format] to [target file format].	Transformation of a file to a format more suitable for preservation. This may be carried out during ingest processing.	file (the target file)
normalization ¹⁰	Normalised manifestation created.	Creation of a normalised manifestation of an e-print. This will involve one or more file normalizations (see previous eventType)	manifestation (the created manifestation)
ingestion	Ingested into preservation repository	Ingest into Fedora repository, at end of ingest process.	e-print
deletion	Deleted from preservation repository	Included for completeness. However, it is not envisaged that any e-print will be removed from the preservation repository. Instead it will be deaccessioned (see following).	e-print
deaccession	Deaccessioned from preservation repository; requested by [institutional repository]; reason: [Reason for deaccession].	The e-print is marked as deleted in Fedora. Thenceforth it is preserved at a bitstream level only, and is invisible to the IR.	e-print
migration	Migrated from [source file format] to [output file format].	Transformation of a file from a format judged to be obsolescent to a more contemporary format, or to a	file (the target file)

⁹ Spelling deliberate to conform to PREMIS document.

¹⁰ Spelling deliberate to conform to PREMIS document.

		format requested by an IR. This will in only be carried out post-ingest (into Fedora).	
migration	Migrated manifestation created.	Creation of a migrated manifestation of an e-print. This will involve one or more file migrations (see previous eventType)	manifestation (the created manifestation)
resubmission request	Resubmission request submitted to [institutional repository]		e-print
dissemination	Dissemination Information Package generated for [institutional repository]		e-print

Table 3 - Event Types

Table ??? in WP 4.4 lists the event outcomes that may be associated with each event type. This is not repeated here, although that table may need to be further refined and extended.

3.4 Relationship Metadata

The relationship metadata in the preservation repository must record at least the following relationships:

- i) Between an e-print and each of its constituent datastreams (original or derived)
- ii) Between a derived datastream and its source datastream
- iii) Between different versions of the “same” e-print (e.g. pre- and post-print versions)
- iv) Between an e-print and each of its manifestations
- v) Between a manifestation and its constituent datastreams.

These relationships are illustrated in the following diagram:

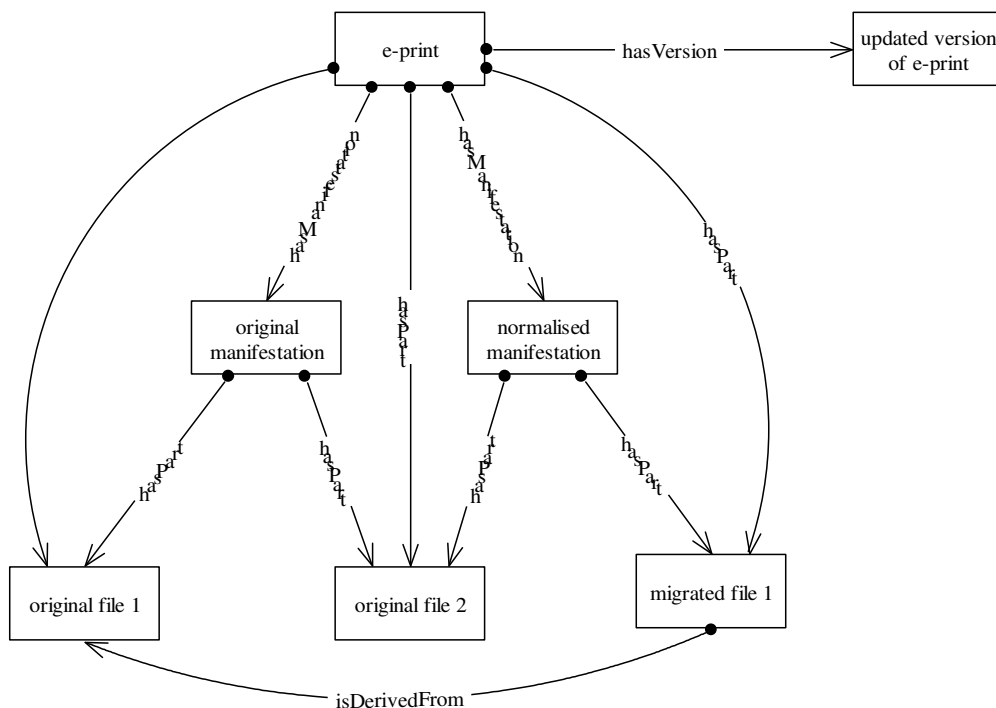


Figure 2 - Relationship Examples

The following tables indicate, for each of these cases, the values that will be used for each relationship semantic unit defined in the PREMIS data dictionary. In the PREMIS data model, the relationship semantic unit is aggregated within an object – below, I have referred to this as the source object, and the object referenced by relatedObjectIdentification as the related object.

Each relationship has an obvious inverse. For example, if A hasPart B, then B isPartOf A. For clarity, these have been included neither in Figure 2 nor in the tables below.

i) Between an e-print object and each of its constituent datastream objects.

Source object: The e-print object

Related object: An original datastream object

Semantic Unit	Semantic Unit	Value
relationshipType		structural
relationshipSubType		hasPart
relatedObjectIdentification	relatedObjectIdentifierType	[from the constituent datastream object]
relatedObjectIdentification	relatedObjectIdentifierValue	[from the constituent datastream object]
relatedObjectIdentification	relatedObjectIdentifierSequence	[Preserve the order of the files as listed in the original metadata ¹¹]
relatedEventIdentification	relatedEventIdentifierType	[from the capture event associated with the e-print.]
relatedEventIdentification	relatedEventIdentifierValue	[from the capture event associated with the e-print.]
relatedEventIdentification	relatedEventSequence	0

Owning object: The e-print object

Related object: A derived datastream object.

Semantic Unit	Semantic Component	Value
relationshipType		structural
relationshipSubType		hasPart
relatedObjectIdentification	relatedObjectIdentifierType	[from the constituent datastream object]
relatedObjectIdentification	relatedObjectIdentifierValue	[from the constituent datastream object]
relatedObjectIdentification	relatedObjectIdentifierSequence	0
relatedEventIdentification	relatedEventIdentifierType	[from the normalization or migration event associated with the datastream object.]
relatedEventIdentification	relatedEventIdentifierValue	[from the normalization or migration event associated with the datastream object.]
relatedEventIdentification	relatedEventSequence	0

¹¹ This order may not be significant.

ii) Between a migrated/normalised datastream and its source datastream:

Owning object: the target datastream object

Related object: the source datastream object

Semantic Unit	Semantic Component	Value
relationshipType		derivation
relationshipSubType		isDerivedFrom
relatedObjectIdentification	relatedObjectIdentifierType	[from the source datastream object]
relatedObjectIdentification	relatedObjectIdentifierValue	[from the source datastream object]
relatedObjectIdentification	relatedObjectIdentifierSequence	[Increment with each successive normalisation/migration]
relatedEventIdentification	relatedEventIdentifierType	[from the migration or normalisation event associated with the creation of the derived object.]
relatedEventIdentification	relatedEventIdentifierValue	[from the migration or normalisation event associated with the creation of the derived object.]
relatedEventIdentification	relatedEventSequence	0

iii) Between different versions of the “same” e-print (e.g. pre- and post-print versions).

Owning object: Any e-print object that has multiple versions in the preservation repository.

Related object: Any e-print object that is another version of the owning object.

Semantic Unit	Semantic Component	Value
relationshipType		derivation
relationshipSubType		hasVersion
relatedObjectIdentification	relatedObjectIdentifierType	[from other e-print version object]
relatedObjectIdentification	relatedObjectIdentifierValue	[from other e-print version object]
relatedObjectIdentification	relatedObjectIdentifierSequence	[identifies the order of deposit of the e-print versions]
relatedEventIdentification	relatedEventIdentifierType	[from ingestion event associated with the later e-print object]
relatedEventIdentification	relatedEventIdentifierValue	[from ingestion event associated with the later

		e-print object]
relatedEventIdentification	relatedEventSequence	0

iv) Between an e-print and each of its manifestations.

Owning object: Any e-print object.

Related object: Any manifestation of the e-print.

Semantic Unit	Semantic Component	Value
relationshipType		derivation
relationshipSubType		hasManifestation
relatedObjectIdentification	relatedObjectIdentifierType	[from manifestation]
relatedObjectIdentification	relatedObjectIdentifierValue	[from manifestation]
relatedObjectIdentification	relatedObjectIdentifierSequence	[the order in which the manifestations were created]
relatedEventIdentification	relatedEventIdentifierType	[from the migration or normalisation event associated with the creation of the manifestation]
relatedEventIdentification	relatedEventIdentifierValue	[from the migration or normalisation event associated with the creation of the manifestation]
relatedEventIdentification	relatedEventSequence	0

v) Between a manifestation and its constituent datastreams.

