



D2.3.2 DEPLOYED PRESERVATION SERVICES

Advanced Search Services and Enhanced
Technological Solutions for the European Digital
Library

Grant Agreement Number: 250527

Funding schema: **Best Practice Network**

Deliverable D2.3.2 WP2.3

Prototype
v1.0 – March 29, 2012
Document. ref.: ASSETS.D2.3.2.ENG.WP2.3.V1.0

	2012			and User/Developer manual sections.
0.6	23-february-2012	Draft	RG(AIT)	Revised Risk Management Service sections.
0.7	27-february-2012	Draft	MN(ENG)	Re-organized contents in three main sections (one for each task of the WP 2.3).
0.9	28-february-2012	Draft	MN(ENG)	Peer review
0.10	1-March-2012	Draft	RG (AIT)	Peer review
0.11	5-March-2012	Draft	AL (AIT)	Peer review
0.12	5-March-2012	Draft	RG (AIT)	Peer review and updates
0.13	15.March-2012	Draft	MN (ENG)	Revision based on feedback
0.14	23-March-2012	Pre-Final	SG (AIT)	Final review
1.0	29-March-2012	Final	LB (ENG)	Approval and Release

Table of Contents

1. INTRODUCTION	2
1.1 THE ASSETS APPROACH AND PROPOSAL	2
2. T2.3.1 RISK MANAGEMENT	6
2.1 INTRODUCTION	6
2.2 BUSINESS SCENARIOS FOR RISK MANAGEMENT	6
2.3 TECHNICAL DOCUMENTATION FOR RISK MANAGEMENT	9
2.3.1 UML diagrams	9
2.3.2 REST services	13
2.3.3 Risk Management: Client APIs	17
2.3.4 Software packaging	20
2.3.5 Installation and configuration	20
2.4 USER/DEVELOPER MANUAL FOR RISK MANAGEMENT	21
2.4.1 USER MANUAL (how to use risk management service home pages index.html and admin.html)	21
2.4.2 DEVELOPER MANUAL	22
3. T2.3.2 PRESERVATION NORMALIZATION	25
3.1 INTRODUCTION	25
3.2 BUSINESS SCENARIOS FOR NORMALIZATION	25
3.3 TECHNICAL DOCUMENTATION FOR NORMALIZATION	26
3.3.1 UML diagrams	26
3.3.2 Web services	33
3.3.3 Software packaging	42
3.3.4 Installation and configuration	44
3.4 USER/DEVELOPER MANUAL FOR NORMALIZATION	44
4. T2.3.2 PRESERVATION NOTIFICATION	57
4.1 INTRODUCTION	57
4.2 BUSINESS SCENARIOS FOR NOTIFICATION	57
4.3 TECHNICAL DOCUMENTATION FOR NOTIFICATION	57
4.3.1 UML diagrams	57
4.3.2 REST services	58
4.3.3 Preservation Notification : Client APIs	63
4.3.4 Software packaging	69
4.3.5 Installation and configuration	69
4.4 USER/DEVELOPER MANUAL FOR NOTIFICATION	71
4.4.1 USER MANUAL (how to use notification service home page, index.html)	71
4.4.2 DEVELOPER MANUAL	72
5. CONCLUDING REMARKS	76
6. REFERENCES	77

Table of Figures

Figure 1 - The OAIS reference model	2
Figure 2 - The ASSETS Digital Preservation Services	5
Figure 3 - Risk Management Domain Objects (Scenario 1).....	10
Figure 4 - Linked open data (LOD) domain objects (Scenario 2).....	11
Figure 5 - Risk Management interfaces: client side	12
Figure 6 - Risk Management: server side	13
Figure 7 - Available risk mangement administration services required for scenario 2 to initialize and check database	14
Figure 8 - Available risk management user services	15
Figure 9 - Format Registry Concepts and Domain Object	28
Figure 11 – Standard Data Exchange Objects	30
Figure 12 – Jasper Migration Service Sample	31
Figure 13 – OSGi Service Utilities	32
Figure 14 – Service Framework Registration Approach	33
Figure 15 - Preservation Notification: domain models	58
Figure 16 - Preservation Notification: service interfaces	58
Figure 17 - Available notification services related to managing external services and taxonomies.....	59
Figure 18 - Available notification services related to managing subscribers and subscriptions	60
Figure 19 - Available notification services related to managing publishers, creating notification channels and publishing messages.....	62
Figure 20 - Available notification services related to managing the delivery of messages.	63

Executive Summary

This report is the release note for the final prototypes of the Preservation Services suite (i.e. Risk Management, Normalization, Preservation Notification).

The specification and the detailed uml models have been provided for the three services in the [D2.0.4] "The ASSETS APIs" for the first year of the project.

During the second year of the ASSETS project, the steps of the development lifecycle (in particular integration and evaluation) allowed to refine prototypes by improving performances, usability, documentation and supported features.

Consequently, this report describes both the final specification and the updated models of the deployed Preservation Services.

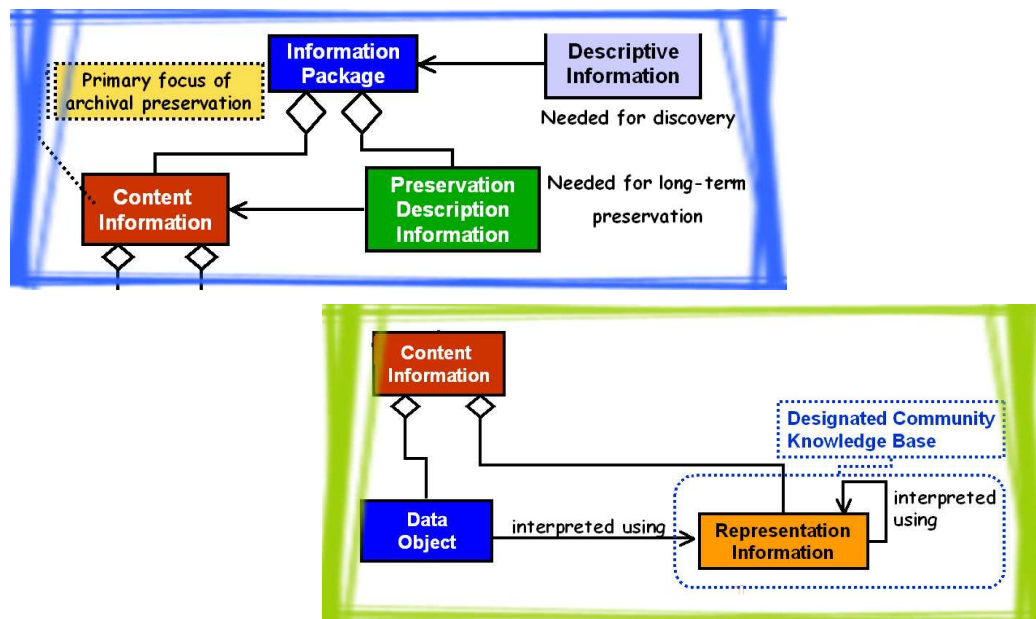
This document provides detailed information on how to deploy, instantiate and use the services for supporting implementation of preservation scenarios.

The document is available as a public document (license CCby-nc-sa).

OAIS (*Open Archival Information System*) Reference Model (see Figure 1) is the ISO standard (14721:2003) adopted for addressing the digital preservation challenges. It identifies and defines a common framework in order to analyse and describe concepts and terminology for Digital Archives/Libraries. The major aim of OAIS Reference Model is to facilitate a much wider understanding of what is required to preserve information for the long term. In particular, OAIS has defined the following key models and guidelines: i) the information model, ii) the archive responsibilities, iii) functional model.

The information model defines how the content information and its data object, primary focus of the long-term preservation, have to be described and packaged (i.e. Submission/Archival/Dissemination Information Package) within the digital archive/library.

The following figures show the key concepts such as Information Package, Content Information, Data Object, Preservation Description Information and Representation Information. In principle, it is feasible to assert that the Preservation Description Information (PDI) provides the suitable information for enabling the long-term preservation (i.e. preservation metadata): it includes details about the “provenance” which deals with the history of creation, ownership, accesses and changes of the content.



Another key concept is the Representation Information which allows the users of a Designated Community to interpret the Content Information and its Data Object. It includes two main type of information: the semantic representation and the structural representation. The latter deals with information such as the Data structure, Format, tools for accessing the information and data.

The Information Package is managed by an OAIS Archive through 6 Functional Blocks: the Ingestion, the Access, the Administration, the Data Management, the Archival Storage and the Preservation Planning. OAIS clarifies which are the roles and responsibilities of those functional blocks, and in particular it remarks that it is important to:

- Identify and characterise events which potentially impact the long-term usability and access to digital information - This deals with the evaluation of potential accessibility risks;

- Monitor occurring events which potentially impact the long-term usability and access to digital information;
- Identify, characterise and evaluate relative corrective actions/plans for mitigating impacts (preservation plans) - This deals with the plans for reducing impacts of risks;
- Communicate/notify impacting events and actions to the right actors involved in the preservation process - This deals with the notification/communication of risks to the actors who are responsible for taking the right decisions and enacting corrective actions;
- Enact, monitor and control relative corrective actions - This deals with the enactment/reaction for mitigating impacts of risks;
- Track, report and document occurring events/actions/changes within the digital archive, as evidence for judging/auditing/certifying the quality of “preservation archive” and the “authenticity/provenance/integrity” of archival objects - This deals with the reporting aspect, which is useful for tracking changes.

OAIS “*Preservation Planning and Administration*” components provide the functionality described above, as shown in the following table.

OAIS Responsibilities	
PLANNING	ADMINISTRATION
<ul style="list-style-type: none"> • Monitoring OAIS environment; • Detect changes/impacts in DCKB (Designated Community Knowledge Base); • Mapping out preservation Strategy; • Provide recommendations. 	<ul style="list-style-type: none"> • Manage submission agreements; • Audit submission; • Maintain configuration management; • Monitor archive operations; • Inventory archive content; • Report on archive content; • Migrate/update archive content; • Manage archive standards/policies.

The ASSETS proposal for the digital preservation issue is to provide implementations of services whose functionalities are defined in the OAIS Preservation Planning and in the OAIS Administration components. In practice, the ASSETS Digital Preservation Services (see Figure 2):

- estimate data preservation risk through the Risk Management service
- track and report/notify occurring events through the Notification service
- enact preservation plans through the Normalisation service.



Figure 2 - The ASSETS Digital Preservation Services

2. T2.3.1 Risk Management

2.1 Introduction

The service aims at mitigating the risk of digital obsolescence by providing risk management reports to content providers, after performing an analysis of the contributed content.

The service performs an inspection of the collection objects and a statistical analysis of the content formats in order to provide a categorization based on the preservation risk.

The aims at providing a reliable identification of long term accessibility risks for the underlying data.

This component addresses the topics of technology watch and enactment of (semi)automated preservation policies. It makes use of available preservation community resources such as technical registries (like PRONOM¹) for policy extraction and the ASSETS Normalisation service for object identification and policy execution.

The service addresses the following issues:

- format obsolescence and limited support for proprietary formats;
- pour technical documentation of digital collections;
- automated collection profiling and recommendation of preservation actions.

2.2 Business scenarios for Risk Management

The Preservation Risk Management covers a very important role by providing metadata analysis and preservation risks estimation for Europeana collections.

The objectives of the service are:

- Evaluation of the Europeana collection metadata statistics. The metadata statistics are represented by parameters availability of information in important fields, accessibility of links, item count. The URI links available in Europeana objects (i.e. “EuropeanalsShownBy” or “EuropeanalsShownAt”) should be examined for their accessibility. Retrieved metadata statistics could be aggregated in relevant preservation dimensions like provenance, context and accessibility. Based on the retrieved metadata statistics and generated preservation dimensions the service computes an aggregated preservation risk scores, identifying collections that could be in danger of not being prepared for supporting long term data accessibility.
- Preservation risk estimation for selected Europeana collections. The preservation risk estimation is based on the analysis of the metadata collections in combination with the analysis of the file formats used for representing the media information. The system uses the external data repositories in order to evaluate the long term preservation compliance for a particular file format. To get more information about linked open data (LOD) repositories, please use the following links:

¹ <http://www.nationalarchives.gov.uk/PRONOM/Default.aspx>

- DBPedia: <http://dbpedia.org>
- Freebase: <http://www.freebase.com>
- Pronom: <http://www.nationalarchives.gov.uk/PRONOM>

The following business scenarios were taken in account for specifying requirements for this component:

1. Preservation risks evaluation for Europeana collections. The main idea of this scenario is to statistically analyse the quality of Europeana collection and to estimate preservation risks. The metadata fields of each Europeana object are examined to evaluate their completeness and/or correctness. The URI links included in metadata are checked for their accessibility. The collected information is stored into the Assets database. This data is used as basis for computation of risk scores and identification of possible risks for information loss. The metadata of the Europeana collection objects are statistically analysed; preservation risk scores are calculated from metadata statistics and aggregated preservation dimensions. The output of this scenario is a generated preservation risk report in HTML format. The steps performed for completing this scenario are presented in the following:
 - Ingestion of collection represented in ESE XML format (if not already done). This is required to provide access to the collection metadata through the rest interface.
 - Define configuration for metadata analysis. In this step it is possible to define which collection should be analysed (by id or collection name), which metadata fields are interesting for the metadata analysis and which risk computation model should be applied. The risk computation model defines the composition of the preservation dimensions and the thresholds indicating preservation risks. A default configuration file is provided with the component, and is recommended to be used for evaluating the metadata collections within the context of Assets project.
 - Perform statistical analysis on metadata fields. In this step we iterate over all Europeana objects in a given collection and collect statistical information. This data is stored in a database for further computations.
 - Compute quantification of metadata analysis results over different preservation dimensions. The preservation risk scores are computed over the preservation dimensions like "Provenance", "Context" and "Accessibility" by using the statistics generated in the previous step.
 - Compute overall preservation risk scores. The preservation risk scores are calculated by aggregating the scores for all preservation dimensions. The risk scores are normalized within the range 0.0 to 1.0. The higher is the risk score value the higher is the preservation risk. For easier interpretation of the risk analysis results we quantify these results into the preservation risk levels a: "Low" (green color), "Middle" (yellow color) and "High" (represented by red color). The thresholds for these levels are defined as follows: 0.5 between Low and Middle and 0.8 between Middle and High. The total risk score value and total risk level are calculated over all analysed dimensions.
 - Generate preservation risk report in HTML format. This report comprises



preservation risk scores and risk levels calculated for associated metadata statistics like “BrokenObjects” and preservation dimensions like “Accessibility”. It also contains Europeana collection analysis report with the collection id, the collection name, the count of broken links and missing fields, total broken objects and total analysed objects.

2. Evaluating software solutions available for enacting preservation plans. The main goal of this scenario is to retrieve rich information on file formats from LOD repositories. In this way a user is able to receive textual descriptions and meaningful information about software supporting the given file format and vendors that created this software. This kind of information is likely to support the creation of preservation plans as a result of the binary analysis performed with the help of the Normalization service. The basis of this scenario relies on rich data descriptions retrieved from Linked Open Data (LOD) repositories like DBPedia, Freebase, PRONOM etc. The LOD data is automatically harvested using the Web access points provided by these repositories and the supported query languages (e.g. like SPARQL or MQL). The collected information is processed, normalized and integrated into the service’s knowledge base. The Rest API of this service supports querying for textual descriptions of the file formats, software and vendors descriptions. The actions to be taken for completing this scenario are the summarized in the followings:

- Check availability of the file format descriptions in the service database and retrieve data from LOD repositories if necessary.
- Generate rich format descriptions. Aggregated reports on FileFormats, Software and Software Descriptions are generated as HTML tables or CSV files containing information like “FileFormatDescription”, “SoftwareName”, “RepositoryName”, “SoftwareHomepage”, “SoftwareDescription” etc. These reports also include:
 - References to LOD repository descriptions (PRONOM/DBPEDIA/FREEBASE). According to the LOD principles, each repository has its own mechanism for nonambiguous referentiation of the managed concepts. By having a reference in a correct format, a user is able to easily address the information from a web service. As example, the references for “pdf” file extension look like following:

DBPedia – Portable_Document_Format²

Freebase – /en/portable_document_format³

- **PRONOM** – fmt/14⁴ Software and Vendors supporting the given format. Here evaluated software and vendor objects are returned in HTML format.

² http://dbpedia.org/resource/Portable_Document_Format

³ http://www.freebase.com/view/en/portable_document_format

⁴ <http://www.nationalarchives.gov.uk/PRONOM/Format/proFormatSearch.aspx?status=detailReport&id=613&strPageToDisplay=summary>



2.3 Technical Documentation for Risk Management

2.3.1 UML diagrams

This service has been identified and preliminary described in [D2.0.4] as a provider of common functionality for the Preservation Riskmanagement Service (outcome of the Task 2.3.1).

The risk management service API consists of domain objects (Figure 3 and Figure 4) and server and client side APIs (Figure 5 and Figure 6).

Risk management domain objects are logically separated in two groups associated with correspondent user evaluation scenario (see class diagrams in Figure 3 and Figure 4): *risk management domain objects for scenario 1* and *LOD domain objects for scenario 2*.

The CollectionAnalysisReport and MetadataAnalysisResult collections contain results of metadata analysis computation, whereas collection analysis report is an overview of metadata analysis result objects.

Based on these results, further metadata statistics calculation is possible. These are:

- Metadata analysis statistics (MetadataAnalysisStatistics, MetadataAnalysisStatistic, MetadataAnalysisValue). Metadata analysis statistics describe a list of metadata analysis statistic objects. Each of these objects comprises Europeana collection name and ID, report name and metadata analysis value list. Metadata analysis value object is a main object of metadata analysis and describes different statistical fields like "Value", "ValueCount", "Percent", "DistinctValueCount", "AllValuesCount", "FillingLevel", "EmptyFieldsCount", "TotalEmptyFieldsCount", "TotalObjectFieldsCount", "EuropeanaUri", "Link" and "ErrorCode". The meaning of these fields is self explanatory.
- Preservation dimensions statistics (PreservationDimensions, PreservationDimension). Preservation dimensions are calculated based on metadata analysis statistics. These are "Provenance", "Context" and "Accessibility". The "Provenance" comprises such Europeana object fields as DcCreator, DcPublisher, DcContributor, DcCoverage, DcTermsProvenance, EuropeanaCountry, EuropeanaDataProvider, EuropeanaProvider, EuropeanaYear, DcTermsSpatial, DcTermsTemporal. The "Context" stands for such Europeana fields as DcDate, DcRelation, DcTermsIsPartOf, DcTermsCreated, DcTermsIssued, DcTermsIsVersionOf, DcTermsIsReplacedBy, DcTermsReplaces, DcTermsRequires, DcTermsHasPart, DcTermsIsReferencedBy, DcTermsReferences, DcTermsIsFormatOf, DcTermsHasFormat, DcTermsConformsTo, DcTermsHasVersion, DcTermsIsRequiredBy. The "Accessibility" contains DcType, DcFormat, DcLanguage, DcRights, EuropeanaObject, EuropeanaShownBy and EuropeanaShownAt.
- Risk score report (OverallRiskScoreReport, RiskScoreReport). The risk score report conducts preservation dimensions evaluation and provides an HTML formatted response including a table of different preservation dimension fields with associated field filling level in percentage. This is a basis for the overall risk score report that comprises preservation risk scores and risk levels calculated for associated metadata statistics and preservation dimensions. It also contains Europeana collection analysis report with collection id, collection name, the count of broken links and missing fields, total broken objects and total analysed objects.

The Figure 3 presents ASSETS domain objects used for the scenario 1 implementation. In this scenario metadata analysis values are filled out by statistical analysis of the selected



Europeana collection. The computation is based on the collection analysis report stored in database that comprises metadata analysis result.

A collection analysis report provides functionality for the generation of different statistical reports. To create a risk score report a list of preservation dimension objects can be evaluated based on data stored in database at metadata collection level. This enables the generation of the overall risk score report with risk scores and risk levels estimations.

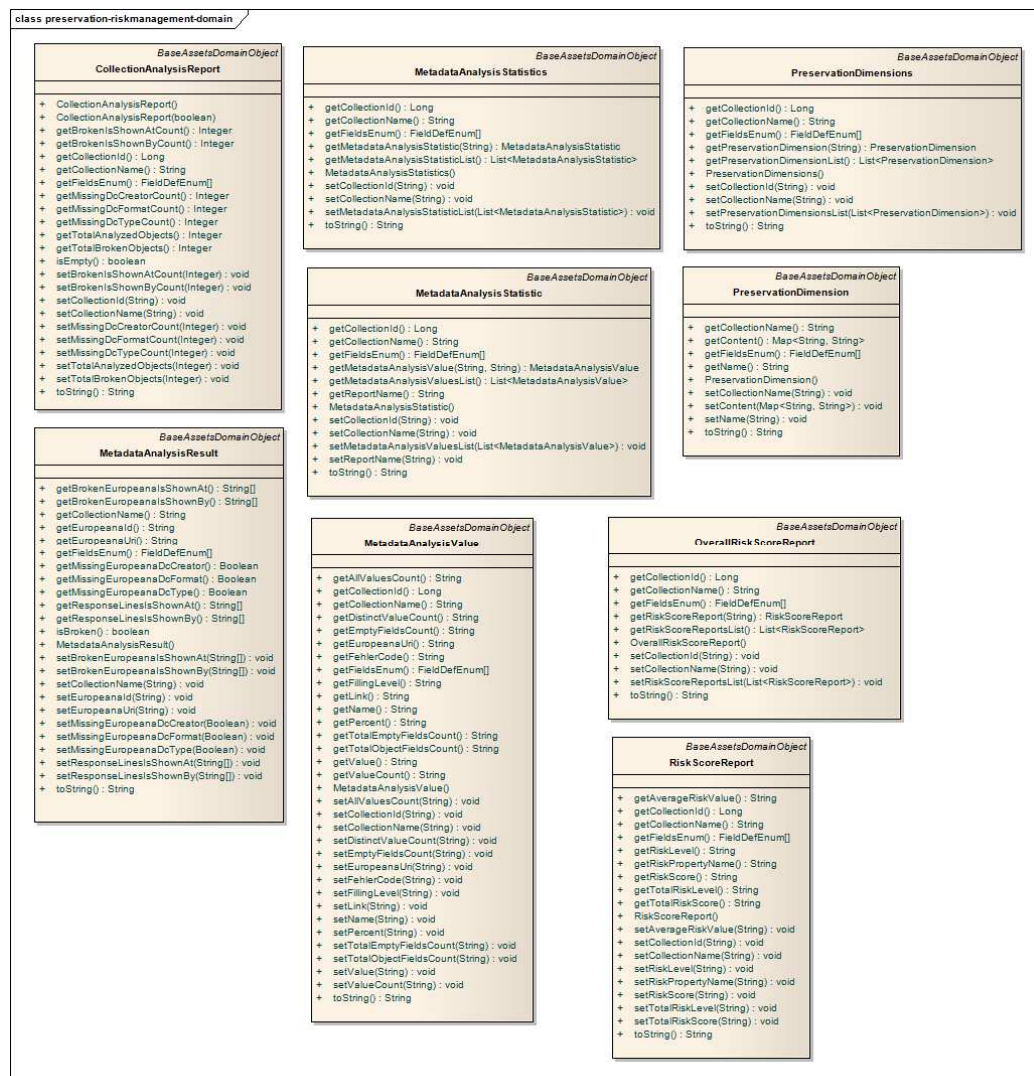


Figure 3 - Risk Management Domain Objects (Scenario 1)

The linked open data (LOD) domain objects (see Figure 4) support storage, retrieval and analysis of information retrieved from LOD repositories.

This structured information is a knowledge base to be used for deriving preservation recommendations. In Figure 4 ASSETS domain objects are depicted; these are an internal representation of the external data (see PronomFileFormat, FreebaseFileFormat and

DBPediaFileFormat classes). collections aggregate file formats from associated repositories.

The LODFormat, LODSoftware and LODVendor objects store data retrieved from LOD repositories using MQL and SPARQL queries and applying file formats from the mentioned file format collections.

The DipFormatId, DipSoftwareId and DipVendorId objects comprise internal mapping between evaluated formats, software and vendors. These objects support merging of similar information from different data sources.



Figure 4 - Linked open data (LOD) domain objects (Scenario 2)

The presented domain object model supports the functionality for LOD analysis related information. The input information is a file format extension. It is the starting point for further steps. The retrieval of associated LOD repository references, software and vendors is supported.

The access to the riskmanagement services from the client applications (see Figure 5) is provided by the PreservationRiskmanagement (scenario 1) and the PreservationRiskmanagementLodDataAnalysis (scenario 2) interfaces.

Within the first business scenario we use the performMetadataAnalysis() method to create a collection analysis report. Then we apply computeMetadataAnalysisStatistics() method for metadata analysis statistics evaluation, computeRiskScore() method for preservation dimensions evaluation and computeOverallRiskScore() method for risk score report generation.

For the second scenario we use the storeAllExtensions() and checkLodData() methods to populate and verify the existence of LOD information within the service database. Then we apply retrieveSoftware() or retrieveVendors() methods to retrieve necessary data according to passed file format extension.

The retrievePreservationStatistics() method presents preservation statistics for a particular type and file format extension.

- Type 1 means all existing statistics. It comprises lists of LODFormats, LODSoftware and LODVendors in CSV format.
- Type 2 includes references to LOD repository descriptions (PRONOM/DBPEDIA/FREEBASE).
- Type 3 comprises lists of textual format descriptions.
- Type 4 comprises lists of software and vendors supporting the given format.

The PreservationRiskmanagementMetadataAnalysis interface can be used for metadata analysis reports creation in CSV format.

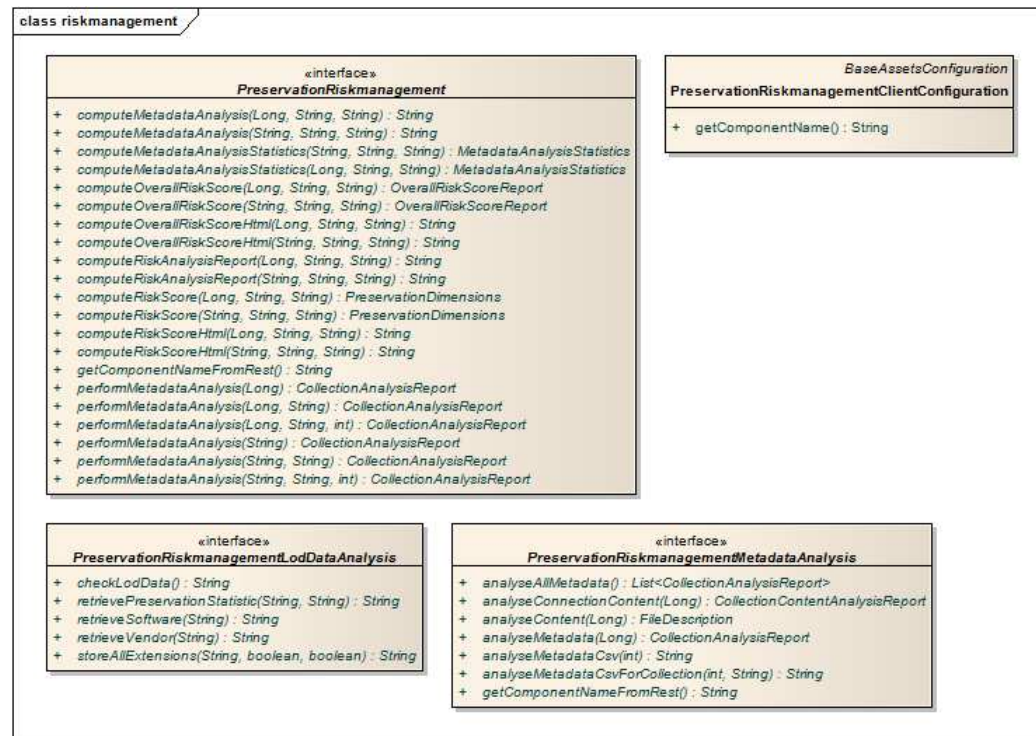


Figure 5 - Risk Management interfaces: client side

On the server side, services support the PreservationRiskmanagementService interface (see Figure 6). Here we see correspondent methods for client methods described above.



Figure 6 - Risk Management: server side

2.3.2 REST services

The risk management service methods are remotely accessible through the associated REST interface using the **restURL** (<http://<server.url>/assets/preservation-riskmanagement/rest>) as a root service link, where <server.url> can either be:

- ☐ <http://assetstest.atc.gr> (test server)
- ☐ <http://assetsdemo.atc.gr> ("production" server)

Figure 7 shows an admin.html page displaying a table with the available administration risk management services required for scenario 2 to initialize Mongo database with rich data from LOD repositories and to examine created database for required collections.

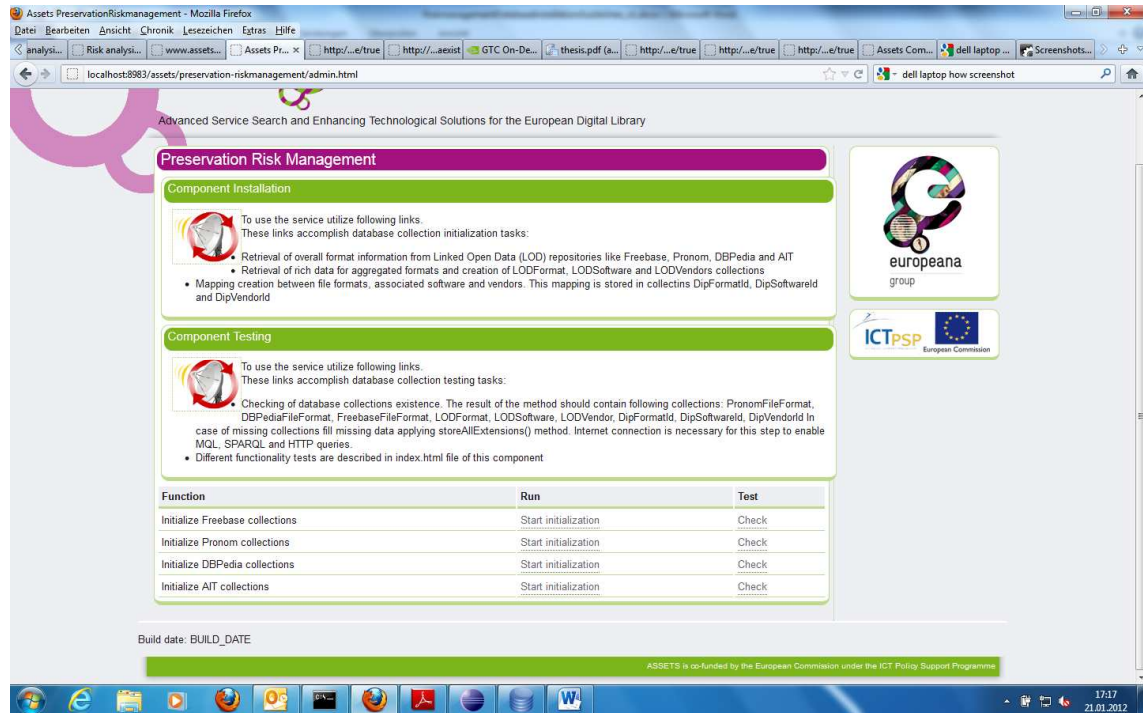


Figure 7 - Available risk management administration services required for scenario 2 to initialize and check database

The specifications for the above displayed services are listed in the following table:

Method	Response type	Name	Input Parameters	Function
GET	TEXT	/storeallexensions/Freebase/	Overwrite repository formats, Overwrite LOD data.	Initialize Freebase collections.
GET	TEXT	/storeallexensions/DBPedia/	Overwrite repository formats, Overwrite LOD data.	Initialize DBPedia collections.
GET	TEXT	/storeallexensions/Pronom/	Overwrite repository formats, Overwrite LOD data.	Initialize Pronom collections.
GET	TEXT	/storeallexensions/AIT/	Overwrite repository formats, Overwrite LOD data.	Initialize AIT collections.
GET	TEXT	/checkdataexist/		Checking of database collections existence.

The result of the check method should contain following collections: PronomFileFormat,

DBPediaFileFormat, FreebaseFileFormat, LODFormat, LODSoftware, LODVendor, DipFormatId, DipSoftwareId, DipVendorId.

Figure 8 shows an index.html page with a table with the available risk management user services related to managing user evaluation scenarios.

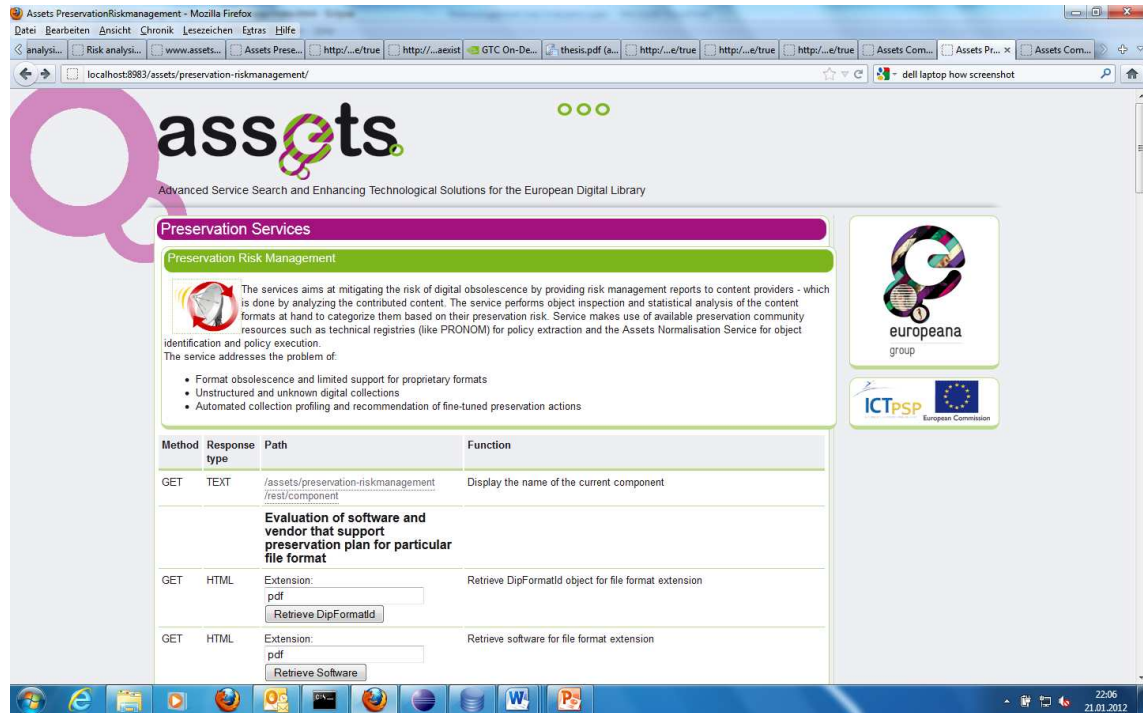


Figure 8 - Available risk management user services

The following table lists user services specifications that provide the Europeana collection preservation risks evaluation (See also the UML Diagrams of the domain model):

Method	Response type	Name	Input Parameters	Function
GET	HTML	/metadataanalysis/html/perform	@collectionId , ID of Europeana collection @config , Configuration (collection names separated by comma) @fileName , Classifications property file name	Perform metadata analysis for Europeana collection.
GET	HTML	/metadataanalysis/html/metadata-statisticsreport	@collectionId , ID of Europeana collection @config ,	Generate metadata statistics report for Europeana

			Configuration (collection names separated by comma) @fileName, Classifications property file name	collection.
GET	HTML	/metadataanalysis/html/riskscore-report	@collectionId , ID of Europeana collection @config , Configuration (collection names separated by comma) @fileName , Classifications property file name	Generate preservation dimension (e.g. Provenance, Context, Accessibility) risk score report for Europeana collection.
GET	HTML	/metadataanalysis/html/overallrisk-scorereport	@collectionId , ID of Europeana collection @config , Configuration (collection names separated by comma) @fileName , Classifications property file name	Generate overall risk score report for Europeana collection that comprises risk levels.