Owning object: Any manifestation of an e-print.

Related object: Any component datastream of the manifestation.

Semantic Unit	Semantic Component	Value
relationshipType		structural
relationshipSubType		hasPart
relatedObjectIdentification	relatedObjectIdentifierType	[from datastream object]
relatedObjectIdentification	relatedObjectIdentifierValue	[from datastream object]
relatedObjectIdentification	relatedObjectIdentifierSequence	[the order in which the manifestations were created]
relatedEventIdentification	relatedEventIdentifierType	[from the migration or normalisation event associated with the creation of the manifestation]

relatedEventIdentification	relatedEventIdentifierValue	[from the migration or normalisation event associated with the creation of the manifestation]
relatedEventIdentification	relatedEventSequence	0

3.5 Format-Specific Metadata

For some file formats, it will be necessary to store format-specific technical metadata in addition to the generic PREMIS preservation metadata. This will be configured in Sherpa DP on a case-by-case basis, as required. Initially, it will apply only to image formats, in which case MIX metadata will be used, as generated by JHOVE.

3.6 Audit Trails

Sherpa DP will log various items of audit data during the lifecycle of a digital object, providing a comprehensive audit trail for the object.

Events that are judged to be significant in the history of an object will be logged in accordance with the PREMIS standard for events, as described above. These events will be stored as XML within a dedicated inline XML datastream of a Fedora object, once the object has been ingested. During the ingest process, the data will have to be logged to a separate XML file, as the Fedora object will not yet have been created. This initial segment of the event metadata will be included in the METS ingest package. Subsequent events will only be recorded in the Fedora object (using the API-M method `ModifyDatastreamXMLMetadata`).

There is an example XML schema for PREMIS events at www.loc.gov/standards/premis/Event-v1-0.xsd. A schema for use in Sherpa DP must be created.

There will be other incidents that are not considered sufficiently important to be recorded as PREMIS events, or that are not associated with a specific object, but which should nevertheless be stored as part of the audit trail. These will be held in the database table `EVENT` (see Section 6.1.8). Some of these are referenced in the use cases, using symbolic names starting with `EVT_`, and more may be added during development.

Events will be logged using a common “log event” interface, which takes the appropriate action depending on the type of event. It is proposed to use JMS, with the events encoded as XML in the message bodies. This would provide a loosely couple architecture, making it easier to work in a distributed environment and simplify the future replacement of components. Message selectors could be used to filter the events and take the appropriate action (whether to create a PREMIS event or write an audit record to the database). Calls to the JMS interface will be wrapped within a web service interface, like the other Sherpa DP services.

4. Component Descriptions – Sherpa DP Services

4.1 Introduction

This section presents the individual services of the Sherpa DP Services component in Fig. 1 in the form of detailed use cases. The use cases will be subject to further analysis during system implementation.

In particular, the following points should be noted:

- Functionality with less predictable results, such as preservation metadata generation or format migration, is likely to require additional user input to manage the processing. In such cases, this is noted in the accompanying notes. However, to avoid complicating the sequence of operations, the precise flow of the user interaction in such cases is not described in the use case, and will be determined later.
- Processing of error conditions is not explicitly mentioned.

The use cases are categorised as follows:

- Access and user management functions
- Institutional repository functions
- Ingest functions
- Post-ingest functions (including preservation)

4.2 Access and User Management Use Cases

4.2.1 User Logon

Description: User logs on to Sherpa DP system.

Preconditions: User not logged on.

Triggers: User.

Summary:

User enters username and password.

User authenticated against USER table.

Authorisation of access to functions depends on associated role.

Post-conditions: The user is logged on. The appropriate functions are available

Alternative paths: Logon failure.

Notes:

- i) Access management is likely to be modified in future to use a Shibboleth-based system. The design should take this into account.

4.2.2 Add IR

Description: Add a new IR to Sherpa DP.

Preconditions:

Triggers: Preservation User.

Summary:

Add entry to COLLECTION table.

Post-conditions: collection table contains entry for new IR.

Alternative paths:

Notes:

- i) We could also add an “IR collection” object to Fedora as well, which will be linked to all e-prints belonging to the IR. TBD whether this would be useful.

4.2.3 Add User

Description: Add a new user to Sherpa DP.

Preconditions:

Triggers: Preservation User.

Summary:

Add new entry to the USER table.

Post-conditions: user table contains entry for new user.

Alternative paths:

Notes:

4.3 Institutional Repository Use Cases

These use cases describe functions available to the IR users.

4.3.1 Request Item Deletion

Description: Request the removal of an e-print from Sherpa DP.

Preconditions:

Triggers: IR User.

Summary:

User requests removal of e-print, specified by IR e-print id.

Add entry to REQUEST.

Notify preservation admin about request.

Post-conditions: Entry in REQUEST queue.

Alternative paths:

Notes:

- i) The user will select an e-print from a list.

- ii) The notification method(s) should be encapsulated. Initially, use the existing RT system.

4.3.2 Request DIP

Description: Request a DIP for an e-print in Sherpa DP. IR staff will be able to request for any e-print in the preservation archive that originates from their home IR.

Preconditions:

Triggers: IR User.

Summary:

User requests DIP, specified by IR e-print id.

Add entry to REQUEST.

Notify preservation admin about request.

Post-conditions: Entry in REQUEST queue.

Alternative paths:

Notes:

- i) The user will select an e-print from a list.
- ii) A DIP will be requested by an IR if, e.g., it has been migrated to new format at the preservation archive, or if the original has been lost or corrupted.
- iii) A possible enhancement would be to allow the user to specify a format (or set of formats for e-prints containing multiple files), although this is likely to require more manual input. By default, use the original format, or the latest migrated format if the original has been superseded.

4.3.3 Request Statistics Report

Description: Request a statistics report for the user's home IR.

Preconditions:

Triggers: IR User.

Summary:

User requests statistics report for a specified reporting period. The period can be open-ended.

Add entry to REQUEST.

Notify preservation admin about request.

Post-conditions: Entry in REQUEST queue.

Alternative paths:

Notes:

4.3.4 View Reports

Description: View the reports available in the IR's report area.

Preconditions:

Triggers: IR User.

Summary:

User lists reports available for IR.

User selects and displays/prints report.

Post-conditions:

Alternative paths:

Notes:

4.3.5 View Holdings

Description: Online display of holdings for the IR.

Preconditions:

Triggers: IR User.

Summary:

This function would display information that is less detailed than that in the statistics report, and may include aggregated data across all IRs. See the PRESERV function as an example. The precise contents are TBD.

Post-conditions:

Alternative paths:

Notes:

4.4 Ingest Use Cases

These use cases describe the stages involved in ingesting e-prints into the Sherpa DP preservation repository. Fig. 3 shows the flow of data from one stage to another, and the dependency relationships between these stages, which require that certain operations are complete before others can take place. "Complete" here implies "complete and successful", e.g. if a checksum mismatch is detected the process cannot continue until the problem is rectified.

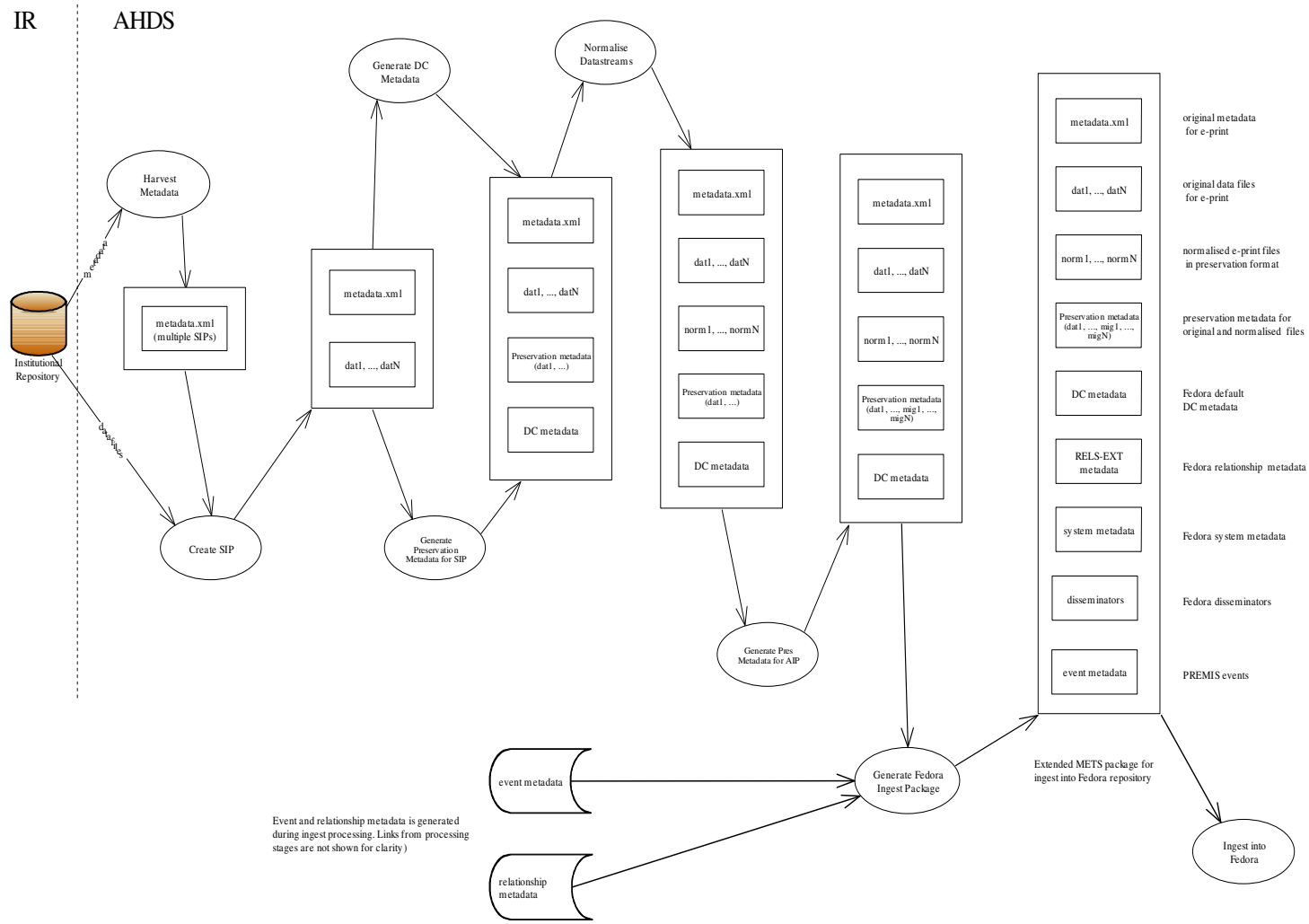


Fig. 3 - Ingest



4.4.1 Harvest Metadata

Description: Harvest new or modified metadata from an IR.

Preconditions: None.

Triggers: Timer (configurable); Preservation User.

Summary:

Send harvest request to IR.

When harvest is complete, create HARVEST entry.

Log EVT_HARVEST.

Post-conditions: The harvest area for the IR contains an XML file that contains metadata for each e-print in the IR that has been added or updated since the previous harvest.

Alternative paths: If an error occurs during harvest, manual intervention will be required to correct the error and restart the harvest.

Notes:

- i) An incremental harvest from each IR will be performed at configurable intervals.
- ii) The format of the harvested metadata record will depend on whether the IR uses EPrints or DSpace. Examples are contained in [\\Shared\Projects\SHERPA\DP\Kirti\preferred solution-phase \metadata\EPrints-metadata.xml](#) and [Dspace-mets-metadata.xml](#) respectively. However, these examples need to be reviewed, and XML schemas should be created.
- iii) The system could be configured to initiate “Process Harvest” automatically on completion of this use case.

4.4.2 Process Harvest

Description: Process the harvested metadata record.

Preconditions: Unprocessed metadata file exists in the IR’s harvested directory.

Triggers: Preservation User

Summary:

For each e-print entry in the metadata record:

Extract metadata for e-print and create XML metadata file in SIP area

Query preservation repository for object (identified by IR + original e-print id)

If not found

/* This is a “new” e-print */

Execute “Create SIP” use case.

Else

/* This is a metadata update for a previously deposited e-print */

Execute “Update AIP Metadata” use case.

End-if

End-For

If successful, mark HARVEST as processed

Log EVT_HARVEST_PROCESSED event.

Post-conditions: The SIP area for the IR contains a directory for each new or modified e-print. This directory contains the metadata and (for new e-prints) any data files referenced by URLs in the metadata.

Alternative paths:

Notes:

- i) Identifiers: EPrints repositories assign a unique (within the IR) numerical id in the <eprintid> element, while DSpace assigns a (globally) unique handle (e.g. <http://hdl.handle.net/123456789/37>) in the <mods:identifier> element.

It is assumed here that: (a) EPrints and DSpace repositories treat each revised version of an e-print (if the data itself is changed) as a separate item with a new identifier, with a reference in the metadata to other revisions. (b) If only metadata is changed, the item will be re-harvested but the identifier will be unchanged.

SherpaDP will treat each revision of an e-print (if the data itself is changed) as a separate object, so in this use case a separate SIP is created. If only the metadata is changed, the metadata in the existing Fedora object will be updated (versioning will be used both versions).

It must be verified that the above assumptions are correct. If not, the use case will need revision. Furthermore, it is not clear how different versions of an e-print are linked in the metadata – this uncertainty needs to be resolved.

- ii) This processing may be initiated automatically by the “Harvest Metadata” use case, depending on configuration. Also, could configure system to initiate “Check SIP Integrity” automatically on completion of this use case.

4.4.3 Create SIP

Description: Create a SIP.

Preconditions: Metadata for SIP extracted from harvested record.

Triggers: “Process Harvest” use case.

Summary:

Generate PID.

For each data file URL in metadata:

Download the file to the SIP area.

Create FILE entry.

Log EVT_FILE_DOWNLOADED or EVT_BROKEN_LINK_FOUND.

If broken link found, raise an alarm.

End-for

Create EPRINT entry.

Log a *capture* event.

Create relationships between e-print and files.

If e-print is a version of an existing e-print

Create relationship(s) between e-print versions

End-if

Post-conditions: The SIP area for the IR contains a directory for the e-print. This directory contains the metadata and any data files referenced by URLs in the metadata.

Alternative paths:

Notes:

- i) For EPrints, the data file URLs are contained in <file_url> elements; for DSpace, in <fileSec><fileGrp><file><FLocat> elements.
- ii) A PID (Persistent Identifier) is created at this point rather than on ingest into Fedora, as it will be required during the ingest process for associating events and relationships with the object. Unique PIDs will be generated using the generatePID method of the Fedora BasicPIDGenerator module. Depending on the Fedora content model used (see Section 5), it may be necessary to generate PID for file objects as well.
- iii) A generic interface should be implemented for alarms. The implementation could write to an alarm log and send e-mails (possibly using the RT system).
- iv) Suggestions for common PID scheme for Sherpa DP (TBD):
 - a) Create an identifier by combining an identifier of the IR with the identifier assigned by EPrints or DSpace, e.g. glaseprints:100, ERAeprints:200 (note that EPrints assigns a numeric id while DSpace assigns a handle).
 - b) In a database, keep track of the highest (numerical) identifier assigned by the system for each IR, and use that to generate subsequent identifiers. For example, if the highest id assigned for the Glasgow IR in the preservation archive is 100, when a new e-print is harvested, create a PID in the form glaseprints:101.
 - c) (preferred solution) The PID need not incorporate the id of the repository. Use Fedora PIDs, which are of form <namespace>:<sequential number>, e.g. ahds.ac.uk-sherpadp:15734.

4.4.4 Update AIP Metadata

Description: Update the original metadata associated with the specified AIP.

Preconditions: AIP exists in preservation repository.

Triggers: Preservation User

Summary:

Update the original metadata datastream of the AIP.

Log a *capture* event.

Post-conditions: The original metadata section of the AIP has been updated (with versioning).

Alternative paths:

Notes:

- i) The metadata is updated using the `ModifyDatastreamXMLMetadata` method of the Fedora API-M.

4.4.5 Check SIP Integrity

Description: Perform checksum verification and virus scan for data files in SIP. If a problem is detected, an alarm is raised. The Preservation User is responsible for contacting the IR and taking remedial action.

Preconditions: SIP created.

Triggers: Preservation User

Summary:

For each datastream in SIP:

Calculate checksum.