The following table lists user services specifications that provide the evaluation of software and vendor that support preservation plan for particular file format:

Method	Response type	Name	Input Parameters	Function
GET	HTML	/loddataanalysis/dipformatid/html/	File format extension	Retrieve DipFormatId object for file format extension.
GET	HTML	/loddataanalysis/software/html/	File format extension	Retrieve software for file format extension.
GET	HTML	/loddataanalysis/vendor/html/	File format extension	Retrieve vendors for file format

				extension.
--	--	--	--	------------

2.3.3 Risk Management: Client APIs

The main APIs (see Figure 5) for the Preservation Risk Management service are the following:

1. PreservationRiskmanagement API

API	PreservationRiskmanagement
Responsibility	<i>This interface deals with metadata analysis.</i>
Provided methods	<p>public CollectionAnalysisReport performMetadataAnalysis (int collectionId, String config, int objectsCount);</p> <p><i>analyzes Europeana objects metadata for particular Europeana collection ID identified by passed id limited by passing objects count</i></p> <p>@param id the Europeana collection Id.</p> <p>@param config the configuration comprises string values standing for analysis types.</p> <p>@param count the number of test objects to be taken in account. Used for test purposes</p> <p>@return the collection analysis report</p> <p>public MetadataAnalysisStatistics computeMetadataAnalysisStatistics (int id, String config, String classification, RiskReportTypesEnum reportType);</p> <p><i>computes metadata analysis statistics for particular collection for customized configuration. Europeana collection ID used as collection identifier.</i></p> <p>@param id the Europeana collection Id.</p> <p>@param config the analysis configuration containing the names of the preservation dimensions.</p> <p>@param configuration the path to the XML file that comprises risk analysis configuration (e.g. weight and risk score thresholds).</p> <p>@param reportType the type of the risk report.</p> <p>@return metadata analysis statistic object</p> <p>public PreservationDimensions computeRiskReportHtml (int id, String config, String classification, RiskReportTypesEnum reportType);</p> <p><i>computes risk score HTML report for particular collection applying the configuration file for the required preservation dimensions @param id</i></p>

	<p>the Europeana collection Id.</p> <p>@param dimensionConfig the configuration containing the names of the preservation dimensions.</p> <p>@param config the path to the XML file that comprises risk analysis configurations like weight and risk score thresholds.</p> <p>@param reportType the type of the risk report.</p> <p>@return the risk score report</p> <p>public OverallRiskScoreReport computeOverallRiskScore (int id, String config, String classification, RiskReportTypesEnum reportType);</p> <p><i>computes overall risk score report object for particular collection applying analysis configuration</i></p> <p>@param id the Europeana collection Id.</p> <p>@param dimensionConfig the configuration containing the names of the preservation dimensions.</p> <p>@param config the path to the XML file that comprises risk analysis configurations like weight and risk score thresholds.</p> <p>@param reportType the type of the risk report.</p> <p>@return the risk score report</p>
	ASSETS Common

2. PreservationRiskmanagementLodDataAnalysis API

API	PreservationRiskmanagementLodDataAnalysis
Responsibility	<i>This interface deals with the LOD data retrieval and analysis.</i>
Provided methods	<p>public String checkLodData();</p> <p><i>informs client about existence of LOD data in service database. If data is not existing. It checks if following collections exist in database and contain data:</i></p> <ol style="list-style-type: none"> <i>1. file extension collections PronomFileFormat, DBPediaFileFormat and FreebaseFileFormat;</i> <i>2. summarized over all LOD repositories LOD formats, software and vendors collections (LODFormat, LODSoftware and LODVendor);</i> <i>3. file formats mapping collection DipFormatId (contains unique generated DIP identifier and maps file formats identifiers and descriptions from all LOD repositories; contains mapping to DipSoftwareId collection);</i> <i>4. software mapping collection DipSoftwareId (contains unique generated DIP identifier and maps software identifiers and description from all LOD repositories; contains mapping to DipFormatId collection);</i> <i>5. vendor mapping collection DipVendorId (contains unique generated DIP</i>

identifier and maps vendor identifiers and description from all LOD repositories; contains mapping to DipFormatId collection).

@return information about data existence - a list of existing collections.

public String storeAllExtensions (String type, boolean overwriteRepositoryFormats, boolean overwriteLodData);

retrieves LOD data from LOD repositories and stores them into the service database.

@param type the type of storing. Retrieve data from all LOD repositories if type is 'All' or use repository name.

@param overwriteRepositoryFormats overwrite repository formats collections FreebaseFileFormat, DBPediaFileFormat and PronomFileFormat if true.

@param overwriteLodData overwrite summarized LOD data collections LODFormat, LODSoftware and LODVendor if true.

@return the list of updated collections.

public String retrieveSoftware(String ext);

retrieves softwares that supports or uses a particular file format extension.

@param ext the file format extension.

@return the software object.

public String retrieveVendor(String ext);

retrieves associated vendors for particular file format extension.

@param ext the file format extension.

@return the vendor object.

public String retrievePreservationStatistic(String type, String ext);

retrieves preservation statistics for particular type and file format extension.

Type 1: all statistics(All).

Type 2: references to LOD repository descriptions (PRONOM/DBPEDIA/FREEBASE); (ReferencesToLodRepositories)

Type 3: textual format descriptions; (TextualFormatDescriptions)

Type 4: software and Vendors supporting the given format. (SoftwareAndVendorsForFormat).

@param type the type of the preservation statistic report.

	@param ext	the file format extension.
	@return	the vendor object.
Dependencies	ASSETS Common	

The above displayed APIs can be remotely invoked by a client application through their own implementative classes (PreservationRiskmanagementImpl and PreservationRiskmanagementLodDataAnalysisImpl) by using the restURL (**Errore. Riferimento a collegamento ipertestuale non valido.** as main argument during instantiation:

```

PreservationRiskmanagement pm = new PreservationRiskmanagementImpl();
.....

PreservationRiskmanagementLodDataAnalysis pmla =
new PreservationRiskmanagementLodDataAnalysisImpl();
.....

```

2.3.4 Software packaging

For the Risk Management service, there are two modules on the ASSETS continuous integration environment ([HUDSON](#)) which ordinally represent client-side classes and server-side services.

The artifacts for the Preservation Riskmanagement are available on the Europeana SVN (<http://europeanalabs.eu/svn/assets/builds/>).

Two modules for the Risk Management can be found on the continuous integration system.

- "preservation-riskmanagement-client", including artifacts for client-side classes to be used to access server-side services:
 - Preservation Riskmanagement Client artifacts: preservation-riskmanagement-client-0.0.1-SNAPSHOT.jar
- "preservation-riskmanagement", including artifacts for server-side classes implementing the services core:
 - Preservation Riskmanagement Service supporting classes artifacts: preservation-riskmanagement-0.0.1-SNAPSHOT.war

2.3.5 Installation and configuration

The Preservation Riskmanagement service is based on two data sources:

- a MongoDB instance - to store ASSETS domain objects, LOD and statistics information;
- a PostgreSQL Database instance - which is used to store and retrieve Europeana collection objects for further statistics computation.



Please read ASSETS Workspace Setup.pdf from the ASSETS project for detailed installation guidelines (http://europeanalabs.eu/svn/assets/trunk/z_project_setup/documentation/).

Riskmanagement database installation guidelines:

1. Install MongoDB
2. Start MongoDB (e.g. C:\java\mongodb-win32-i386-1.6.5\bin>mongod --rest --port 8060 --bind_ip localhost)
3. Start SOLR - AssetsSolrBackendStarter if metadata statistical analysis services are required
4. Start riskmanagement service - AssetsPreservationRiskmanagementBackendStarter
5. Load data using URI prefix <http://localhost:8983/assets/preservation-riskmanagement/rest/loddataanalysis/> into riskmanagement database if LOD data analysis is required:

a. URI=storeallexensions/AIT/true/true (As a response you will get created collections names like: AitFileFormat; LODFormat; LODSoftware; DipFormatId; DipSoftwareId; LODVendor; DipVendorId;)

b. URI=storeallexensions/Pronom/true/true

c. URI=storeallexensions/Freebase/true/true

d. URI=storeallexensions/DBPedia/true/true

The storeallexensions service comprises three parameter:

- type - The type of storing. Retrieve data from all LOD repositories if type is 'All' or use repository name
- overwriteRepositoryFormats - Overwrite repository formats collections FreebaseFileFormat, DBPediaFileFormat and PronomFileFormat if true
- overwriteLodData - Overwrite summarized LOD data collections LODFormat, LODSoftware and LODVendor if true

The last two parameters are flags that could be set to false in order to reduce overhead if you need to refresh particular part of the database later.

6. Check resulting collections existence using service URI=checkdataexist.

As a response you should get a list of stored collections like (PronomFileFormat; DBPediaFileFormat; FreebaseFileFormat; LODFormat; LODSoftware; LODVendor; DipFormatId; DipSoftwareId; DipVendorId).

2.4 User/Developer Manual for Risk Management

2.4.1 USER MANUAL (how to use risk management service home pages index.html and admin.html)

The available risk management service methods are displayed in the risk management service home pages:

- <http://<server.url>/assets/preservation-riskmanagement/index.html>



- `http://<server.url>/assets/preservation-riskmanagement/admin.html`

where `<server.url>` can either be:

- ☐ `http://assetstest.atc.gr` (test server)
- ☐ `http://assetsdemo.atc.gr` (“production” server)

Please refer to Figure 7 and Figure 8 for a series of screenshots of the above mentioned service home page.

For details about all the operations that can be performed on this page (representing a way of using the risk management service REST URLs), please refer to par. 2.3.2 REST services

2.4.2 DEVELOPER MANUAL

This section is going to show some short JAVA code snippets taken from the risk management test classes that will allow a developer to quickly understand how objects can be instantiated and used by using the service classes.

2.4.2.1 Perform statistical analysis on metadata fields

In order to perform metadata statistical analysis, we have to access methods from a `PreservationRiskmanagement` object. As an input parameter we should pass either `TEST_COLLECTION_ID` or `TEST_COLLECTION_NAME`. The `performMetadataAnalysis` method returns a `CollectionAnalysisReport` object that comprises metadata analysis report.

```
PreservationRiskmanager prm = new PreservationRiskmanagementImpl();
CollectionAnalysisReport collectionAnalysisReport =
    prm.performMetadataAnalysis(TEST_COLLECTION_ID);
```

2.4.2.2 Compute quantification of metadata analysis results over preservation dimensions

In order to compute quantification of metadata analysis results over preservation dimensions, we have to access methods from a `PreservationRiskmanagement` object. Based on retrieved metadata statistics, preservation dimensions like “Provenance”, “Context” and “Accessibility” can be evaluated.

In this sample we also use request configuration “`configList`” and classification property file “`my-assets-preservation-riskmanagement-classification.xml`” to get customized results for a particular `TEST_COLLECTION_ID`.

```
PreservationRiskmanager prm = new PreservationRiskmanagementImpl();
String riskScoreReportString = null;
List<String> configList = new ArrayList<String>();
configList.add("RiskScoreReport");
configList.add("ProvenanceEnum");
riskScoreReportString = preservationRiskmanagement
    .computeRiskScoreHtml(TEST_COLLECTION_ID,
        configList.toString(),
```

```
"my-assets-preservation-riskmanagement-classification.xml");
```

The computeRiskScoreHtml method returns a report in HTML format.

2.4.2.3 Compute preservation risk scores and levels

In order to compute preservation risk scores and risk levels for metadata statistics like “BrokenObjects” and preservation dimensions like “Accessibility”, we have to access methods from a PreservationRiskmanagement object. Whereas minimum risk score value is 0.0 and maximum risk score value is 1.0. The higher is the risk score value the higher is the preservation risk. There are three preservation risk levels in AIT analysis model: “Low” (green color), “Middle” (yellow color) and “High” represented in red color. The total risk score value and total risk level are calculated over all analysed preservation dimensions.

```
PreservationRiskmanager prm = new PreservationRiskmanagementImpl();
List<String> configList = new ArrayList<String>();
configList.add("BrokenObjects");
configList.add("AccessibilityEnum");
riskScoreReportString = preservationRiskmanagement
    .computeOverallRiskScoreHtml(TEST_COLLECTION_ID,
        configList.toString(),
        "my-assets-preservation-riskmanagement-classification.xml");
```

The computeOverallRiskScoreHtml method returns a report in HTML format.

2.4.2.4 Check LOD data availability in service database

In order to check LOD data availability in service database, we have to access methods from a PreservationRiskmanagementLodAnalysis object.

We use the checkLodData method to check whether required database collections (PronomFileFormat, FreebaseFileFormat, DBPediaFileFormat, LODFormat, LODSoftware and LODVendor) exist.

```
PreservationRiskmanagementLodAnalysis pmla = new
PreservationRiskmanagementLodAnalysisImpl();
String report = pmla.checkLodData();
```

2.4.2.5 Store LOD data in database for particular repository

If LOD data is not available and database is empty or not complete we have to create or update the database. In order to store LOD data in database for particular repository (e.g. DBPedia), we have to access methods from a PreservationRiskmanagementLodAnalysis object.

The second parameter “true” means that file format collection for DBPedia will be overwritten in database if they already exist. The third parameter “true” means that LOD data will be overwritten in database if such a data already exist. The text report in response informs about the stored or updated collection names.

```
PreservationRiskmanagementLodAnalysis pmla = new
```



```
PreservationRiskmanagementLodAnalysisImpl();
String report = pmla.storeAllExtensions("DBPedia", true, true);
```

2.4.2.6 Retrieve references to LOD repository

In order to retrieve references to LOD repository (PRONOM/DBPEDIA/FREEBASE), we have to access methods from a PreservationRiskmanagementLodAnalysis object. Each LOD repository has its own reference system. Having a reference in a correct format user is able to easily request this repository from another endpoint. Here are some sample references for

"pdf"	file	extension.	DBPedia	–
<http://dbpedia.org/resource/Portable_Document_Format>;			Freebase	-
/en/portable_document_format;	PRONOM	–	fmt/14.	

Evaluated references are returned in HTML format.

```
PreservationRiskmanagementLodAnalysis pmla = new
PreservationRiskmanagementLodAnalysisImpl();

String report = pmla.retrievePreservationStatistic(
    "ReferencesToLodRepositories", TEST_FILE_EXTENSION);
```

2.4.2.7 Retrieve textual format descriptions

In order to retrieve textual format descriptions, we have to access methods from a PreservationRiskmanagementLodAnalysis object. Each repository has its own textual description. To leave a trace of the original repository, we add repository ID in XML format to the description text e.g. <freebase></en/portable_document_format>Description text</en/portable_document_format></freebase>

Evaluated textual format descriptions are returned in HTML format.

```
PreservationRiskmanagementLodAnalysis pmla = new
PreservationRiskmanagementLodAnalysisImpl();

String report = pmla.retrievePreservationStatistic(
    "TextualFormatDescriptions", TEST_FILE_EXTENSION);
```

2.4.2.8 Retrieve software and vendors supporting the given format

In order to retrieve software and vendors supporting the given format, we have to access methods from a PreservationRiskmanagementLodAnalysis object. The evaluated software and vendor objects are returned in HTML format

```
PreservationRiskmanagementLodAnalysis pmla = new
PreservationRiskmanagementLodAnalysisImpl();

String softwareReport = pmla.retrieveSoftware(TEST_FILE_EXTENSION);
String vendorsReport = pmla.retrieveVendor(TEST_FILE_EXTENSION);
```

3. T2.3.2 Preservation Normalization

3.1 Introduction

The ASSETS Normalization Preservation service offers a wide range of tools (e.g. Droid, JHove, ImageMagick, etc.) and is responsible for deploying and exposing their functionality through a well defined set of standardized preservation operations. These include for example the identification, characterisation, migration and validation of digital objects.

The provided APIs allow for example to migrate a digital object from its original representation into open and preservation-friendly archival formats, such as e.g. PDF/A for documents or TIFF for images - or to profile digital collections. Based on the results of the risk management analysis, the service can automatically perform a normalization strategy on the provider's collection.

3.2 Business scenarios for Normalization

The Preservation Normalization service is part of the ASSETS Digital Preservation services suite and covers the areas of content identification, object characterisation and file format migration as well as (semi) automated QA - all essential aspects for covering the requirements of a digital roundtrip preservation management component.