Log a *message digest calculation* event.

Compare calculated checksum with checksum in original metadata.

Log a *fixity check* event (the `eventOutcome` will contain the result).

If not successful, raise an alarm.

Perform virus scan on datastream

Log a *viruscheck* event (the `eventOutcome` will contain the result).

If not successful, raise an alarm.

End-For

Post-conditions: Either:

- i) The SIP has been successfully checked for data integrity (including freedom from viruses)
- ii) The SIP has failed at least one of the checks and at least one alarm has been raised.

Alternative paths:

Notes:

- i) MD5 checksums will be used.
- ii) If a file became infected with a virus after the IR checksum was calculated, then the checksum will have changed and so the virus check is superfluous. However, the infection may have occurred before the IR's checksum was generated. For virus checking, we can use the free software CLAMAV, which can be executed programmatically via a PERL script.

- iii) This is closely related to the “Check AIP Integrity” use case, which checks the integrity of an AIP in Fedora. This commonality should be taken into account during development.
- iv) This processing may be initiated automatically by the “Create SIPs” use case, depending on configuration. Also, the system could be configured to initiate “Generate DC Metadata” and “Generate Preservation Metadata for SIP” automatically on completion of this use case.
- v) The checksum algorithm and anti-virus software should be configurable.

4.4.6 Generate DC Metadata

Description: Generate Dublin Core metadata from the metadata harvested from the IR.

Preconditions: SIP created and validated.

Triggers: Preservation User.

Summary:

Create DC metadata file for the e-print.

Log an EVT_DC_GENERATED event.

Post-conditions: A DC metadata file has been generated for the SIP.

Alternative paths:

Notes:

- i) 2 XSLT stylesheets will be required for transforming each of the 2 IR metadata formats into DC. The element mappings are TBD.
- ii) The DC must conform to the OAI DC schema (this is a requirement of Fedora).
- iii) This processing may be initiated automatically by the “Check SIP Integrity” use case, depending on configuration.
- iv) This processing is not strictly necessary, as any metadata mapped to DC will also be contained in the original metadata. However, Fedora automatically creates a minimal set of DC metadata that is indexed automatically, so we may as well create a minimal common set for all e-prints stored.

4.4.7 Generate Technical Metadata for SIP

Description: Generate technical metadata for all original data files in a SIP.

Preconditions: SIP created and validated.

Triggers: Preservation User.

Summary:

For each data file in the SIP:

Execute “Generate Technical Metadata for File” use case.

End-For

Log an EVT_TECH_MD_GENERATED_FOR SIP event.

Post-conditions: Technical metadata has been generated for each data file.

Alternative paths:

Notes:

- i) Currently, the tools available do not allow all the required metadata elements to be generated automatically. Once the metadata has been generated for a SIP, the preservation user will be able to review the generated metadata for dubious values, make manual changes and complete blank elements. We could use the web interface to “force” the user to do this, by obliging the user to accept the generated data before being allowed to continue with the ingest process. Any manual changes made by the user must be recorded for audit purposes by logging an EVT_TECH_MD_MODIFIED event.
- ii) This processing may be initiated automatically by the “Check SIP Integrity” use case, depending on configuration.

4.4.8 Generate Technical Metadata for File

Description: Generate technical metadata for a single data file.

Preconditions: File exists.

Triggers: “Generate Technical Metadata for SIP” use case; Preservation User.

Summary:

Determine format of data file (see Note (ii)):

Use DROID make request to PRONOM for file format information.

Log a *validation* event.

If status of identification is not “Positive (Specific)” or “Positive (Generic)”

Raise alarm & terminate

End-if

Verify that the metadata generation tool(s) can handle the identified format (see Note (i)):

If JHOVE plugin not available for identified format

Raise alarm & terminate

End-if

Generate technical metadata for data file:

Execute JHOVE, passing the identified format as parameter.

Log a *validation* event.

If execution fails

Raise alarm & terminate

End-if

Create preservation metadata file(s) for data file:

Transform JHOVE metadata output to PREMIS (see Note (iii)).

Log an EVT_Tech_MD_Generated_for_File event.

If required for this file format (see Note (iv)):

Transform JHOVE output to format-specific technical metadata

Log an EVT_Tech_MD_Generated_for_File event.

End-if

Post-conditions: Preservation metadata has been generated for the data file.

Alternative paths:

Notes:

- i) Technical metadata corresponds to the object-level preservation metadata for the file, described in Section 3.2.
- ii) Initially, only JHOVE will be used as metadata generation tool. Plug-ins are available for a number of file formats.
- iii) JHOVE functions best when the format of the file is passed as a parameter. One possible (and simple) approach would be to assume as a first guess that the file suffix accurately indicates the file format - JHOVE would inform us if this assumption is wrong. However, the best approach available at present is to determine the format of the file from the PRONOM registry, using the DROID API (see <http://www.nationalarchives.gov.uk/aboutapps/pronom/droid.htm>). If this fails to recognise the file format, or if the file extension does not match the format proposed by DROID, then manual input will be required. The possible DROID status values are: *Positive (Specific)*, *Positive (Generic)*, *Positive (Specific) - Possible file extension mismatch*, *Positive (Generic) - Possible file extension mismatch*, *Tentative*, *Negative*.
- iv) The technical metadata will be based on the PREMIS proposals (see WP 4.4). XML schemas for PREMIS metadata must be created (see www.loc.gov/standards/premis/schemas.html for example schemas created by the Library of Congress). A number of PREMIS elements can be obtained from the XML output by JHOVE (see Section 3.2). An XSLT stylesheet will be required to transform the JHOVE XML output into PREMIS. Some elements, which cannot be extracted from JHOVE, will be left empty as place-markers.
- v) For some file formats, format-specific technical metadata will be stored. Initially, this will apply only to image files, in which case MIX metadata will be used.
- vi) It is likely that the tools and services used to determine file formats and generate preservation metadata will change as technology develops. The design must allow the tools/services (and the corresponding XSLT) used to be easily configurable to facilitate future changes.

- vii) This processing may be initiated automatically by the “Check SIP Integrity” use case, depending on configuration.

4.4.9 Normalise Datastreams for SIP

Description: Generate normalised version of data file for each input data file that is not in a format suitable for preservation.

Preconditions: SIP created and validated; technical metadata files for SIP generated.

Triggers: Preservation User.

Summary:

For each data file in the SIP:

Execute “Normalise Datastream” use case.

End-For

If any data file in the SIP required normalisation

Log a *normalization* event (for the e-print).

End-if

Create relationship(s) between e-print and normalised manifestation.

Post-conditions: A normalised file has been generated for each data file that requires it.

Alternative paths:

Notes:

- i) The preservation file formats corresponding to the main expected input file formats have been tentatively identified in WP 6.5, and as far as possible normalisation will be automated in accordance with this correspondence, which will be encoded using an XML configuration file.
- ii) The preservation formats proposed for the main expected deposit formats are PDF/A, OpenDocument XML, and SVG. Several software tools (e.g. Adobe Acrobat, Solid Converter PDF) exist that export PDF documents to PDF/A and other common formats, but they do not always produce expected results. In addition, an IR may export a file that is not in one of the expected formats. For these reasons, the user will be able to review the results of normalisation before proceeding to the next stage of ingest, so that manual changes can be applied where necessary. We could use the web interface to “force” the user to do this, by obliging the user to accept the normalised file before being allowed to continue with the ingest process.

4.4.10 Normalise Datastream

Description: Convert a data file to a normalised form, where necessary.

Preconditions: Technical metadata generated for data file.

Triggers: “Normalise Datastreams for SIP” use case; Preservation User.

Summary:

Extract file format from technical metadata for data file.

If conversion required for file format

Determine configured preservation format and conversion tool.

Perform conversion.

Log a *normalization* event (for the data file).

Create relationship(s) between original file and normalised file.

Create relationship(s) between e-print and normalised file.

Create relationship(s) between normalised manifestation and normalised file.

Else if file format acceptable for preservation

Create relationship(s) between normalised manifestation and original file.

Else

/* Unknown whether normalisation required or not */

Raise alarm and terminate.

End-if

Post-conditions: Normalised version of file created, if required.

Alternative paths:

Notes:

- i) The normalisation mappings are defined in WP 6.5. However, these are likely to change, and the transformation rules and conversion tools should be stored in an easily configurable way, such as an XML file. This should indicate:
 - the formats that are acceptable for preservation (and so do not require normalisation)
 - the formats that require normalisation, together with target format and conversion tool
- ii) If the system as configured is unable to carry out the processing automatically, manual conversion of data may be required. In this situation, event and relationship metadata must still be created.

4.4.11 Generate Technical Metadata for Normalised Files

Description: Generate technical metadata for normalised files.

Preconditions: File normalisation carried out.

Triggers: Preservation User.

Summary:

For each normalised file:

Execute “Generate Technical Metadata for File” use case.

End-For

Log an EVT_TECH_MD_GENERATED_FOR_NORM_FILES event.

Post-conditions: Technical metadata has been generated for each normalised file.

Notes:

- i) Additional user input will be necessary to accept the results (see “Generate Technical Metadata for SIP” use case).

4.4.12 Generate Fedora Ingest Package

Description: Create a METS package for ingesting the AIP into Fedora.

Preconditions:

- SIP created and validated;
- DC metadata generated for SIP;
- Any required normalised files generated;
- Technical metadata generated for all data files (including normalised files).

Triggers: Preservation User.

Summary:

Create a Fedora METS ingest package for the AIP. Options for the contents of the package are described in Section 5.5). Note that, depending on the option selected, there may be several METS documents associated with the AIP.

Log an EVT_FEDORA_INGEST_PACKAGE_CREATED event.

Post-conditions: A valid Fedora ingest package (comprising one or more METS documents) exists for the AIP.

Alternative paths:

Notes:

- i) Define template METS document(s) that can be completed programmatically. It may however be necessary to enter some elements by hand – this will not be clear until more detailed work has been done.
- ii) When creating the METS package, follow the guidelines in <http://www.fedora.info/download/2.0/userdocs/digitalobjects/rulesForMETS.html>
- iii) Once the Fedora ingest metadata has been generated, the user should be allowed to review it, and to make manual changes to it, before accepting it. We could use the web interface to “force” the user to do this, by obliging the user to accept the generated data before being allowed to continue with the ingest process.
- iv) It would be possible to use FOXML (the Fedora-specific XML format) to define the input package, rather than METS. METS was chosen for interoperability reasons.

4.4.13 Ingest AIP

Description: Ingest the AIP into Fedora.

Preconditions: Fedora METS ingest package created for AIP.

Triggers: Preservation User.

Summary:

Ingest AIP into Fedora.

Log an *ingestion* event.

Update the index to include the new Fedora object(s) and relationships.

Execute “Generate Ingest Report” use case.

Post-conditions: E-print exists as digital object(s) in Fedora archive.

Alternative paths:

Notes:

- i) Ways of ingesting objects into Fedora:
 - a) The IngestObject method of the Fedora API-M interface. This ingests a single Fedora object (corresponding to a single METS document).
 - b) The fedora-batch-ingest utility, which calls AutoBatchIngest. AutobatchIngest has problems with performance and memory consumption when more than 2500 object are to be ingested. However, such a scenario is unlikely with SherpaDP. As Ingest is in general expensive in terms of memory, an explicit call System.gc() (garbage collector) may help to reduce memory consumption.
 - c) The DirIngest extension. This is useful when ingesting a large number of objects that are arranged in a particular directory structure.

The most appropriate method will depend on circumstances. For the compound content model (see Section 5.5.1), where each e-print corresponds to a single Fedora object, (a) will be most appropriate. In the case of the Sherpa DP repository, even with an atomic content model an e-print will correspond to a small number of objects with a simple inter-object structure, and a number of invocations of (a) may still be the most appropriate method. For future extensions of the system in which more complex file structures are ingested, then (c), or an enhancement of it, should be investigated.
- ii) Level of re-indexing: a single item can be indexed separately and added to the index. However, there may be relations between items, e.g. the item may be a new “version” of an existing item, which may required more re-indexing to take place. The re-indexing should incorporate the new object(s) and all existing Fedora objects with which they have relationships in RELS-EXT.
- iii) The basic indexing mechanism in Fedora can be used to index the DC and RELS-EXT metadata only. Instead, use the Fedora Generic Search module, which is available as a plug-in (although it is currently a Beta version only). This plug-in allows any metadata datastream to be indexed.

4.4.14 Generate Ingest Report

Description: Generate an Ingest Report for the SIP.

Preconditions:

Triggers: “Ingest AIP” use case; “Create SIP” use case; “Check SIP Integrity” use case; Preservation User

Summary:

Generate an Ingest Report for the specified item, containing:

- IR name
- Original e-print id
- Harvest date/time
- Sherpa DP PID
- Ingest date/time (into Fedora)
- For each data file:
 - Original URL
 - Size
 - URL valid/invalid
 - Checksum check result
 - Virus scan result
 - Fedora URL of original file
 - Migration format (if any)
 - Fedora URL of migrated file
- Status and any error conditions

Store the report in the IR’s report directory.

Email report to the designated email address for the IR.

Post-conditions: Report stored in the IR’s report directory and emailed to the IR.

Alternative paths:

Notes:

- i) This report is generated when **either** the ingest is complete **or** an error occurs that is the responsibility of the IR and that the AHDS cannot fix, e.g. virus scan or checksum failure, broken data file URL, invalid metadata received from IR.
- ii) The list of fields is tentative. It should be discussed with the IRs to see what information they would like it to contain.
- iii) The (tentative) format of the report name:
ingest-<IR-Name>-<original-eprint-id>-<report-generation-datetime>.txt

4.5 Post-Ingest Use Cases

These use cases cover activities that taken place after ingest into Fedora, including preservation-related activities.

In some cases the processing overlaps the ingest use cases. The development must take this commonality into account.

4.5.1 Check Repository Integrity

Description: Perform integrity check on the repository.

Preconditions:

Triggers: Timer; Preservation user

Summary:

For each AIP that has not been checked for integrity for at least the configured period

 Perform “Check integrity of AIP” use case

End-for

Post-conditions:

Alternative paths:

Notes:

- i) The time of the last integrity check for a file can be determined from the event metadata, which will be stored within the Fedora object (and indexed).

4.5.2 Check Integrity of AIP

Description: Verify the checksums of all files in an AIP, both original and derived versions.

Preconditions: AIP ingested into Fedora.

Triggers: “Check Repository Integrity” use case; Preservation User.

Summary:

For each data datastream (original and derived):

 Calculate MD5 checksum.

 Log a *message digest calculation* event.

 Compare calculated checksum with checksum in original metadata.

 Log a *fixity check* event (the eventOutcome will contain the result).

 If not successful, raise an alarm.

End-For

Post-conditions: Either:

- i) The AIP has been successfully checked for data integrity
- ii) The AIP has failed the check and an alarm has been raised.

Alternative paths:

Notes:

- i) This is closely related to Use Case “Check SIP Integrity” (q.v.), which checks the integrity of a SIP on ingest, and will have processing in common.

- ii) It is not necessary to perform a virus scan, as there will be a checksum mismatch if a file has become infected with a virus since ingest.
- iii) The Fedora development team plans to incorporate checksum validation. This will not be implemented within the timescales of the initial Sherpa DP development, but our implementation should take their plans into account if possible.

4.5.3 Check Format Obsolescence

Description: Check obsolescence of file formats that are of interest to the repository.

Preconditions:

Triggers: Timer; Preservation user

Summary:

For each file format/version in the repository:

Check format for obsolescence (see Note (ii))

If format obsolescent

Log EVT_FORMAT_OBSOLESCENT event

For each non-deleted file in the repository of this format

Mark object as having obsolescent format

Raise alarm

Log a *validation* event (?????)

Log EVT_DATASTREAM_OBSOLESCENT event

End-for

End-if

End-for

Post-conditions:

Alternative paths:

Notes:

- i) As the repository is likely to contain a large number of files in a much smaller number of formats, performance may be improved by maintaining a list of all formats and versions used by objects in the repository. This has not been taken into account in the ingest use cases above.
- ii) Several remote format registries have been or are being set up to provide data on file format obsolescence, for example GDFR, PRONOM and FRED. Currently none of these registries offers an entirely satisfactory service, and the landscape is likely to change over the years to come. In particular, it may well turn out that different services are appropriate for different data formats. I suggest that for the moment we define an interface to our own service, which will in future marshal the results of these or other publicly available services, and also enable us to incorporate our own decisions about what formats should be superseded.

4.5.4 Migrate Datastream

Description: Migrate a datastream to a new format.

The target format will be specified by the user and passed as an input, as the mapping from source format to migration format may depend on the IR.