The following business scenarios have been taken into account when designing the ASSETS Preservation Normalization service:

1. Deployment / Scalability / ASSETS Key Performance Indicators,
2. Definition of atomic-preservation operations and data exchange format,
3. Tools / Proof of Concept,
4. OPF / Planets / UIM interoperability.

Deployment / Scalability

When selecting an adequate technology, framework and deployment scheme the gaps between the following design considerations have to be bridged: on one hand having a modular, robust and failure-tolerant deployment model and technology at hand both backed by an industry quality level open-source implementation and active community; on the other, issuing modular, loosely coupled and failure tolerant components which are capable of batch processing the given volume of ASSETS / Europeana data collections in terms of fulfilling the ASSETS Key Performance Indicators but however at the same time having the degree of flexibility to allow easy exploration and individual experimentation with tools and services at hand via remote hooks.

Definition of Standard Preservation Nouns

This document describes the key API concepts of the ASSETS Preservation Normalization service. It is organized around the two core aspects of the API: services and data. It is essential to provide interface definitions for the core atomic preservation actions (as migration, validation, characterisation, assisted QA, etc.) and an open data exchange



format. This allows homogeneous access to a heterogeneous collection and functionality of tools. Components for actual framework interaction, performance monitoring, execution, service and format lookup, etc. complete the picture. Workflow management is out of scope as this part of system's functionality is being dealt with by the Europeana Unified Ingestion Manager (UIM) and is integrated by using a plugin development mechanism. Reference implementations for all key components have been established and tested.

Tools and available Services

Reference and proof of concept implementations in terms of wrapping tools like Droid, Pdftbox, ImageMagic and others, according to the given technology stack, service wrapping guidelines, exposing them as atomic preservation operations and making them available within the ASSETS Preservation Normalization component for experimentation and/or batch processing.

OPF / Planets / UIM interoperability

The Open Planets Foundation (OPF) has been established to provide practical solutions and expertise in digital preservation, building on the research and development outputs of the Planets project. Planets stands for Preservation and Long-term Access through Networked Services, which was a four-year project co-funded by the European Union under the Sixth Framework Programme to address core digital preservation challenges. OPF is a not-for-profit company, registered in the UK. To find out more about the OPF and how to join, please visit: www.openplanetsfoundation.org.

For ASSETS as well as for OPF it is important to have a solution that is widely adopted and easily being picked up by national heritage organisations and content providers. The OPF believes that establishing digital preservation practice requires an open community that actively shares best practice and is able to apply group learning.

The ASSETS preservation Normalization component therefore aims to be compliant and interoperable with OPF, even having to bridge the gap of different technological requirements, as for example requiring a modular, failure tolerant and loosely coupled deployment model.

3.3 Technical Documentation for Normalization

3.3.1 UML diagrams

This section extends the deliverable [D2.0.4] for the ASSETS Preservation Normalization service.

Format Registry

A central term in the ASSETS Normalization API is the notion of a format, for example to specify the format an object should be migrated to (e.g. migrate file to TIFF 6.0). File formats are represented as URIs and can be specified either as pronom fmt identifier, by file extension or MIME type.

The provided format registry enables creation of format URIs as well as mapping and conversion of different format types. The implementation is based on the droid signature file (for further information see <http://www.nationalarchives.gov.uk/PRONOM/>).



```
FormatRegistry registry = OsgiPreservationUtils.getFormatRegistry();
```

Given the registry, we can create format URIs for PRONOM IDs, MIME types or file extensions, e.g.:

```
URI puid = registry.createPronomUri("fmt/13");
```

The format registry also provides ways to map the different format types (PRONOM, MIME, extension) onto each other, e.g.:

```
Set<String> extensions = registry.getExtensions(puid);
```

The iPojo meta.xml declaration for dependency injection for the OsgiPreservationUtil implementation class is depicted below:

```
<ipojo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="org.apache.felix.ipojo
http://felix.apache.org/ipojo/schemas/CURRENT/core.xsd
      org.apache.felix.ipojo.extender
http://felix.apache.org/ipojo/schemas/CURRENT/extender-pattern.xsd"
      xmlns="org.apache.felix.ipojo">
  <component

      classname="eu.europeana.assets.services.preservation.common.impl.utils.OsgiPres
ervationUtilsImpl"

      name="eu.europeana.assets.services.preservation.common.impl.utils.OsgiPreservat
ionUtilsImpl" immediate="true">

      <!-- provides a service of the interface: PreservationLogHelper -->
      <provides />
      <requires field="logger"/>
      <requires field="serviceregistry"/>
      <!--iPojo dependency injection-->
      <requires field="formatregistry"/>
    </component>

    <instance
component="eu.europeana.assets.services.preservation.common.impl.utils.OsgiPreservatio
nUtilsImpl" />
  </ipojo>
```

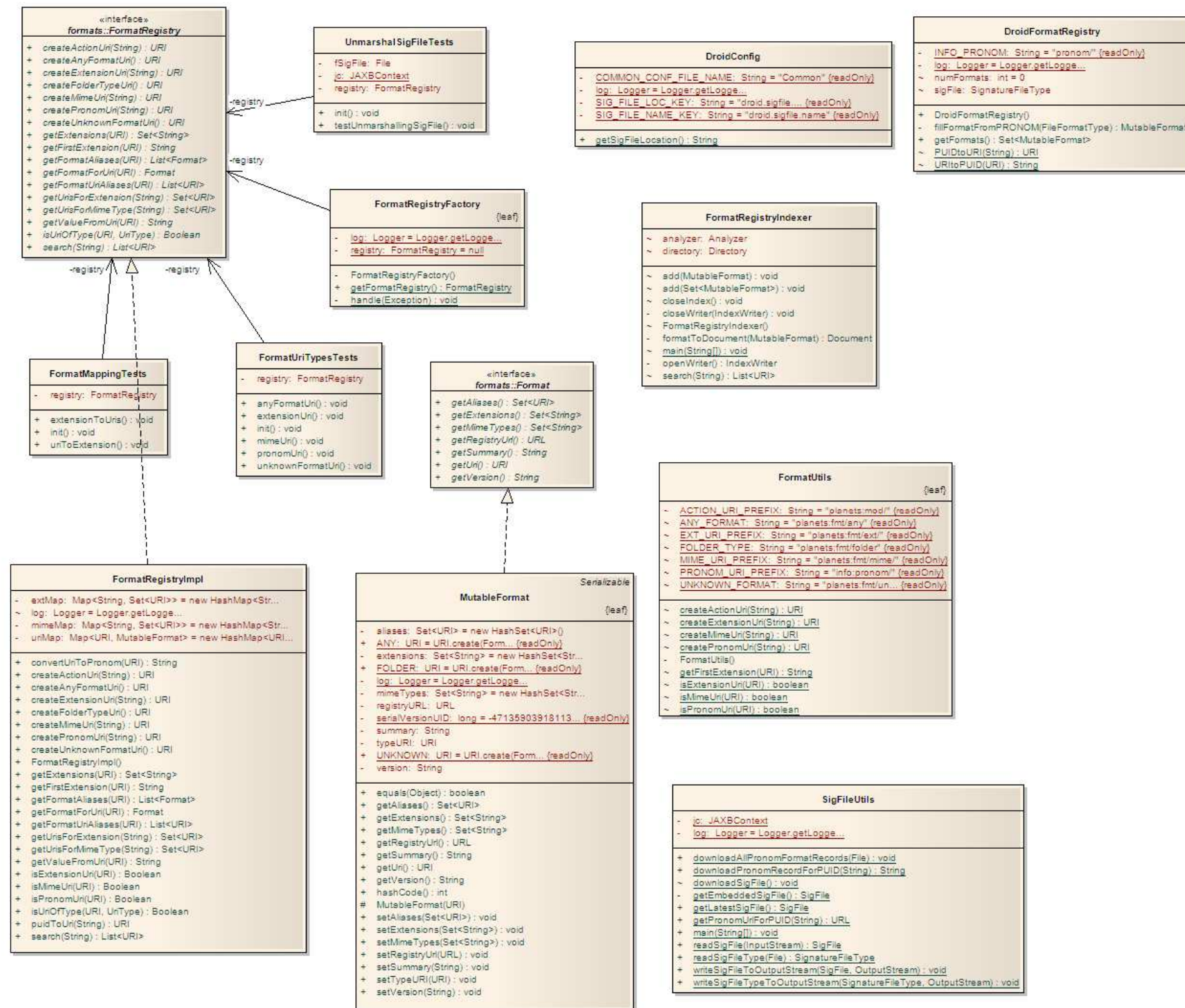



Figura 9 - Format Registry Concepts and Domain Object

Data Exchange Format

Service interoperability is essential to have a simple data exchange format at hand which on the one hand is able to deliver the actual byte stream but at the same time is able to hold simple metadata on the object's metadata (as processing history, events, actors, etc).

To allow streaming of large files even via web services, the object's content can be created in two different ways: by value (the content is embedded in the XML representation of the digital object) or by reference (the content will be streamed)

The central entities are implemented as immutable classes, which are created using the builder design pattern. A minimal digital object consists of nothing but its content. Setting additional attributes therefore can be done on the builder before constructing the actual object. For example:

```
new DigitalObject.Builder(Content.byValue(bytes)).permanentUri(id).build();
```

At the same time, combining an interface with a builder allows the actual implementation class to be hidden behind the API (e.g. to be changed or swapped out after releasing the API).

- Jasper Migration



- OSGi Service Lookup



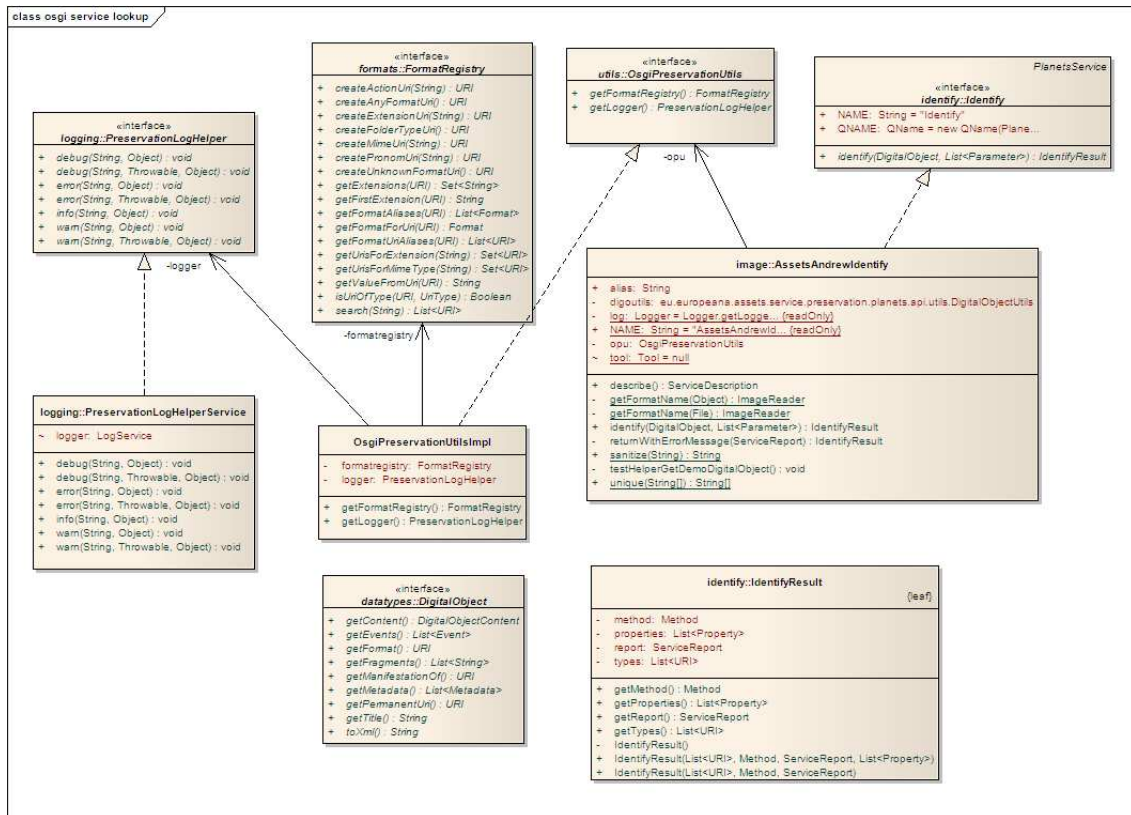


Figure 12 – OSGi Service Utilities

- Service Framework Registration

The service registry and notification handler at hand enables users and other components to look up atomic preservation services that have been registered on the OSGi runtime. A notification mechanism is used to monitor framework events for activated bundles of the supported interfaces. Information provided by the service registry can be used to dynamically select and invoke simple services through the text user interface as well as through its web service operations.

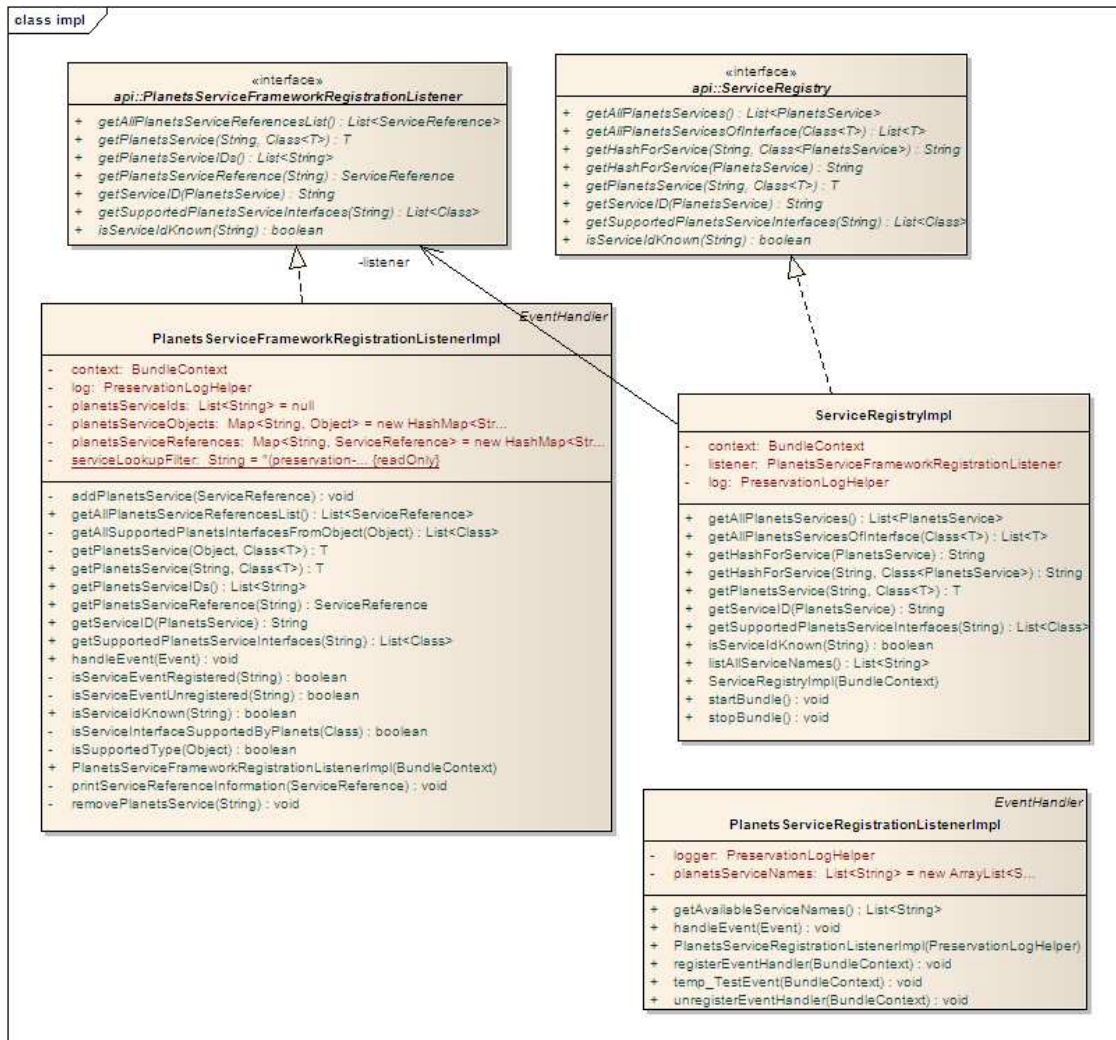


Figure 13 – Service Framework Registration Approach

3.3.2 Web services

The ASSETS Preservation Normalization component takes a Java-first approach. This means that web-services are generated from Java Interfaces that carry web service annotations (see chapter technology decision). Therefore, the ASSETS Normalization components are available both as a platform-independent web services for easy experimentation as well as a plain Java library for digital preservation systems and service development. The main APIs for the Preservation Normalization service are the following:

- preservation operations,
- service registry,
- registration manager,
- format registry.

This is accomplished by APIs for text user system interaction, data access, performance monitoring, logging, and normalisation of measurements.

Preservation Operations

The different kinds of preservation operations are defined as Java Interfaces where every operation is reflected by a corresponding interface definition: Migrate, Validate, Identify, Characterise, Compare and Modify.