Preconditions:

Triggers: Preservation user.

Summary:

Generate migrated datastream (see Note (i)):

- Extract source datastream

- Transform source datastream to target format and save output.

Generate preservation metadata:

- Execute “Generate Technical Metadata for File” for migrated file

Update Fedora (the precise actions depend on the Fedora object model selected):

- Add new datastream

- Add technical metadata for new datastream

- Mark source datastream as obsolete.

- Log a *migration* event.

- Create relationship(s) between e-print and migrated datastream.

- Create relationship(s) between source datastream and migrated datastream.

- Create relationship(s) between migrated manifestation and migrated file.

Post-conditions:

Alternative paths:

Notes:

- i) It may be possible to add a disseminator that does the transformation directly by calling the appropriate tool.

4.5.5 Delete AIP

Description: Mark an e-print object as deleted.

Preconditions: The object corresponding to the e-print exists in the Fedora archive and is not marked as deleted.

Triggers: Preservation user.

Summary:

Mark the specified e-print object, and any dependent objects, as deleted in Fedora (using the DeleteObject API-M method).

Update REQUEST with processed_flag and processed_timestamp.

Log a *deaccession* event.

Send a notification to the IR's designated email address

Post-conditions: The object corresponding to the e-print is marked as deleted in Fedora, together with any dependent objects.

Alternative paths:

Notes:

- i) The IR users can request the deletion of an e-print (see use case “Request Item deletion”), in which case the request is placed in a queue and is processed by the preservation user (the current use case). The e-print is then no longer visible to the IR; however, the data is still held in the Fedora repository and is visible to the preservation user.
- ii) In the case where the report was requested by an IR user, the PA user will select the request from a list.
- iii) Depending on the Fedora object model selected, a single e-print may correspond to several Fedora objects. All of these must be marked as deleted.

4.5.6 Generate DIP

Description: Generate a DIP (Dissemination Information Package) for an e-print.

Preconditions: AIP is in Fedora repository.

Triggers: Preservation user.

Summary:

Generate DIP for selected e-print. This will consist of:

- i) For each of the e-print's constituent files, either the original file (if it has not been migrated) or the latest migration.
- ii) The most recent version of the “original metadata” datastream.

Update REQUEST with processed_flag and processed_timestamp.

Log a *dissemination* event.

Perform “Generate Dissemination Report” use case.

Post-conditions:

- i) The IR's dissemination directory area contains the requested DIP.
- ii) The dissemination report is in the IR's report area, and has been emailed to the IR.

Alternative paths:

Notes:

- i) The IR users can request a DIP for an e-print (see use case “Request DIP”), in which case the request is placed in a queue and is processed by the preservation

user (the current use case). The preservation user can also generate a DIP independently, for example if the file formats of the AIP have been migrated.

- ii) In the initial implementation, Sherpa DP will zip up the files and metadata and make the zip file available for download by the IR, which can access the downloads via the web interface. In a later version of the system, Sherpa DP may generate files in a format suitable for direct ingest into EPrints/DSpace.

4.5.7 Generate Dissemination Report

Description: Generate a Dissemination Report.

Preconditions: The AIP is in the preservation repository.

Triggers: “Generate DIP” use case.

Summary:

Generate a dissemination report for the specified DIP, containing:

- IR
- Original id of e-print
- Request timestamp
- Report generation timestamp
- Comments from the preservation staff
- For each file:
 - Filename
 - Size
 - Format
 - Status (original/migrated)

Store the report in the IR’s report directory.

Log an EVT_DISSEMINATION_RPT event.

Email report to the designated email address for the IR.

Post-conditions:

- i) The IR’s report area contains a dissemination report for the DIP.
- ii) The report has been sent to the IR’s designated email address.

Alternative paths:

Notes:

- i) The list of fields is tentative. It should be discussed with the IRs to see what information they would like it to contain.
- ii) The (tentative) format of the report name:
DIP-<IR-Name>-<original-eprint-id>-<report-generation-datetime>.txt

4.5.8 Generate Statistics Report

Description: Generates a statistics report for an IR.

Preconditions: The specified request exists in the REQUEST table.

Triggers: Timer (configurable); Preservation User.

Summary:

Generate statistics report for the specified IR and reporting period, containing:

- Start/end of reporting period
- Number of e-prints pulled during reporting period
- Number of e-prints whose format has been migrated during reporting period
- Number of e-prints that failed integrity check (new & periodic) during reporting period
- Number of DIPs requested/generated during reporting period
- Average file transfer to AHDS from IR per week during reporting period
- Total storage utilisation in the preservation archive by IR (size & number, broken down by format)

Output report to the IR's report directory, and email it to the IR's designated email address..

If report requested by user

Update REQUEST with processed_flag and processed_timestamp.

End-if

Log an EVT_STATISTICS_RPT event.

Post-conditions:

Alternative paths:

Notes:

- i) Report generation frequency can be configured for a repository in the config.properties file. The report can also be requested directly by the Preservation User.
- ii) In the case where the report was requested by an IR user, the PA user will select the request from a list.
- iii) Precise content and format of the report should be discussed with IRs.

5. Component Descriptions – Fedora

5.1 Content Modelling Options

At an abstract level, the preservation repository contains the following entities:

- collections (i.e. all e-prints corresponding to an individual institutional repository participating in Sherpa DP)
- e-prints
- files, either original files captured from the IRs, or normalised/migrated files created by the AHDS.
- manifestations of an e-print, either the original manifestation, the preservation manifestation (which may be the same as the original manifestation), and migrated manifestations (if any). A manifestation is defined by the set of files that belong to it, and the type of manifestation (original, preservation, migrated).

Not all of these need be represented as objects in Fedora. Collections correspond to IRs, and in Fedora will correspond to the set of all e-prints originating from the IR. This can be determined from an e-print object's metadata. No further metadata or information about the IR will be stored within Fedora for the moment, although in future we could model the IRs as agents.

Among Fedora users there are two main approaches to representing digital objects, usually referred to as “atomistic” or “compound” models. In the former, a data object with multiple content components is represented in Fedora by multiple objects, which are linked by means of relationships encoded in the Fedora metadata. This simplifies the internal object structure at the cost of increasing the number of objects and complicating the model. In the compound model, such a data object is represented as a single Fedora object containing multiple content datastreams, resulting in fewer objects having a more complex internal structure. In the case of Sherpa DP, the internal structure of an e-print will be relatively simple, in the majority of cases consisting of a single file, in which case the distinction becomes rather pedantic. However, an aim of the design should be to make the system easily extensible, for example to other types of resource, so the model used should aim to be of more general application.

One major advantage of atomistic models is that they enable wider discovery and re-use of content in different contexts. However, as Sherpa DP is intended only as a preservation archive, this is of little relevance. A more practical advantage of atomistic models is that Fedora has better built-in support for modelling and querying relationships between Fedora objects than it does for relationships between datastreams within a Fedora object, and thus makes preservation activities easier to manage. In order to base the final decision on a more secure foundation, it is proposed to prototype and assess both forms of content model during development. Below we give a description of the Fedora METS packages to be created for each case.

5.2 Expressing Relationships in Fedora

An important component of the PREMIS preservation metadata to be represented in the preservation repository is that which concerns relationships between data objects or their

components (see Section 3.4). Non-relational metadata can be represented quite easily in Fedora as inline XML datastreams, but the representation of relational metadata is more complex. Fedora provides two pre-defined metadata streams specifically intended to allow relational metadata to be stored, in the form of as RDF assertions: RELS-EXT and RELS-INT.

5.2.1 RELS-EXT

The RELS-EXT metadata section in Fedora allows relationships between objects to be expressed in RDF, in such a way that they can be indexed and queried (using the query languages RDQL or ITQL). For example:

```
<rdf:Description rdf:about="ahds.ac.uk-sherpadp:10">
  <fedora:hasPart rdf:resource="ahds.ac.uk-sherpadp:100"/>
</rdf:Description>
```

expresses the relationship that the object with PID "ahds.ac.uk-sherpadp:10" has the relationship "hasPart" to the object with PID "ahds.ac.uk-sherpadp:100" (where, say, ahds.ac.uk-sherpadp:10 corresponds to an e-print and ahds.ac.uk-sherpadp:100 to a constituent file). However, there are certain restrictions imposed by Fedora on the RDF used in this section (for details, see <http://www.fedora.info/download/2.1.1/userdocs/digitalobjects/introRelsExt.html>, Section 4), which make it difficult to associate the linking event with the relationship within the RELS-EXT section.

The following are possible approaches to this problem:

i) Represent the relationships as part of the object-related preservation metadata, following the structure in the PREMIS Data Dictionary. This has the disadvantage of abandoning the very useful features of RELS-EXT and RDF, and making searches over relationships significantly more difficult.

ii) What we really want to assert is 2 things – the relationship and the fact that the relationship is associated with an event, e.g.

```
<rdf:Description rdf:about="ahds.ac.uk-sherpadp:10">
  <fedora:hasPart rdf:ID="reln1234" rdf:resource="ahds.ac.uk-sherpadp:100"/>
</rdf:Description>
<rdf:Description rdf:about="reln1234">
  <myns:hasEvent rdf:resource="event1234"/>
</rdf:Description>
```

where rdf:ID="reln1234" represents the relationship. However, the RELS-EXT rules don't allow this.

Alternatively, we could use an rdf:ID attribute in the property to link the relationship directly to the event, e.g.

```
<rdf:Description rdf:about="ahds.ac.uk-sherpadp:10">
  <fedora:hasPart rdf:ID="event1234" rdf:resource="ahds.ac.uk-sherpadp:100"/>
</rdf:Description>
```

where the value of `rdf:ID` is the identifier of the linking event. However, this is not the intended use of the `rdf:ID` attribute.

iii) Include a datastream (say, RELS-EVTS), which contains the associations of relationships to events. For example, RELS-EXT might contain:

```
<rdf:Description rdf:about="ahds.ac.uk-sherpadp:10">  
  <fedora:hasPart rdf:ID="reln1234" rdf:resource="ahds.ac.uk-sherpadp:100"/>  
</rdf:Description>
```

and RELS-EVTS:

```
<rdf:Description rdf:about="reln1234">  
  <myns:hasEvent rdf:resource="event1234"/>  
</rdf:Description>
```

The former RDF fragment is valid in RELS-EXT. This approach enables the full relationship metadata to be stored, and the RELS-EXT component to be searched. However, the search mechanism will not be able to link up the two sections without some additional software development.

5.2.2 RELS-INT

The RELS-INT metadata is intended to represent relationships between datastreams within a single Fedora object, much as RELS-EXT represents external relationships. However, although the datastream is present and can be used to represent these relationships, in the current version of Fedora (2.1) it is neither indexed nor validated, other than as valid XML.

5.3 E-print Versioning

When an IR modifies an e-print that already exists in the preservation archive, Sherpa DP will harvest the modified e-print as part of its standard synchronisation mechanism (see [4]). The updated e-print will have a new identifier in the IR, and its metadata will indicate its relationship with the earlier version. The precise mechanism for indicating this relationship is not clear from the metadata examples, and will have to be determined before the software is developed.

Sherpa DP will treat the updated e-print as a new digital object, with a new PID. The Fedora object corresponding to the e-print will incorporate the original metadata, and thus will contain the relationship between IR identifiers and Sherpa DP PIDs. The relationship between the updated e-print and its earlier version will be represented in the RELS-EXT section of the Fedora object.

If an update to an e-print at the IR involves only the metadata (i.e. not the actual e-print content), it is assumed that the IR does not treat this as a new e-print, and so will use the same identifier. Again, this assumption must be verified with the IRs.

5.4 Correspondence with OAIS Model

The major data objects in the OAIS model are the SIP (Submission Information Package), AIP (Archival Information Package) and DIP (Dissemination Information Package). In the Sherpa DP repository, these objects correspond to the following sets of data:

OAIS Model	Sherpa DP
SIP	<p>All information for an e-print obtained from the IR, comprising:</p> <ul style="list-style-type: none"> • the metadata record (extracted from the harvested metadata). • all content files referenced by the metadata record. <p>In some cases, there will be no content files, for example if the item is just a bibliographic reference.</p>
DIP	<p>Data for a specified e-print record fetched from the preservation archive at the IR's request, comprising:</p> <ul style="list-style-type: none"> • the original metadata record (or the latest version of it, if it has been updated). • for each constituent data file, either the original file (if it has not been migrated) or its most recent migrated version. This corresponds to the latest migrated manifestation of the e-print generated by the preservation archive.
AIP	<p>All the data required for managing the e-print within the Sherpa DP preservation repository. This includes all original metadata and data, any normalised or migrated data files, preservation metadata, DC metadata for the e-print, and Fedora-specific data and metadata. This is shown schematically in Figure 4, although the precise allocation of data to Fedora digital objects will depend on the content model chosen.</p>

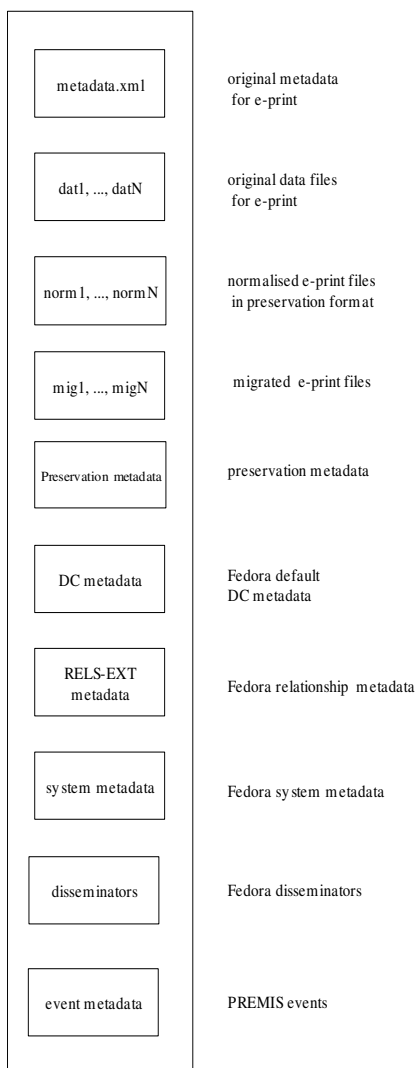


Figure 4 - Fedora Ingest Package

5.5 Fedora Ingest METS packages

In this section we describe the METS document(s) required for ingesting an e-print into the Fedora repository (see “Generate Fedora Ingest Package” use case). Two cases are considered, corresponding to the 2 approaches to content modelling described above:

- Option 1: Compound model, in which a single Fedora object correspond to an e-print
- Option 2: Atomistic model, in which multiple Fedora objects correspond to a single item

In each case the description contains an outline of the sections contained in the document(s), and in particular describes how the PREMIS preservation metadata (technical, event and relationship) will be represented therein. The descriptions concentrate on the data - disseminators have not been specified at this stage.

These descriptions will form the basis for template METS documents that will be used in the working system, although given the scant documentation they are likely to be revised during implementation. In particular, the RDF fragments included are for illustrative purposes only.

5.5.1 Option 1: single Fedora object per e-print

The Fedora ingest package will consist of a single METS document, containing the following sections:

- 1) <METS:dmdSec> element group containing DC metadata (as generated above)
- 2) <METS:amdSec>/<METS:sourceMD> element containing the original metadata
- 3) <METS:amdSec> element corresponding to each manifestation.
- 4) <METS:fileSec>/<METS:fileGrp>/<METS:fileGrp>/<METS:file> element for each data file (original or derived). [Note: the first <METS:fileGrp> groups together the different datastreams, the second versions of the same datastream – in our case however there will only be one version of a datastream].
- 5) <METS:amdSec>/<METS:techMD> element for each data file (original or derived), containing PREMIS technical metadata (excluding relational and event metadata)¹².
- 6) <METS:amdSec>/<METS:techMD> element for each data file (original or derived) that requires format-specific technical metadata
- 7) <METS:amdSec>/<METS:techMD> elements containing PREMIS event metadata. There will be one of these for each PREMIS object with which events may be associated, namely: the e-print; each manifestation of the e-print; each constituent data file (original or derived).
- 8) <METS:amdSec>/<METS:techMD> element containing RELS-EXT metadata (this will represent the relationships with other e-print objects corresponding to versions of this e-print)
- 9) <METS:amdSec>/<METS:techMD> element containing a RELS-INT datastream. This will define the relationships internal to the Fedora object. These will include:
 - (a) relationships between PREMIS objects (see Section 3.4), for example between a migrated datastream and its source.
 - (b) relationships that are contingent on this particular implementation, for example between a file datastream and its technical metadata datastreams.
- 10) <METS:amdSec>/<METS:techMD> element containing RELS-EVTS datastream. This will define the associations between RELS-EXT relationships and events.
- 11) disseminators – details TBD, although a standard set of disseminators is likely to be used for each object, in which case this section of the package will be the same in each case. In particular, there will be a disseminator to produce a DIP.