Additionally, there are some specialized versions of these Interfaces and some additional, less common interfaces. A service implementation written in Java implements the desired interface and defines itself as a web service using the `@javax.jws.WebService` annotation with an `endpointInterface` attribute.

```
@WebService(name = AssetsAndrewIdentify.NAME,
    serviceName = Identify.NAME,
    targetNamespace = PlanetsServices.NS,
    endpointInterface
    "eu.europeana.assets.service.preservation.common.api.services.identify.Identify"
)
```

This allows the service to reuse all web service specific settings from the Interfaces, without declaring them itself. For Java OSGi clients, DOSGi (Distributed OSGi) enables to use the exposed web service without interfering with SOAP or a WSDL directly, just using the objects as standard service bundle objects. A client is therefore able to work with those objects. A non-Java / non-OSGi client can generate stubs from the WSDL by making use of JAX-WS or by standalone tools as for example SOAP-UI. This way, the ASSETS preservation services can be accessed in a language-independent way.

All operations provide action specific return objects to the given function they implement (e.g. `MigrateResult`, `ValidateResult`, `IdentifyResult`). These are typically composed of the specific result (e.g. a migrated object `Migration`, a format identifier for `Identify`, etc.) as well as a general service report which contains a type (info, error, warn), a status (success, installation error, tool error) and a message. For example:

```
ServiceReport report = new ServiceReport(Type.INFO, Status.SUCCESS, message);
```

Each interface presented above extends the `PlanetsService` interface (see requirement OPF interoperability within the introduction of this document) and therefore must implement a `describe()` method. A service description contains service metadata like supported input formats, the underlying tool name and version, etc. These service descriptions are used to present and explore registered services after they have been picked up by the service registry event notification mechanism.

The main APIs for the atomic preservation operations are the following:

API	Migrate
Responsibility	<i>Migrates a digital object from an input-Format to an output-Format.</i>
Provided methods	public MigrateResult migrate(final DigitalObject digitalObject, URI inputFormat, URI outputFormat, List<Parameter> parameters)

	<p><i>takes the binary content of a digital object container and transforms it according to the given output format specification. Additional tool parameters for fine tuning the tool's settings can be specified</i></p> <p>@param digitalObject The digital object to migrate</p> <p>@param inputFormat The initial format (migrate from)</p> <p>@param outputFormat The required format (migrate to)</p> <p>@param parameters A list of parameters to provide fine grained tool control</p> <p>@return A new digital object, the result of migrating the given object</p>

API	Identify
Responsibility	<i>Identification of a digital object</i>
Provided methods	<p>public IdentifyResult identify(DigitalObject digitalObject, List<Parameter> parameters);</p> <p><i>takes the binary content of the digital object container and runs a format identification on it.</i></p> <p>@param digitalObject The Digital Object to be identified.</p> <p>@return Returns a Types object containing the identification result</p>

API	Compare
Responsibility	<i>Comparison of two digital objects</i>
Provided methods	<p>public CompareResult compare(final DigitalObject first, final DigitalObject second, final List<Parameter> config);</p> <p><i>takes two binaries provided by the digital object container and performs a parameter by parameter similarity check</i></p> <p>@param first The first of the two digital objects to compare</p> <p>@param second The second of the two digital objects to compare</p> <p>@param config A configuration parameter list</p> <p>@return A list of result properties, the result of comparing the given digital object, wrapped in a result object</p> <p>public List<Parameter> convert(final DigitalObject configFile);</p> <p><i>Convert a tool-specific configuration file to a list of service properties that follow the standard URI scheme and therefore can be used within the atomic service API.</i></p> <p>@param configFile The tool-specific configuration file</p> <p>@return A list of parameters containing the configuration values</p>

Service Registry

The service registry listener enables users and service providers to look up information about ASSETS atomic preservation services and to easily interact with them.

In combination with the service registration event handler, which is used to listen for supported services that are registered on the OSGi framework, it is straight forward and simple to request and manage information about available services. The service registry can be accessed from the text user interface shell extension as well as by the SOAP-based web service and distributed OSGi API for getting a handle to execute upon a given set of data objects.

The main APIs for the atomic preservation operations are the following (the Interfaces of the Assets services are made compatible with the OPF technology stack, therefore the intensive usage of the Planets service Interfaces)

API	ServiceRegistry
Responsibility	<i>Provides hooks for querying all registered and known atomic preservation services which have been deployed on the system</i>
Provided methods	<p>public <T extends PlanetsService> List<T> getAllPlanetsServicesOfInterface(Class<T> ofinterface) <i>gets a list of all registered preservation services of a given interface type</i> @param interface The service's interface type @return A typed list of all known service instances of the requested type</p> <p>public List<Class> getSupportedPlanetsServiceInterfaces(String serviceID); @param serviceId A unique identifier for a service on the system @return A list of all supported preservation operations of a given service implementation</p> <p>public <T extends PlanetsService>T getPlanetsService(String serviceId, Class<T> interface) <i>gets a list of all registered preservation services of a given interface type</i> @param serviceId A unique identifier for a service on the system @param interface according interface of the requested service @return Returns a typed PlanetsService handle for the requested serviceId if the service is known to the system.</p> <p>public List<PlanetsService> getAllPlanetsServices() <i>queries the service registry for all available atomic preservation services</i></p>

@return a list of all available Planets Services on the framework

Format Registry

Another central noun in the given API is the notion of a ‘format’ for example to specify the format an object should be migrated to (e.g. migrate a digital object to JPEG2000) or the supposed format of a file to validate against (e.g. validate that a digital object is a PNG). Formats are represented in the code as URIs and can either be specified as PRONOM id (e.g. assets://pronom/fmt/13, see PRONOM), as file extension, or as MIME type.

A registry instance can (same as for the Service Registry) be obtained through dependency injection by iPojo or just be using the given OsgiPreservationUtil utility class which again enables us to hide the actual registry implementation behind the API. The provided implementation is based on the Droid signature file.

```
private OsgiPreservationUtils opu; FormatRegistry registry = opu.getFormatRegistry();
```

The format registry API enables creation of and conversion between these supported format URIs. Given the registry, we can create format URIs for PRONOM IDs, MIME types or file extensions, e.g.:

```
URI puid = registry.createPronomUri("fmt/13");
URI euri = registry.createExtensionUri("PNG");
```

The format registry also provides ways to map the different format types (PRONOM, MIME, extension) to each other, e.g.:

```
Set<String> extensions = registry.getExtensions(puid);
```

The main web-service APIs for the Format Registry component:

API	FormatRegistry
Responsibility	<i>Handling of format URIs e.g. migration of mime-type to PRONOM conversion between different URI namespaces (e.g. assets://pronom/fmt/13, planets://pronom/fmt/13)</i>
Provided methods	<p>public <T extends PlanetsService> List<T> getAllPlanetsServicesOfInterface(Class<T> ofinterface) <i>gets a list of all registered preservation services of a given interface type</i></p> <p>@param interface The service’s interface type @return A typed list of all known service instances of the requested type</p> <p>List<URI> search(String query); @param query The query (as file extension) @return the list of URIs matching query</p> <p>Set<URI> getUrisForExtension(String extension);</p>

@param extension The extension

@return the set of URIs for the passed extension

Set<URI> getUriForMimeType(String mime);

@param mime The mime type

@return the Set of URIs for the passed mimetype

List<URI> getFormatUriAliases(URI typeUri);

This class looks up the different Format URIs consistent with the given URI.

@param typeUri The URI

@return a List of format URIs consistent with the passed URI

List<Format> getFormatAliases(URI typeURI);

@param typeURI The type URI

@return All aliases (in other format types) for the given URI

Format getFormatForUri(URI puri);

@param puri URI schema provided by the FormatRegistry

@return A format instance for the given URI

Set<String> getExtensions(URI uri);

@param uri The URI to find extensions for

@return Extensions corresponding to the given URI

String getFirstExtension(URI uri);

@param uri The URI to find an extension for

@return The first extension found corresponding to the given URI

URI createExtensionUri(String extension);

@param extension The simple file extension

@return A URI representing the extension

URI createActionUri(String action);

@param action The action a modify service can perform (e.g.: "repair", "rotate", "crop" ...)

@return a URI representing this action

URI createPronomUri(String pronom);

@param pronom The pronom ID to create URI for, e.g. "fmt/101"

@return A URI representing the given pronom ID

URI createMimeUri(String mime);

@param mime The mime type to create a URI for

@return A URI representing the given mime type

URI createAnyFormatUri();

@return A URI representing any format.

URI createFolderTypeUri();

@return a URI representing a folder.

URI createUnknownFormatUri();

@return A URI representing an unknown format.

Boolean isUriOfType(URI uri, UriType type);

@param uri The URI to test

@param type The URI type to test for

@return True, if the given URI is of the given type

String getValueFromUri(URI uri);

@param uri The URI

@return The raw value the URI was created with or null

Shell User Interface

Most of the components, like the service registry for example, provide a command line shell extender which allows invoking preservation specific operations through the gogo shell on Karaf (i.e. standard usage of Karaf based OSGi platforms). All components use the scope 'preserv' to provide well documented commands on the shell. The snippet below depicts parts of the service registry's execution interface with arguments for providing the content - either by file, directory or Europeana Semantic Elements (ESE) – as well as a reference on the selected atomic preservation action to execute.

```
@Command(name = "servicereg-execute", scope = "preserv")
public class ServiceRegistryTUIExecute implements Function, Action{
    //these fields are injected by blueprint
```

```

private PreservationLogHelper log;
private ServiceRegistry servreg;

    @Argument(name="content", required=true, description=("the src object as input.
e.g. file reference, directory, or EuropeanID"), index=1)
    private String inputsrc;

    @Argument(name="preservation serviceID", required=true, description=("the
planets preservation service.id to execute"), index=0)
    private String serviceID;
    [...]
    @Option(name="-s",aliases={"--src"},description=("the type of the provided input
src object. e.g. FILE, DIR, ESE"))
    private InputDigitalObjectType inputsrctype = InputDigitalObjectType.FILE;
    /**
     * Blueprint requires to have constructors for handling the arguments
     * @param servicereg
     * @param util
     */
    public ServiceRegistryTUIExecute(ServiceRegistry servicereg, PreservationLogHelper
util){
        this.servreg = servicereg;
        this.log = util;
    }
    /* (non-Javadoc)
     * @see org.apache.felix.gogo.commands.Action#execute(org.osgi.service
.command.CommandSession)
     */
    public Object execute(CommandSession session) throws Exception {
        return execute(session, null);
    }
    /* (non-Javadoc)
     * @see org.osgi.service.command.Function#execute(org.osgi.service.command.
CommandSession, java.util.List)
     */
    public Object execute(CommandSession commandSession, List<Object> arguments)

```

```

        throws Exception {
            log.debug("Service-Registry execute");
            if(this.inputsrctype.equals(this.inputsrctype.FILE)){
                //case1: we're having input type FILE
                File f = new File(this.inputsrctype);
                Identify identify = servreg.getPlanetsService(this.serviceID,
Identify.class);

                Return identify(commandSession,f,identify);
            }
        }

        public void identify(CommandSession commandSession, File f, Identify
identifyService){
            IdentifyExecution exec = new IdentifyExecutionImpl
(f,identifyService,servreg);
            exec.execute();
        }
    
```

The main web-service APIs for the Text User Interface component:

API	IdentifyExecution
Responsibility	<i>Setup and triggers an atomic preservation service for the Identify action.</i>
Provided methods	public void execute() <i>this is responsible for launching a pre-configured workflow on the system. The workflow includes: selecting the Identification service to run, the data itself as well as the post-processing i.e. data normalisation and database persistency. The constructor and methods for configuring an IdentifyExecution are all privately invoked by the given command line interface but not exposed through the web-service API as this part (i.e. workflow) is covered by UIM (unified ingestion manager plugin) integration rather than by this component.</i>

Service Performance

A central key to standardised timing and performance monitoring is the given service performance measurement utility for service developers, which can be used to track performance consistently across the different services. To use it first instantiate the class when the preservation action starts which will trigger the timing on the construction. This is to prevent accidental re-use of each instance of the object, in order to avoid copies of the object being used in a non-thread-safe manner. When the service has finished its work and before creating the final ServiceReport call stop() to halt the timer and use getPerformanceProperties() to attach the measured results to the ServiceReport object. Besides monitoring the wall-clock, it also provides information on CPU time usage of the



current thread (e.g. half the wall-clock time if this process is only getting 50% of the CPU time), compilation time, heap usage, memory allocation, etc.

Overview

The purpose of this document is to provide a high-level introductory overview of the API. For detailed documentation consult the Javadoc API documentation and the unit tests, both included in the ASSETS Preservation Normalization SVN repository. It contains detailed documentation and examples on service implementations of the different interfaces (Migrate, Validate, Identify, etc.) including sample usage and construction of digital objects, the service and format registries, as well as information on additional components as the log-utility, performance measurement, data normalization, directory monitoring or the client utility classes in terms of Javadoc and corresponding JUnit-Tests.

3.3.3 Software packaging

Technology Decisions

Due to the given requirements regarding UIM compatibility / integration as well as other business cases described in the first part of this document, a different choice of technology stack and setup, compared to most other ASSETS 'Spring' services, was chosen. Modularity is achieved by following the OSGi design principles and technology.

OSGi makes it possible to define dependencies of every individual module (so called bundles) with others. An OSGi implementation sits on top of the JVM and provides mechanisms for service management, component definition, execution, management and life cycle control of modules. Services are exposed as interfaces and registered with providing implementations. Through this framework lookup of artefacts and exposition of services together with the event based notification mechanism of life-cycle (start, stop, installed, etc.) events, the system is loosely-coupled compared to regular java structures. This even makes it possible to exchange components on the fly during runtime thus allowing other bundles to react in an expected and failure tolerant manner.

Apache Karaf is built upon the Apache Felix runtime and provides higher level features and services specifically designed for creating OSGi-based servers. This for example includes hot deployment of OSGi bundles, dynamic configuration management, Karaf-features for easy and efficient component deployment. The extensible shell console for example was used to provide an easy to use text interface to expose the ASSETS Preservation Normalization component's functionality. For simplifying the process of OSGi service lookup and binding, we have chosen iPojo dependency injection. Together with Apache Maven 2 for artefact handling, dependency management and the maven plugins 'maven-bundle-plugin' and 'maven-ipojo-plugin' which smoothly integrate into the build process, this has proven as a solid and modular way of packaging and handling bundle artefacts.

Finally to meet the requirements of additionally exposing atomic-preservation services and key components through SOAP web services we have chosen to go with the 'PAX whiteboard extender' bundle. This is an extender bundle that eases the pain of registering servlets, resources, filters and listeners and keeping track of Http Service availability. In our case it picks up javax.jws.WebService annotations and XML configurations to expose the SOAP web services. By having the 'Distributed OSGi Distribution Software Single-Bundle Distribution' available on both the client and server side OSGi runtime profiles these remotely exposed components can be treated as standard OSGi bundles without even

noticing a difference.

Please note, the main drawback of the above described technology decision is although it's simple and straight forward to actually wrap a dependant jar as OSGi bundle, the complexity is hidden in discovering malfunctions of 3rd party libraries. As the scope of all dependant java libraries is handled by the framework all bundles have to adhere to the restricted OSGi classloading principles. Not all libraries (especially the older ones) are able to fully cope with this setup. We had for example to modify and deploy a custom version of JAXB as at this time no OSGi compliant version existed.

For the Preservation Normalization service, there are six modules on the ASSETS continuous integration environment ([HUDSON](#)).

The artifacts for the Preservation Normalization are available on the Europeana SVN (<http://europeanalabs.eu/svn/assets/builds/>).

The above mentioned modules on the continuous integration system are:

- “preservation-service-registry-0.0.1-SNAPSHOT.jar”, packaging type: bundle. osgi bundle for listing and tracking registered preservation services on the system
- “preservation-planets-techreg-0.0.1-SNAPSHOT.jar”, packaging type: bundle. Osgi bundle for handling PRONOM file format information
- “preservation-planets-services-client”, packaging type: pom, including artifacts for client-side standalone distributed OSGi service bindings
 - “assets-all-services-dosgi-client-0.0.1-SNAPSHOT.jar”, packaging type: bundle. A stand alone DOSGi client for consuming the Assets/Planets preservation services functionality
 - “assets-all-services-dosgi-client-binding-0.0.1-SNAPSHOT.jar”, packaging type: bundle. A single Maven artefact to import all required service descriptions and interfaces for a standalone DOSGi client binding
- “preservation-planets-services-parent”, packaging: pom. A root artefact for atomic preservation service implementations
 - “assets-preservation-services-droid-0.0.1-SNAPSHOT.jar”, packaging type: bundle. Identification service wrapper for Droid 6. Exposing DOSGi service deployment via iPojo
 - “assets-preservation-services-jasper19-0.0.1-SNAPSHOT.jar”, packaging type: bundle. Migration service wrapper for the Jasper Transcoder Version 1.900.1 for JPG to JP2 (JPEG2000) and, vice versa, JP2 to JPG conversion.
- “preservation-planets-common-0.0.1-SNAPSHOT.jar”, packaging: bundle. The common infrastructure for the ‘legacy’ planets preservation components
- “preservation-common-0.0.1-SNAPSHOT.jar”, packaging: pom. common infrastructure
 - “preservation-osgi-utils-0.0.1-SNAPSHOT.jar”, packaging: bundle. Provides helpers that via dependency injection pick up OSGi services. A more convenient way of using the preservation services techreg, servicereg, etc.

- “preservation-log-utils-impl-0.0.2-SNAPSHOT.jar” and “preservation-log-utils-api-0.0.1-SNAPSHOT.jar” Preservation Log Utility that exposes constants and convenient interfaces for logging preservation service events.