The PREMIS relationship metadata maps onto the above structure as follows:

¹² METS:techMD elements for a given file should be grouped in a single METS:amdSec element.

1) The relationships between an e-print and each of its constituent data files (original or derived) is implicit, as the file datastreams are contained within the Fedora object corresponding to the e-print.

2) The relationship between a derived datastream and its source datastream is recorded in RELS-INT, e.g.

```
<rdf:Description rdf:about="PID/DS15">
  <ahds-rels:isDerivedFrom rdf:resource="PID/DS6">
    <premis:relatedObjectIdentifierSequence>5</premis:relatedObjectIdentifierSequence>
    <premis:relatedEventIdentifierValue>5927</premis:relatedEventIdentifierValue>
</rdf:Description >
```

relationshipType and relatedEventIdentifierType are implicit and do not need to be stored.

3) The relationships between different versions of the “same” e-print (e.g. pre- and post-print versions) are recorded in RELS-EXT, e.g.

```
<rdf:Description rdf:about="PID15">
  <ahds-rels:hasVersion rdf:resource="PID26">
</rdf:Description>
```

4) The relationships between an e-print and each of its manifestations are implicit, as the datastreams corresponding to the manifestations are contained within the Fedora object corresponding to the e-print.

5) The relationships between a manifestation and its constituent datastreams are recorded in RELS-INT, e.g.

```
<rdf:Description rdf:about="PID/DS64">
  <fedora:hasPart rdf:resource="PID/DS122"/>
  <premis:relatedEventIdentifierValue>2736</premis:relatedEventIdentifierValue>
</rdf:Description>
```

5.5.2 Option 2: multiple Fedora objects per e-print

The Fedora ingest package will contain the following METS documents:

- 1) one for the item
- 2) one for each manifestation (on ingest, this will be just the original and the preservation)
- 3) one for each data file (original or derived)

The top-level METS document will contain:

- 1) <METS:dmdSecFedora>/<METS:dmdSec> element containing DC metadata (generated from the original metadata, as described above)
- 2) <METS:amdSec>/<METS:sourceMD> element containing the original metadata
- 3) <METS:amdSec>/<METS:techMD> elements containing PREMIS event metadata (for events associated with the item).

- 4) <METS:amdSec>/<METS:techMD> element containing RELS-EXT metadata (see below).
- 5) <METS:amdSec>/<METS:techMD> element containing RELS-INT metadata (see below).
- 6) disseminators – details TBD.

The manifestation METS documents will contain:

- 1) <METS:dmdSecFedora>/<METS:dmdSec> element containing DC metadata (the minimal Fedora DC)
- 2) <METS:amdSec>/<METS:techMD> elements containing PREMIS event metadata (for events associated with the manifestation).
- 3) <METS:amdSec>/<METS:techMD> element containing RELS-EXT metadata (see below).
- 4) <METS:amdSec>/<METS:techMD> element containing RELS-INT metadata (see below).
- 5) disseminators – details TBD.

The data file METS documents will contain:

- 1) <METS:dmdSecFedora>/<METS:dmdSec> element containing DC metadata (the minimal Fedora DC)
- 2) <METS:fileSec>/<METS:fileGrp>/<METS:fileGrp>/<METS:file> element for each data file (original, normalised or migrated)
- 3) <METS:amdSec>/<METS:techMD> element containing PREMIS technical metadata (excluding relational and event metadata)
- 4) <METS:amdSec>/<METS:techMD> element containing PREMIS technical metadata (excluding relational and event metadata)
- 5) <METS:amdSec>/<METS:techMD> element containing format-specific technical metadata (optional – depends on format of data file).
- 6) <METS:amdSec>/<METS:techMD> element containing RELS-EXT metadata (this will represent the relationships with other e-print objects corresponding to versions of this e-print)
- 7) <METS:amdSec>/<METS:techMD> element containing RELS-INT metadata (see below)
- 8) disseminators – details TBD.

PREMIS relationship metadata maps onto the above structure as follows:

- 1) Between an e-print and each of its constituent datastreams (original or derived) – in the RELS-EXT section of e-print and/or file object.
- 2) Between a derived datastream and its source datastream – in the RELS-EXT section of original and/or derived file object.

- 3) Between different versions of the “same” e-print (e.g. pre- and post-print versions) – in the RELS-EXT section of the e-print object(s).
- 4) Between an e-print and each of its manifestations – in the RELS-EXT section of e-print and/or manifestation object
- 5) Between a manifestation and its constituent datastreams – in the RELS-EXT section of manifestation and/or file object

The RELS-INT datastream will be used to represent only relationships that are internal to an object. For example, the fragment

```
<rdf:Description rdf:about="PID/DS15">  
  <ahds-rels:isTechnicalMetadataFor rdf:resource="PID/DS6">  
</rdf:Description >
```

relates a file datastream and its corresponding technical metadata datastream.

6. Database specification

The majority of ancillary information about the e-prints and files stored in the repository is held as XML-formatted metadata within the Fedora objects themselves, so that it is closely linked to the object with which it is associated. In certain cases, however, data will be stored in a relational database external to Fedora, as described in Section 2.3.

This section contains a logical view of this relational database. Further details will be added during implementation.

6.1 Logical database design

The following sections describe the main tables in the database and their structure.

6.1.1 User

Stores details of IR and Preservation users.

Column name	Format	Comments
username	string	username and password for logging on via web interface.
password	string	
name	string	Full name of user
email	string	Email address of user
telephone	string	Telephone number of user
disabled_flag	boolean	Set to true if user has been disabled.
IR_id	string	User's home IR
role_id	string	User's role

6.1.2 Role

Lists the available roles in the Sherpa DP system. These control what functions are available. We could add another table mapping roles to functions, but this may be excessive. Ultimately, the availability of functions to roles will be implemented using Shibboleth. In the short term, it will probably be sufficient to have two roles only, corresponding to IR users and preservation archive users.

Column name	Format	Comments
role_id	string	Uniquely identifies role.
role_name	string	Longer textual description of role

6.1.3 Collection

Holds information about each collection (which corresponds 1-1 to an IR)

Column name	Format	Comments
IR_id	string	Identifies IR within Sherpa DP
IR_long_name	string	Long name of IR.
contact_name	string	Details of designated contact at IR.

contact_address	string	
contact_email	string	
contact_telephone	string	
archive_url	string	OAI provider URL.

6.1.4 Harvest

Holds information about each harvest from each IR.

Column name	Format	Comments
IR_id	string	
metadata_pathname	string	Location of harvested record on local file system.
num_items	integer	Number of separate items in harvested record.
processed_flag	boolean	Whether all items have been successfully extracted from the harvested record.
creation_timestamp	timestamp	Time record created.

6.1.5 Eprint

Holds information about each e-print captured.

Column name	Format	Comments
IR_id	string	
original_id	string	Identifier or handle used by IR (unique within IR)
PID	string	PID assigned by Fedora. This should be included only if it is possible to generate the PID prior to Fedora ingest (see Section 4.4.3).
metadata_pathname	string	Location of metadata record on local file system.
creation_timestamp	timestamp	Time record created.

6.1.6 File

Holds information about each data file captured.

Column name	Format	Comments
IR_id	string	Identifies the e-print to which the file belongs.
original_id	string	
data_pathname	string	Location of file on local file system.
creation_timestamp	timestamp	Time record created.

6.1.7 Request

Holds requests submitted by IR staff.

Column name	Format	Comments
username	string	Identifies who made the request.
IR_id	string	The requester's home IR – not strictly necessary as username will be unique within Sherpa DP system.
request_type	string	DIP, Report, Removal.
request_parameters	string	The parameters to the request (comma separated). Number and type will depend on the type of request.
request_id	integer	Unique identifier of the request.
request_timestamp	timestamp	Time request was made.
processed_flag	boolean	Whether the request has been serviced.
processed_timestamp	timestamp	Time request was serviced.

6.1.8 Event

Holds system audit information. Note: Some events are stored as PREMIS events, in which case they are held as XML within Fedora objects, not in the database.

Column name	Format	Comments
event_id	integer	Unique identifier of the event.
event_type	integer	
event_timestamp	timestamp	
event_parameters	string	Event parameters – format depends on event_type.