3.3.4 Installation and configuration

The Preservation Normalization service is based on and tested against Apache Karaf 2.1.0. To install the components on Karaf use Maven to build and to deploy the bundles into the central Maven repository. Alternatively a separate OSGi Bundle Repository (OBR) could be used if you’re not allowed to deploy into the central Maven repository. Make sure you have included the root Karaf-feature descriptor by calling `features:addurl` before running the individual `features:install` command:

```
<features name="preserve-services-jtidy ">
  <feature name="planets-service-jtidy" version="1.0.1.SNAPSHOT">

    <!--features this feature depends upon-->
    <feature version='2.1.0'>http</feature>

    <!-- use the org.ops4j.pax.url wrapper for deploying
jars that aren't available as bundles-->
    <bundle>wrap:mvn:com.sun.xml.ws/jaxws-rt/2.2</bundle>
    <bundle>mvn:commons-io/commons-io/1.4</bundle>

    <!--provided and mavenized by the jtidy's build process-->
    <bundle>wrap:mvn:planets-suite/planets-services-
3rdpartyjars-jtidy/1</bundle>
  </feature>

  <feature name="ipojo" version="1.6.0">

<bundle>mvn:org.apache.felix/org.apache.felix.ipojo/1.6.0</bundle>
  </feature>
</features>
```

For detailed information please refer to the `Setup_and_Installation_Guidelines.pdf` which is part of the source code subversion repository of the project.

3.4 User/Developer Manual for Normalization

This section gives a walkthrough on tool wrapping process and contains a specific example showing how to wrap Droid through its native Java API. Further the required steps that are required for deploying the service on the ASSETS Normalization system are presented.

- **Introduction**

Pronom and Droid are mainly developed at the National Archives (TNA) of the UK and have been a key contribution to the digital preservation community. Pronom is a registry of information about file formats. The TNA provides access to the Pronom registry on-line at <http://www.nationalarchives.gov.uk/PRONOM> as well as by web service endpoints and maintains the information. Droid is a software application that uses some of the file format information to identify the type of specific digital objects through external and internal signature checks. Droid is available on-line through SourceForge at



<http://droid.sourceforge.net/> and is managed as an open source project. In order to identify file formats Droid uses a XML file containing signature information. First, it includes the typical file extensions for the format. For example, PDF files typically end with a 'pdf' extension. Pronom calls these 'external signatures'. Second, it includes patterns that can be used to recognise a file format based on the binary object. For example, a PDF file starts with and ends with Pronom calls these 'internal signatures'. Third, it includes some relationships between formats. For example, PDF is a supertype of PDF 1.1, 1.2, and so on. For more information on the signature files see: [OPF on Pronom and Droid](#)

An alternative tool that makes use of the Droid signature files for object identification is for example Fido, see [Fido on OPF for high performance object identification](#)

- **Selecting Functionality for the Service Wrapper**

As Droid as tool itself is quite flexible in its options it is essential before creating a service wrapper to take the decision what functionality should be exposed. For example Droid identification is performed by checking the file against a set of internal and external signatures. There are several possible outcomes of the identification process as

- Positive, the file is identified and one or more PRONOM identifiers are returned.
- Tentative, the file matches external signatures, one or more Pronom identifiers are returned.
- Negative, the file could not be identified, no identifier is returned.

Droid signatures are contained in an accompanying XML signature file. The tool provides methods for checking whether a current signature file is up to date and downloading a new one from the PRONOM web site. This is higher level functionality and it was decided to simplify by packaging the project with the latest signature file. The current service wrapper implementing comes with Droid 3.0, signature file version 49. When the signature file requires updating the latest version is added to the project and the service is rebuilt and redeployed.

- **Choosing an Interface to Implement**

The `eu.europeana.assets.service.preservation.common.api.services.identify.Identify` interface requires to implement the methods `describe()`, which delivers service and tool information (as licence information, etc.) and `identify(DigitalObject obj)` which takes a digital object and returns a URI indicating the identity of the file format of the object or an unknown type URI if identification failed. Attached we present a entire walkthrough over the required configurations and settings to provide an assets preservation service.

```
/**
 * @author Andrew Lindley (AIT) - andrew.lindley@ait.ac.at
 * @since 11.02.2011
 * Digital Preservation Object Identification Service based on Droid
 *
 * *****
 * Copyright (c) 2010, 2011 The Assets4Europeana Project Partners.
 *
 * All rights reserved. This program and the accompanying
```

```

* materials are made available under the terms of the
* European Union Public Licence (EURL), version 1.1 which
* accompanies this distribution, and is available at
* [http://ec.europa.eu/idabc/eupl.html]
*
*****

* Parts of this work is based on The Planets Project
* Copyright www.openplanetsfoundation.org
* Apache License, Version 2.0
* [http://www.apache.org/licenses/LICENSE-2.0.txt]
*****

*/

@WebService(name = Droid.NAME, serviceName = Identify.NAME, targetNamespace =
PlanetsServices.NS, endpointInterface = "eu.planets_project.services.identify.Identify")
public final class Droid implements Identify, Serializable {
    [...]
    //Note: OSGi abstracts the location of the bundle contents. It is an abuse of the class
loader API to assume all resource URLs are on the file system
    // The configuration for the service
    private static BinarySignatureIdentifier DROID = new BinarySignatureIdentifier();
    static { try{
        //get the Signature File from the bundle's resources which have been included via
bundle-include
        InputStream in = getClass().getResourceAsStream("/resources/Droid_Signature
File.xml");
        File tempSig = copyStreamToTempFile(in);
        DROID.setSignatureFile(tempSig.getAbsolutePath());
        DROID.init();
    } catch(Exception e){}
    }

    private IdentificationMethod method;
    /**
    * {@inheritDoc}
    * @see eu.planets_project.services.identify.Identify

```

```
#identify(eu.planets_project.services.datatypes.DigitalObject, java.util.List)
    */
    public IdentifyResult identify(final DigitalObject digitalObject,
final List<Parameter> parameters) {
        File file = toFile(digitalObject);
        List<URI> types = identifyOneBinary(file);
        ServiceReport report = null;
        if (types == null || types.size() == 0) {
            report = new ServiceReport(Type.ERROR, Status.TOOL_ERROR,
                "No identification result for: " + file);
        } else {
            report = new ServiceReport(Type.INFO, Status.SUCCESS, "");
        }
        IdentifyResult.Method method = null;
        if (IdentificationMethod.BINARY_SIGNATURE.equals(this.method)) {
            method = IdentifyResult.Method.MAGIC;
        } else if (IdentificationMethod.EXTENSION.equals(this.method)) {
            method = IdentifyResult.Method.EXTENSION;
        }
        IdentifyResult result = new IdentifyResult(types, method, report);
        return result;
    }

/**
 * {@inheritDoc}
 * @see eu.planets_project.services.identify.Identify#describe()
 */
    public ServiceDescription describe() {
        ServiceDescription.Builder sd = new ServiceDescription.Builder(
            "DROID Identification Service",
            Identify.class.getCanonicalName());
        sd.version(VERSION);
        sd.classname(this.getClass().getCanonicalName());
        sd.description("Identification service based on Droid (DROID 3.0, Signature
File 16).");
    }
}
```

```

sd.author("Carl Wilson, Fabian Steeg");
sd.tool(Tool.create(null, "DROID", "6.0", null,
    "[http://droid.sourceforge.net/");]
sd.furtherInfo(URI.create("[http://droid.sourceforge.net/");]
// Taking this out as logo is no longer hosted there, and this is bad
// practice anyway - should be hosted locally.
// sd.logo(
//
URI.create("[http://droid.sourceforge.net/wiki/skins/snaphouston/droidlogo.gif");]
sd.serviceProvider("The Planets Consortium.");
return sd.build();
}

/**
 * Identify a file represented as a byte array using Droid.
 *
 * @param tempFile The file to identify using Droid
 * @return Returns the Pronom IDs found for the file as URIs in a Types object
 */
private List<URI> identifyOneBinary(final File tempFile) {
    // Set up the identification request
    RequestMetaData metadata = new RequestMetaData(tempFile.length(),
        tempFile.lastModified(), tempFile.getName());
    RequestIdentifier identifier = new RequestIdentifier(tempFile.toURI());
    identifier.setParentId(1L);
    IdentificationRequest request = new FileSystemIdentificationRequest(
        metadata, identifier);
    try {
        request.open(new FileInputStream(tempFile));
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

```
// Get the results collection
IdentificationResultCollection resultSet = DROID
    .matchBinarySignatures(request);

List<IdentificationResult> results = resultSet.getResults();
List<URI> formatHits = new ArrayList<URI>(results.size());

// Now iterate through the collection and create the format URIs
for (IdentificationResult result : results) {
    formatHits.add(URI.create("info:pronom/" + result.getPuid()));
    this.method = result.getMethod();
}

return formatHits;
}
}
```

All endpoint implementation classes must have a Web Service annotation.

@WebService (javax.jws.WebService)

The name parameter is the name of the wsdl:portType, the serviceName is the Service name of the web service,

@WebResult (javax.jws.WebResult)

This annotation is used to customise the mapping of the method return value to a WSDL part. The name parameter gives the name if the return value in the WSDL, targetNamespace supplies the XML namespace of the return value, while partName specifies the partName for the result.

- **Building and Deployment**

The following artefact which is taken from the Droid identification service's pom.xml file presents the steps which are required to actually build the osgi bundle.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <groupId>eu.europeana.assets</groupId>
        <artifactId>preservation-planets-services-parent</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <relativePath>../pom.xml</relativePath>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <groupId>eu.europeana.assets</groupId>
```



```

<artifactId>assets-preservation-services-droid</artifactId>

<packaging>${packaging.type}</packaging>

<name>Assets Preservation - Normalization - planets services - droid</name>

<description>wrapping droid 6.0 as Assets preservation-service. exposing DOSGI
web-service deployment via iPojo</description>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.5</version>
        </plugin>
    </plugins>
</build>
<profiles>
    <profile>
        <id>bundle</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <properties>
            <packaging.type>bundle</packaging.type>
        </properties>
    </profile>
</profiles>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.felix</groupId>
            <artifactId>maven-bundle-plugin</artifactId>
            <version>2.2.0-SNAPSHOT</version>
            <extensions>true</extensions>
            <configuration>
                <instructions>
                    <Bundle-SymbolicName>${artifactId}</Bundle-SymbolicName>
                    <Bundle-Version>${pom.version}</Bundle-Version>
                    <Export-Package/>
                </instructions>
            </configuration>
        </plugin>
    </plugins>
</build>

```

```

        </instructions>
      </configuration>
    </plugin>
    <!-- osgi - ipojo: service activation meta.xml located in src>main>osgi>ipojo -->
    <plugin>
      <groupId>org.apache.felix</groupId>
      <artifactId>maven-ipojo-plugin</artifactId>
      <version>1.7.0-SNAPSHOT</version>
      <executions>
        <execution>
          <goals>
            <goal>ipojo-bundle</goal>
          </goals>
          <phase>package</phase>
          configuration>
        <metadata>src/main/resources/OSGI-INF/ipojo/meta.xml</metadata>
      </configuration>
    </execution>
  </executions>
</plugin>
</plugins>
</build>
</profile>
</profiles>
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.7</version>
    <type>jar</type>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>eu.europeana.assets</groupId>
    <artifactId>preservation-common-api</artifactId>

```

```

        <version>0.0.1-SNAPSHOT</version>
        <type>bundle</type>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>eu.europeana.assets</groupId>
        <artifactId>preservation-planets-common</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <type>bundle</type>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>uk.gov.nationalarchives</groupId>
        <artifactId>droid-core</artifactId>
        <version>6.0</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>uk.gov.nationalarchives</groupId>
        <artifactId>droid-core-interfaces</artifactId>
        <version>6.0</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>byteseek</groupId>
        <artifactId>byteseek</artifactId>
        <version>1.1</version>
        <scope>runtime</scope>
    </dependency>
</dependencies>
</project>

```

So in this case the different service artefacts have dependencies on the Droid Java API, which however is not directly available via a 'bundle' packaging mechanism. So therefore we have chosen to directly include them as runtime dependencies within the Maven build and handle the deployment of dependent artifacts on Karaf on our own by hand using the `osgi:install` in combination with the 'PAX URL wrap protocol' for converting jars into OSGi

bundles. For example

```
osgi:install -s wrap:mvn:uk.gov.nationalarchives/droid-core-interfaces/6.0
```

An example on how to actually combine the PAX URL wrap protocol within the Karaf feature deployment mechanism is given [here](#)

As described earlier, on the basic atomic preservation services are pure Java classes with some annotations, which get picked up by the system. To make this possible you need to specify the property configuration within the iPojo meta.xml.

```
<instance component=" eu.planets_project.ifr.core.services.identification.droid.impl.Droid
">
  <property name="osgi.remote.interfaces" value="*" />
  <property name="osgi.remote.configuration.type" value="org.apache.cxf.ws" />
  <property
    name="osgi.remote.configuration.pojo.address"
    value="http://localhost:8080/assets-preservation-services-droid" />
</instance>
```

So you can either access its functionality on the Java platform without working in a web service environment or access them remotely. In the latter case JAX-WS can be used to hide the SOAP layer and retrieve a proxy object from a server that will conform to the interface (i.e. you will work with an instance of a class that implements the interface, e.g. Migrate:

```
URL wsdl = new URL("http://127.0.0.1:8080/assets-preservation-services-
droid/Droid?wsdl");
Migrate jtidy = ServiceUtils.createService(Migrate.QNAME, Migrate.class, wsdl);
```

To access the services from non-Java platforms, you can either generate stubs from the WSDL exposed for the service or directly create SOAP messages conforming to the web service schemas (for service descriptions, digital objects, etc).

- **Frequently asked questions**

If you want to directly use distributed OSGi on the client side to integrate the services transparently you only have to provide a service-services.xml within the \resources\OSGI-INF\remote-service

```
<service-descriptions xmlns="http://www.osgi.org/xmlns/sd/v1.0.0">
  <service-description>
    <provide interface="org.apache.felix.ipoj.SamplService" />
    <property name="service.exported.interfaces">*</property>
    <property name="service.exported.configs">org.apache.cxf.ws</property>
    <property
      name="org.apache.cxf.ws.address">http://localhost:9090/SamplServiceInstanceEndpoint<
    /property>
  </service-description>
</service-descriptions>
```

Is it possible to register external / custom 3rd party jars that a given artefact has



dependencies upon into the Maven repository so that it can be referenced by for example Karaf-features.

```
<plugin>

  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-install-plugin</artifactId>
  <version>2.3.1</version>
  <executions>
    <execution>
      <phase>generate-sources</phase>
      <configuration>
        <groupId>preservation-planets-services</groupId>
        <artifactId>jtidy-3rdpartyjar </artifactId>
        <file>${basedir}/lib/Tidy.jar</file>
        <version>1</version>
        <packaging>jar</packaging>
      </configuration>
      <goals>
        <goal>install-file</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

How is it possible to attach arbitrary artifacts to the different stages of the Maven build process? In the provided sample following fragment adds the Karaf feature descriptor using the org.codehaus.mojo build-helper-maven-plugins attach-artefact goal.

```
<plugin>

  <groupId>org.codehaus.mojo</groupId>
  <artifactId>build-helper-maven-plugin</artifactId>
  <version>1.5</version>
  <executions>
    <execution>
      <id>attach-artifacts</id>
      <phase>package</phase>
      <goals>
        <goal>attach-artifact</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

```

        </goals>
        <configuration>
            <artifacts>
                <artifact>
                    <file>target/features/jtidy-osgi-karaf-
features.xml</file>
                    <type>xml</type>
                    <classifier>features</classifier>
                </artifact>
            </artifacts>
        </configuration>
    </execution>
</executions>
</plugin>

```

Within the plugin's configuration section the attached artifacts are retrieved from the target folder of the build. By default Maven copies resources into the target/class folder. In order to keep these files out of the bundle and off the classpath when creating the feature descriptor as part of the bundle module, and additional execution of the maven resources plugin is configured.

```

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
    <version>2.4.3</version>
    <executions>
        <execution>
            <id>copy-features</id>
            <phase>generate-resources</phase>
            <goals>
                <goal>copy-resources</goal>
            </goals>
            <configuration>
                <outputDirectory>target/features</outputDirectory>
                <resources>
                    <resource>
                        <directory>etc/karaf-features</directory>
                        <filtering>true</filtering>

```

```

        </resource>
    </resources>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>

```

- **Eclipse remote debugging with Karaf**

This short tutorial has proven very useful and depicts how to startup Apache Karaf in debug mode and hook up with Eclipse for remotely debugging the application? The easiest way to debug Karaf or any application deployed onto it is to use remote debugging. Remote debugging can be easily activated by setting the KARAF_DEBUG environment variable to true and can be done using the following command on Unix systems:

```
export KARAF_DEBUG=true
```

On Windows, use the following command (on the cmd shell)

```
set KARAF_DEBUG=true
```

Then, you can launch Karaf using the usual way:

```
bin/karaf
```

or

```
bin\karaf.bat
```

Inside the Eclipse IDE connect to the remote application (the default port to connect to is 5005). This is done the following way: • Start Eclipse • Go to Run -> Debug Configurations • Create a new Remote Java Application configuration • Configure the remote application's details

4. T2.3.2 Preservation Notification

4.1 Introduction

Preservation Notification is one of the digital preservation services, which guarantees the adequate communication and management procedures in reaction to events that could impact long term preservation within a digital archive.

As actions, appropriate messages are dispatched according to event types, well defined rules, roles of the entities involved in the digital environment (i.e. curator, preserver, holder).

The alerted entity (i.e. human actor and/or automatic tool) is able to enact corrective actions according to established preservation plans.

4.2 Business scenarios for Notification

The Preservation Notification service is part of the ASSETS Digital Preservation services suite and provides a means to inform user communities that digital content is being managed (amended/added/deleted/archived) and is available/unavailable for use.

Changes and events are represented/reported by terms (TOPICS OF INTEREST) which are classified in a hierarchical structure (e.g. a TAXONOMY).

Data curators (SUBSCRIBERS) express their interest for receiving notification (ALERTS) for specific changes/events based on their own capabilities and skills.

Notification of change/impacting events are filtered by adopting rules and classifications. And this allows generating informational/warning alerts for delivering/exchanging knowledge to interested actors (which can then properly react).

4.3 Technical Documentation for Notification

4.3.1 UML diagrams

The service has been identified and preliminary described in [D2.0.4] as provider of common functionality for the Preservation Notification Service (outcome of the Task 2.3.3) and the Taxonomy-based Notification Service (outcome of the Task 3.2.3).

In particular, the service supports:

1. **Message Management** – management of the notification messages (i.e. creation, publishing, delivering);
2. **Subscription Management** – management of the subscribers and their topics of interest for receiving alerts;
3. **Taxonomy Management** – management of the taxonomies which contain the terms used by the publishers and subscribers for describing events, objects and topics of interest.

In this perspective, the Notification core is represented by three interfaces: i) NotificationManager, ii) RegistrationManager and iii) TaxonomyManager. These interfaces manage the concepts modelled in the pictures below.

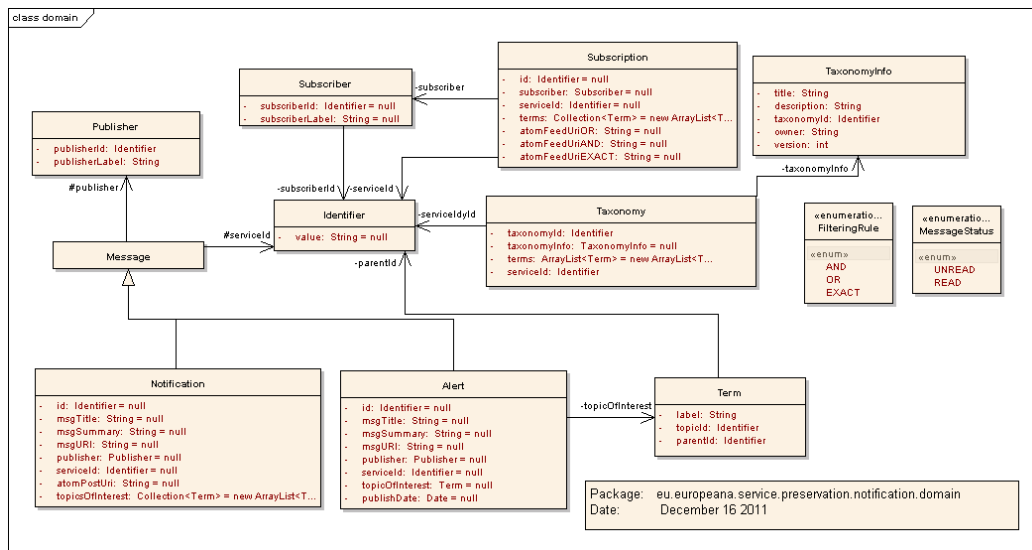


Figure 14 - Preservation Notification: domain models

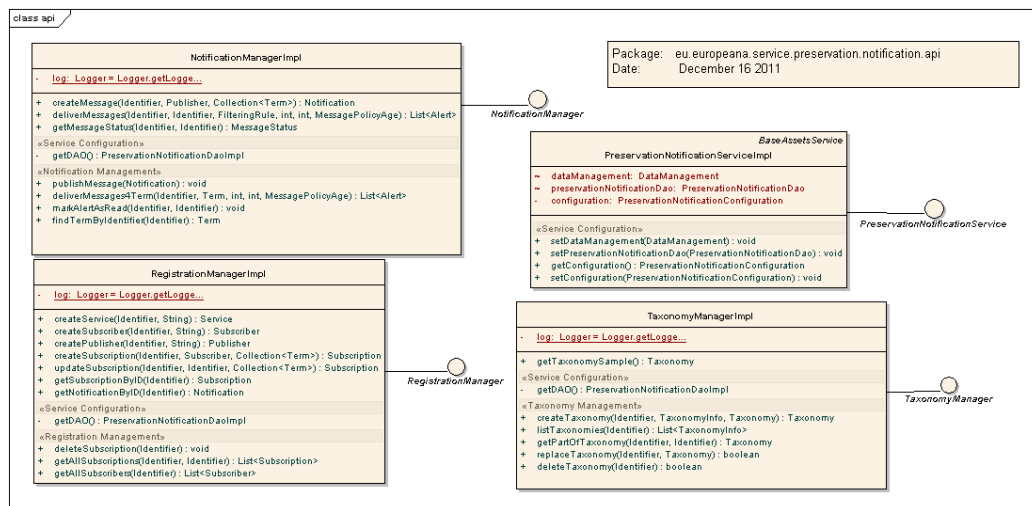


Figure 15 - Preservation Notification: service interfaces

4.3.2 REST services

The available notification service methods are remotely accessible through the associated REST interface using the **restURL** (<http://<server.url>/assets/preservation-notification/rest>) as a root service link, where <server.url> can either be:


- ☐ <http://assetstest.atc.gr> (test server)
- ☐ <http://assetsdemo.atc.gr> ("production" server)



Figure 16 shows a table with the available notification services related to managing external services and taxonomies.

Preservation Services

Preservation Notification



The Notification service guarantees the adequate communication and management procedures in reaction to events that could impact long term preservation within a digital archive.

Its responsibilities are:

- to allow external services to create and load a taxonomy of terms of interest;
- to allow subscribers to subscribe to terms/topics of interest of a pre-loaded taxonomy (CREATING A SUBSCRIBER, CREATING A SUBSCRIPTION FOR TERMS OF INTEREST);
- to allow publishers to submit messages for specific terms/topics of interest (CREATING A PUBLISHER, CREATING A NOTIFICATION FOR TERMS and PUBLISHING A MESSAGE);
- to deliver messages to subscribers (DELIVERING MESSAGES TO SUBSCRIBERS).

Method	Send Request as	Response type	(Path) Example of implementation	Function
GET		TEXT	/assets/preservation-notification/rest/component	Displays the name of the current component.
POST	XML	XML	/assets/preservation-notification/rest/registrationManager/createService	Registers a service under which subscriptions and notification channels may be created.
POST	XML	XML	/assets/preservation-notification/rest/taxonomyManager/createTaxonomy	Creates and load a taxonomy (for a specific service).
GET	QUERY STRING PARAMETER (?serviceId=(value))	XML	/assets/preservation-notification/rest/taxonomyManager/listTaxonomies?serviceId=(value)	Lists existing taxonomies related to a specific service.
GET	QUERY STRING PARAMETERS (?serviceId=(value)&taxonomyId=(value))	XML	/assets/preservation-notification/rest/taxonomyManager/getPartOfTaxonomy?termId=(value)&taxonomyId=(value)	Lists a part (a term and its descendants) of a specific taxonomy.

Figure 16 - Available notification services related to managing external services and taxonomies

The specifications for the above displayed services are listed in the following table:

Method	Response type	Name	Input Parameters	Function
POST	XML	/registrationManager/createService	XML	Registers a service under which subscriptions and notification channels may be created.
POST	XML	/taxonomyManager/createTaxonomy	XML	Creates and loads a taxonomy (for a specific

				service).
GET	XML	/taxonomyManager/listTaxonomies	@serviceld, id of the service for which a list of pre-loaded taxonomies will be retrieved.	Lists existing taxonomies related to a specific service.
GET	XML	/taxonomyManager/getPartOfTaxonomy	@termId, id of a term of a specific taxonomy for which its descendants will be retrieved. @taxonomyId, id of a taxonomy	Lists a part (a term and its descendants) of a specific taxonomy.

Figure 17 shows a table with the available notification services related to managing subscribers and subscriptions.

Creating a Subscriber				
POST	XML	XML	/assets/preservation-notification/rest/registrationManager/createSubscriber	Create a subscriber
Subscriber name: <input type="text"/>				
Creating a Subscription for terms of interest				
POST	XML	XML	/assets/preservation-notification/rest/registrationManager/createSubscription	Creates a Subscription for receiving messages related to chosen terms (e.g. "European" and "Drawings") of a pre-loaded taxonomy (e.g. ASSETS, SKOS)
Term 1: <input type="text"/>				
Term 2: <input type="text"/>				
Viewing a Subscription				
GET		XML	/assets/preservation-notification/rest/registrationManager/getAllSubscribers	Returns an XML file listing all the subscribers
GET	QUERY STRING PARAMETER (?serviceld=(value))	XML	/assets/preservation-notification/rest/registrationManager/getAllSubscribers?serviceld=(value)	Returns an XML file listing all the subscribers related to a specific service
GET	QUERY STRING PARAMETERS (?serviceld=(value)&subscriberId=(value))	XML	/assets/preservation-notification/rest/registrationManager/getAllSubscriptions?serviceld=(value)&subscriberId=(value)	Returns an XML file with the details of all the subscriptions related to a particular service which have been created by a specific subscriber
POST	XML	XML	/assets/preservation-notification/rest/registrationManager/updateSubscription	Amends a specific subscription
POST	XML	XML	/assets/preservation-notification/rest/registrationManager/deleteSubscription	Deletes a specific subscription

Figure 17 - Available notification services related to managing subscribers and subscriptions

The specifications for the above displayed services are listed in the following table:

Method	Response type	Name	Input Parameters	Function
--------	---------------	------	------------------	----------



POST	XML	/registrationManager/createSubscriber	XML	Creates a subscriber
POST	XML	/registrationManager/createSubscription	XML	Creates a Subscription for receiving messages related to chosen terms of a pre-loaded taxonomy (e.g. ASSETS, SKOS)
GET	XML	/registrationManager/getAllSubscribers		Returns an XML file listing all the subscribers.
GET	XML	/registrationManager/getAllSubscribers	@serviceId , id of the service for which a list of registered subscribers will be retrieved.	Returns an XML file listing all the subscribers related to a specific service.
GET	XML	/registrationManager/getAllSubscriptions	@serviceId , id of the service for which a list of subscriptions (registered by a specific subscriber) will be retrieved. @subscriberId , id of the subscriber for which a list of his/her own subscriptions will be retrieved.	Returns an XML file with the details of all the subscriptions related to a particular service which have been created by a specific subscriber.
POST	XML	/registrationManager/updateSubscription	XML	Amends a specific subscription.
POST	XML	/registrationManager/deleteSubscription	XML	Deletes a specific subscription.

Figure 18 shows a table with the available notification services related to managing publishers, creating notification channels and publishing messages.

Creating a Publisher				
POST	XML	XML	(/assets/preservation-notification/rest/registrationManager/createPublisher)	Create a publisher
Publisher name: <input type="text"/> Create Publisher				

Creating a Notification for terms				
POST	XML	XML	(/assets/preservation-notification/rest/notificationManager/createMessage)	Creates a Notification channel where to publish messages whose content is related to chosen terms (e.g. "European" and "Drawings") of a pre-loaded taxonomy (e.g. ASSETS, SKOS)
Term 1: <input type="text" value="European"/> Term 2: <input type="text" value="Drawings"/> Create Notification				

Publishing a message				
POST	XML	XML	(/assets/preservation-notification/rest/notificationManager/publishMessage)	Publish a message on a Notification channel
Content URL (for which a notifying message will be sent to subscribers): <input type="text" value="http://assetstest.atc.gr/portal/record/92001/23DADB2389A55A91"/> Message Title: <input type="text" value="Map of Luxembourg"/> Message Summary: <div style="border: 1px solid #ccc; padding: 5px;"> This map of the town and fortress of Luxembourg, the second of its kind, was drawn in situ around 1560 by Jacob Roelofs, aka Jacob van Deventer, after the town of his birth </div> Publish Message				

Figure 18 - Available notification services related to managing publishers, creating notification channels and publishing messages

The specifications for the above displayed services are listed in the following table:

Method	Response type	Name	Input Parameters	Function
POST	-XML	/registrationManager/createPublisher	XML	Creates a publisher.
POST	-XML	/notificationManager/createMessage	XML	Creates a Notification channel where to publish messages whose content is related to chosen terms of a pre-loaded taxonomy (e.g. ASSETS, SKOS).
POST	-XML	/notificationManager/publishMessage	XML	Publishes a message on an existing Notification channel.

Figure 19 shows a table with the available notification services related to managing the

delivery of messages.

Delivering messages (already published by a publisher) to subscribers (alerts)				
POST	XML	XML	(/assets/preservation-notification/rest/notificationManager/deliverMessages) Subscription ID: <input type="text"/> Type of delivery rule: <input checked="" type="radio"/> AND <input type="radio"/> EXACT <input type="radio"/> OR Starting from message n°: <input type="text"/> Max bunch of messages to be delivered: <input type="text"/> Deliver Messages	Delivers all the messages for a given subscription (according to the chosen delivery rule, "AND", "OR", "EXACT")
POST	XML	XML	/assets/preservation-notification/rest/notificationManager/deliverMessages4Term	Delivers all the messages published under a given term/topic of interest (all subscribers who have subscribed to that term/topic of interest will receive alerts).

Figure 19 - Available notification services related to managing the delivery of messages.

The specifications for the above displayed services are listed in the following table:

Method	Response type	Name	Input Parameters	Function
POST	XML	/notificationManager/deliverMessages	XML	Delivers all the messages for a given subscription (according to the chosen delivery rule, "AND", "OR", "EXACT").
POST	XML	/notificationManager/deliverMessages4Term	XML	Delivers all the messages published under a given term/topic of interest (all subscribers who have subscribed to that term/topic of interest will receive alerts).

4.3.3 Preservation Notification : Client APIs

The main APIs for the Preservation Notification service are the following:

1. TAXONOMY MANAGER API

API	TaxonomyManager
Responsibility	<i>This interface deals with registration and management of taxonomies.</i>
Provided methods	public Taxonomy createTaxonomy(Identifier servceld, TaxonomyInfo taxonomyInfo, Taxonomy taxonomy) throws Exception; <i>allows a specific service to create and register a taxonomy by providing the fundamental information: the taxonomy and its information</i> @param servceld the identifier of the service invoking the

	creation method. This service is the owner of the taxonomy.
@param taxonomyInfo	the information of the taxonomy (title, description).
@param taxonomy	the taxonomy as set of terms (with broaders/parents and narrowers/childs).
@return	the taxonomy created and registered with an identifier (taxonomyId)
public List<TaxonomyInfo> listTaxonomies(Identifier serviceId);	
<i>allows to obtain the information of all the registered taxonomies from a specific service</i>	
@param serviceId	the identifier of the service
@return	a list of all the taxonomies registered by a specific service
public Taxonomy getPartOfTaxonomy(Identifier termId, Identifier taxonomyId);	
<i>allows to obtain a part of a taxonomy, starting from a specific term.</i>	
@param termId	the identifier of the term
@param taxonomyId	the identifier of the taxonomy
@return	(part of) a taxonomy
public boolean replaceTaxonomy(Identifier taxonomyId, Taxonomy newTaxonomy);	
<i>allows to replace an existing taxonomy with a new one</i>	
@param taxonomyId	the identifier of the taxonomy being replaced.
@param newTaxonomy	the new taxonomy used for replacing the old one.
@return	true, if the replacement occurred successfully; otherwise false.
public boolean deleteTaxonomy(Identifier taxonomyId);	
<i>allows to remove an existing taxonomy</i>	
@param taxonomyId	the identifier of the taxonomy to be removed.
@return	true if the deletion occurred successfully, otherwise false.

Dependencies	ASSETS Common
--------------	---------------

2. REGISTRATIONMANAGER API

API	RegistrationManager
Responsibility	<i>This interface deals with the registration of subscriptions and subscribers involved in the notification process.</i>
Provided methods	<p>public Service createService(Identifier id, String label) throws Exception; <i>allows to register a Service with specified id and label.</i></p> <p><i>If id == null, it will be generated.</i></p> <p><i>If a Service with provided id already exists, it will be returned.</i></p> <p>@param id the Identifier of the Service to be created, or null if a new one should be generated.</p> <p>@param label the name of the Service to be created.</p> <p>@return the created Service (or the existing one).</p> <p>public Subscriber createSubscriber(Identifier id, String label) throws Exception; <i>allows to register a Subscriber with a specified id and a label (subscriber's name).</i></p> <p><i>If id == null, it will be generated.</i></p> <p><i>If a Subscriber with the provided id already exists, it will be returned</i></p> <p>@param id the Identifier of the Subscriber to be created, or null if a new one should be generated.</p> <p>@param label the label (name) of the Subscriber to be created.</p> <p>@return the created Subscriber (or the existing one).</p> <p>public Publisher createPublisher(Identifier id, String label) throws Exception; <i>allows to register a Publisher with a specified id and a label (publisher's name).</i></p> <p><i>If id == null, it will be generated.</i></p> <p><i>If a Publisher with the provided id already exists, it will be returned</i></p> <p>@param id the Identifier of the Publisher to be created, or null if a new one should be generated.</p> <p>@param label the label (name) of the Publisher to be created.</p> <p>@return the created Publisher (or the existing one).</p>

public Subscription createSubscription(Identifier servid, Subscriber subscriber, Collection<Term> terms) throws Exception ;

allows to register a subscription for a specific subscriber and to specify the set of terms of interest for receiving alerts. The operation returns an identifier for the registered subscription.

@param servid the identifier of the service invoking the creation of a subscription.

@param subscriber the subscriber who is interested to specific terms.

@param terms the terms of interest for which to receive notifications

@return the subscription to the specified terms of interest.

public Subscription updateSubscription(Identifier servid, Identifier subscriptionId, Collection<Term> terms) throws Exception;

allows to update the information of a registered subscription.

@param servid the identifier of the service invoking the update of a subscription.

@param subscriptionId the identifier of the being updated subscription.

@param terms the new terms replacing the old ones in the being updated subscription.

@return the updated subscription.

public void deleteSubscription(Identifier subscriptionId) throws Exception;

allows to remove the registration of a specific subscription.

@param subscriptionId the identifier of the subscription to be removed.

@return true if the deletion is successfully completed, otherwise false.

public List<Subscription> getAllSubscriptions(Identifier servid, Identifier subscriberId);

allows to obtain all the subscriptions registered under a service by a specific subscriber.

@param servid the identifier of the service.

@param subscriberId the identifier of the subscriber.

@return the list of all the subscriptions.

	<p>public List<Subscriber> getAllSubscribers(Identifier serviceId);</p> <p><i>allows to obtain all the subscribers (whose subscription have been registered) under a specific service</i></p> <p>@param serviceId the identifier of the service.</p> <p>@return the list of all subscribers' identifiers.</p>
Dependencies	ASSETS Common

3. NOTIFICATIONMANAGER API

API	NotificationManager
Responsibility	<i>This interface manages the messages and their lifecycle. The message is created and published by a publisher. Finally it is delivered to the interested subscribers.</i>
Provided methods	<p>public Notification createMessage(Identifier serviceId, Publisher publisher, Collection<Term> terms) throws Exception;</p> <p><i>allows to create a notification message for a set of topics of interest representing events and/or objects. It has a MessageHeader, MessageProperties and a MessageBody. A notification message may be addressed for more than one topic.</i></p> <p>@param serviceId the identifier of the service from which the method is invoked.</p> <p>@param publisher who is going to publish a message.</p> <p>@param terms the topics for which the message is going to be published.</p> <p>@return the notification message.</p> <p>public void publishMessage(Notification notification) throws Exception;</p> <p><i>allows a publisher to submit and publish a notification message for a set of topics. This notification is previously created by using the createMessage operation.</i></p> <p>@param notification The message which has to be published.</p> <p>public List<Alert> deliverMessages(Identifier serviceId, Identifier subscriptionId, FilteringRule filterRule, int indexFrom, int maxBunch, MessagePolicyAge policyAge);</p> <p><i>allows a subscriber to receive a bunch of alert messages for a specific subscription of a service, according to the expressed message policy (e.g. if the message has been posted before the subscription) and the filtering rule (e.g. AND, OR);</i></p>

	@param serviceId	the identifier of the service invoking the delivery.
	@param subscriptionId	the identifier of the subscription for a set of terms.
	@param filterRule	filter the messages by applying AND or OR.
	@param indexFrom	index from which the alerts have to be returned.
	@param maxBunch	max size of the returned alerts.
	@param policyAge	specifies the age policy of delivered alerts.
	public List <Alert> deliverMessages4Term(Identifier serviceId, Term term, int indexFrom, int maxBunch, MessagePolicyAge policyAge); <i>allows to receive a bunch of alert messages for a specific term of interest, according to the expressed message policy. The messages refer to the exact matching for the term, and not for its childs.</i>	
	@param serviceId	the identifier of the service invoking the delivery.
	@param term	the term of interest contained in the notification.
	@param indexFrom	index from which the alerts have to be returned.
	@param maxBunch	max size of returned alters.
	@param policyAge	specifies the age policy of delivered alerts.
	public MessageStatus getMessageStatus(Identifier alertId, Identifier subscriberId); <i>allows to obtain the status (i.e. read or unread) of a specific alert message.</i>	
	@param alertId	the alert message identifier.
	@param subscriberId	the subscriber identifier.
	public void markAlertAsRead(Identifier alertId, Identifier subscriberId) throws Exception; <i>allows to set the status of a specific alert message as read.</i>	
	@param alertId	the alert message identifier.
	@param subscriberId	the subscriber identifier.
Dependencies	ASSETS Common	

These above displayed APIs can be remotely invoked by a client application using classes TaxonomyManagerImpl, RegistrationManagerImpl, NotificationManagerImpl) by using the restURL (**Errore. Riferimento a collegamento ipertestuale non valido.** as main argument during instantiation:

```
TaxonomyManagerImpl tm = new TaxonomyManagerImpl(restURL);
.....
NotificationManagerImpl nm = new NotificationManagerImpl(restURL);
.....
.....
RegistrationManagerImpl rm = new RegistrationManagerImpl(restURL);
.....
```

4.3.4 Software packaging

For the Preservation Notification service, there are two modules on the ASSETS continuous integration environment ([HUDSON](http://europeanalabs.eu/svn/assets/builds/)) which ordinally represent client-side classes and server-side services.

The artifacts for the Preservation Notification are available on the Europeana SVN (<http://europeanalabs.eu/svn/assets/builds/>).

The above mentioned modules that can be found on the continuous integration system are.

- "preservation-notification-client", including artifacts for client-side classes to be used to access server-side services:
 - Preservation Notification Client artifacts: preservation-notification-client-0.0.1-SNAPSHOT.jar
 - Preservation Notification Client Javadoc artifacts: preservation-notification-client-0.0.1-SNAPSHOT-javadoc.jar
- "preservation-notification", including artifacts for server-side classes implementing the services core:
 - Preservation Notification Service supporting classes artifacts: preservation-notification-0.0.1-SNAPSHOT-classes.jar
 - Preservation Notification Service Javadoc artifacts: preservation-notification-0.0.1-SNAPSHOT-javadoc.jar
 - Preservation Notification Service artifacts: preservation-notification-0.0.1-SNAPSHOT.war

4.3.5 Installation and configuration

The information contained in this section is publicly available on <http://assetsdev.atc.gr/trac/wiki/preservation-notification-deployment>.



The Preservation Notification is based on the following data source:

- a PostgreSQL Database instance - which is used for the persistence of messages, notifications, subscribers, subscriptions and taxonomies of topics.

The following three steps have to be followed in order to configure the persistence layer needed for the service to work correctly.

- STEP 1: from either the web interface or at command line, create a role with no privileges,

```
user: <notification_user>
password: <notification_pwd>
```

- STEP 2: create a database instance on PostgreSQL (e.g. named "notification_db") and assign its ownership to the role already created in the step 1,

```
db name: <notification_db>
db owner: <notification_user>
encoding: UTF8
```

As an alternative, you might use the following SQL script to execute both STEP1 and STEP2

```
CREATE ROLE notification_user LOGIN PASSWORD 'notification_pwd'
NOSUPERUSER NOINHERIT NOCREATEDB NOCREATEROLE;

CREATE DATABASE notification_db WITH OWNER = notification_user
ENCODING = 'UTF8';
```

- STEP 3: configure the persistence.xml (that can be found at assets\services\preservation-notification\src\main\resources\META-INF\persistence.xml) as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="preservation-notificationPU" transaction-
type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>

    <class>eu.europeana.assets.service.preservation.notification.dao.Tme
ssage</class>

    <class>eu.europeana.assets.service.preservation.notification.dao.Tno
tification</class>

    <class>eu.europeana.assets.service.preservation.notification.dao.Tpu
blisher</class>

    <class>eu.europeana.assets.service.preservation.notification.dao.Tse
gguid</class>
```



```
<class>eu.europeana.assets.service.preservation.notification.dao.Tse
rvice</class>

<class>eu.europeana.assets.service.preservation.notification.dao.Tsu
bscriber</class>

<class>eu.europeana.assets.service.preservation.notification.dao.Tsu
bscription</class>

<class>eu.europeana.assets.service.preservation.notification.dao.Tta
xonomy</class>

<class>eu.europeana.assets.service.preservation.notification.dao.Tte
rm</class>

<class>eu.europeana.assets.service.preservation.notification.dao.Tre
adby</class>
  <properties>
    <property name="hibernate.connection.username"
value="notification_user"/>
    <property name="hibernate.connection.driver_class"
value="org.postgresql.Driver"/>
    <property name="hibernate.connection.password"
value="notification_pwd"/>
    <property name="hibernate.connection.url"
value="jdbc:postgresql://localhost:5432/notification_db"/>
    <property name="hibernate.hbm2ddl.auto" value="update"/>
  </properties>
</persistence-unit>
</persistence>
```

For detailed information please refer to the [Setup_and_Installation_Guidelines.pdf](#) which is part of the source code subversion repository of the project.

4.4 User/Developer Manual for Notification

A guide on how to use the server-side service will be detailed in the next paragraphs. Regarding the user interface, a short user manual will be provided.

4.4.1 USER MANUAL (*how to use notification service home page, index.html*)

The available notification service methods are displayed in the notification service home page:

<http://<server.url>/assets/preservation-notification/index.html>

where <server.url> can either be:

- ☐ <http://assetstest.atc.gr> (test server)
- ☐ <http://assetsdemo.atc.gr> ("production" server)

Please refer to Figure 16, Figure 17, Figure 18 and Figure 19 for a series of screenshots of



the above mentioned service home page.

For details about all the operations that can be performed on this page (representing a way of using the notification service REST URLs), please refer to chapter 3 “The Preservation Notification Test Scenario (for CONTENT PROVIDERS)” in [NOTIFICATION TEST SCENARIO].

4.4.2 DEVELOPER MANUAL

This section is going to show some short JAVA code snippets taken from the notification test classes that will allow a developer to quickly understand how objects can be instantiated from the service classes.

2.4.2.1 Creating an external client service to access the server-side notification service

The server-side "notification service" has been conceived in order to make available its features to external client services. Each client service could have different taxonomies, subscribers and subscriptions.

So, it is necessary to instantiate a “client service” (or to have available a “client service” identifier). This is accomplished by using the method `createService` (through an instance of the `RegistrationManager` class) which input parameters are (Identifier `serviceId`, String `serviceLabel`).

```
RegistrationManager rm = new RegistrationManagerImpl();
Service s = rm.createService(new Identifier(serviceID), serviceLabel);
```

2.4.2.2 Creating a taxonomy

In order to create a taxonomy object, we have to access methods from a `taxonomyManager` object. These methods will be accessible either through a local instance of the `TaxonomyManager` class (if a service client is available)

```
TaxonomyManager tm = new TaxonomyManagerImpl();
```

or through a remote REST invocation of the `TaxonomyManagerImpl` interface:

```
String restURL = 'http://<server.url>/assets/preservation-notification/rest';
TaxonomyManagerImpl tm = new TaxonomyManagerImpl(restURL);
```

We will then use the `createTaxonomy` method which input parameters are (Identifier `serviceId`, `TaxonomyInfo` `ti`, `Taxonomy` `taxo`).

```
TaxonomyInfo ti = new TaxonomyInfo(title, descr, owner, 1);

Taxonomy taxo = new Taxonomy();
taxo.addTerm("Programming", null);
taxo.addTerm("Theory", "Programming");
taxo.addTerm("Languages", "Programming");
taxo.addTerm("Algorithms", "Programming");
taxo.addTerm("OOL", "Languages");
```

```
taxo.addTerm("Sort","Algorithms");
taxo.addTerm("C++","OOL");
taxo.addTerm("Java","OOL");
taxo.addTerm("MergeSort","Sort");
taxo.addTerm("BubbleSort","Sort");
taxo.addTerm("QuickSort","Sort");
taxo.addTerm("JSP","Java");
taxo.addTerm("JavaBean","Java");

Taxonomy tax = tm.createTaxonomy(new Identifier(serviceID), ti, taxo);
```

2.4.2.3 Creating a publisher

In order to create a publisher object, we have to access methods from a registrationManager object. These methods will be accessible either through a local instance of the RegistrationManager class (if a service client is available)

```
RegistrationManager rm = new RegistrationManagerImpl();
```

or through a remote REST invocation of the RegistrationManagerImpl interface:

```
String restURL = 'http://<server.url>/assets/preservation-notification/rest';
RegistrationManagerImpl rm = new RegistrationManagerImpl(restURL);
```

We will then use the method createPublisher which input parameters are (Identifier publisherId, String publisherLabel).

```
Publisher p = rm.createPublisher(new Identifier(pID), pLabel);
```

2.4.2.4 Creating a notification (for given terms of a taxonomy)

In order to create a notification object, we have to access methods from a notificationManager object. These methods will be accessible either through a local instance of the NotificationManager class (if a service client is available)

```
NotificationManager nm = new NotificationManagerImpl();
```

or through a remote REST invocation of the NotificationManagerImpl interface:

```
String restURL = 'http://<server.url>/assets/preservation-notification/rest';
NotificationManager nm = new NotificationManagerImpl(restURL);
```

We will then use the method createMessage which input parameters are (Identifier serviceid, Publisher p, ArrayList<Term> terms).

```
ArrayList<Term> terms = new ArrayList<Term>();
terms.add(new Term(new Identifier("term1")));
terms.add(new Term(new Identifier("term2")));
.....
```

```

.....
terms.add(new Term(new Identifier("termN")));

Publisher p = new Publisher();
p.setPublisherId(new Identifier(pID));
Notification n = nm.createMessage(new Identifier(serviceID), p, terms);

```

2.4.2.5 Creating a subscriber

In order to create a subscriber object, we have to access methods from a registrationManager object. These methods will be accessible either through a local instance of the RegistrationManager class (if a service client is available)

```
RegistrationManager rm = new RegistrationManagerImpl();
```

or through a remote REST invocation of the RegistrationManagerImpl interface:

```
String restURL = 'http://<server.url>/assets/preservation-notification/rest';
RegistrationManagerImpl rm = new RegistrationManagerImpl(restURL);
```

We will then use the method createSubscriber which input parameters are (Identifier subscriberId, String subscriberLabel).

```
Subscriber p = rm.createSubscriber(new Identifier(sID), sLabel);
```

2.4.2.6 Registering a subscription

In order to register a subscription, thus meaning to create a subscription object, we have to access methods from a registrationManager object. These methods will be accessible either through a local instance of the RegistrationManager class (if a service client is available)

```
RegistrationManager rm = new RegistrationManagerImpl();
```

or through a remote REST invocation of the RegistrationManagerImpl interface:

```
String restURL = 'http://<server.url>/assets/preservation-notification/rest';
RegistrationManagerImpl rm = new RegistrationManagerImpl(restURL);
```

We will then use the method createSubscription which input parameters are (Identifier serviceId, Subscriber s, ArrayList<Term> terms).

```

ArrayList<Term> terms = new ArrayList<Term>();
terms.add(new Term(new Identifier("term1")));
terms.add(new Term(new Identifier("term2")));
.....
.....
terms.add(new Term(new Identifier("termN")));

Subscriber s = new Subscriber();

```

```
s.setSubscriberId(new Identifier(sID));
Subscription n = rm.createSubscription(new Identifier(serviceID), s, terms);
```

2.4.2.7 Publishing messages (on a notification channel)

In order to publish a message on a notification object, we have to access methods from a notificationManager object. These methods will be accessible either through a local instance of the NotificationManager class (if a service client is available)

```
NotificationManager nm = new NotificationManagerImpl();
```

or through a remote REST invocation of the NotificationManagerImpl interface:

```
String restURL = 'http://<server.url>/assets/preservation-notification/rest';
NotificationManager nm = new NotificationManagerImpl(restURL);
```

We will then use the method publishMessage whose input parameter is (Notification n).

```
Notification n = new Notification(new Identifier(nID));
n.setMsgURI(msgURI);
n.setMsgTitle(msgTitle);
n.setMsgSummary(msgSummary);
nm.publishMessage(n);
```

2.4.2.8 Delivering messages/alerts (filtering rules)

In order to deliver messages on a notification object, we have to access methods from a notificationManager object. These methods will be accessible either through a local instance of the NotificationManager class (if a service client is available)

```
NotificationManager nm = new NotificationManagerImpl();
```

or through a remote REST invocation of the RegistrationManagerImpl interface:

```
String restURL = 'http://<server.url>/assets/preservation-notification/rest';
NotificationManager nm = new NotificationManagerImpl(restURL);
```

Delivering messages (alerts) to subscribers will then be accomplished by using the method deliverMessages whose input parameter are (Identifier serviceId, Identifier SubscriptionId, String filteringRule, int indexFrom, int maxBunch, MessagePolicyAge policyAge) where

- @param indexFrom index from which the alerts have to be returned.
- @param maxBunch max size of returned alters.
- @param policyAge specifies the age policy of delivered alerts.

```
List<Alert> la = nm.deliverMessages(new Identifier(serviceID), new Identifier(sID), fr, -1, -1, null);
```

5. Concluding Remarks

In this deliverable we have described the ASSETS services for the Digital Preservation, implemented and tested in ASSETS WP2.3.

The technical aspects of the following components have been explained in detail:

- Risk management service,
- Preservation Normalization service,
- Preservation Notification service.

The software requirements, the technical documentation (UML diagrams, services description and API documentation, the software packaging and installation), and the user manual have been provided for each service in order to allow developers to understand how to use these services, and to know the steps to follow during their installation and configuration process.

6. References

[D2.0.4] Deliverable 2.0.4 “The ASSETS APIs” – delivered by ASSETS in March 2011

[Notification JavaDoc] Notification JavaDoc - available on <http://assetsdemo.atc.gr/content/assets-javadocs/preservation-notification/>

[NOTIFICATION TEST SCENARIO] the TEST SCENARIO for “DIGITAL PRESERVATION NOTIFICATION SERVICE” available on the project’s WIKI at the following URL: http://62.101.90.79/c/document_library/get_file?p_l_id=11549&folderId=65911&name=DLFE-3113.pdf