Global System Architecture Report

D2.2.1

Deliverable Type: Report
Number: D2.2.1;
Nature: Public
Contractual Date of Delivery: 30 September 2001
Actual Date of Delivery: 12 October 2001
Revised: 03 March 2003
Task WP2.2:

Name of responsible: Donatella Castelli
Pasquale Pagano

CNR-IEI
Area di Ricerca di Pisa
56124 Pisa
Italy

castelli@iei.pi.cnr.it
pagano@iei.pi.cnr.it

Abstract:
This is the Global System Architecture Report. It presents the activity carried out as part of task T2.2. The overall system architecture is introduced by describing its components and their distribution. These architectural components are specified in terms of their abstract state and the set of service requests. Each service request is specified in terms of its input, output and behaviour.

Keywords: digital library architectures, open architectures, open protocols, digital library services, multimedia documents, document annotations, cross-language access, personalised dissemination.

TABLE OF CONTENTS

1. INTRODUCTION

This report presents the work of Task 2.2. The aim of this task is to specify the Scholnet architecture.

The report is organised as follows:

Section 2 lists the general requirements that have influenced the architectural choices.

Section 3 briefly motivates the architectural and protocol design decisions.

Section 4 presents the document model adopted in Scholnet.

Section 5 briefly presents the OpenDLib Application Profile (OLAP).

Section 6 presents the overall architecture.

Section 7 provides a specification of the format of the communication protocol.

Sections 6 to 20 present the specifications of the architectural components.

Section 21 describes the functionality and architectural details of the metadata editor tool that will be implemented to support the editing of document metadata descriptions.

Section 22 briefly discusses possible extensions to the current version of the system.

Appendix A defines the list of acronyms used in this report.

Appendix B specifies the metadata application profile (OLAP) used to support the realisation of the Scholnet services.

Appendix C specifies the mapping from the ETRDL metadata format to OpenDLib Metadata Set.

Appendix D specifies the metadata schema that describes the structure of Scholnet documents.

Appendix E introduces the metadata format that has been used to describe users.

Appendix F introduces the metadata format that has been used to describe groups.

Appendix G introduces the metadata format that has been used to describe collections.

Appendix H introduces the metadata format that has been used to describe thesaurus.

Appendix I introduces some DTDs used to validate the OLP protocol response.

2. REQUIREMENTS

The architectural specification described in this report is the interpretation of the Scholnet Functional Specifications [ScholnetD2.1.1] as established by the project Technical Annex [ScholnetTA]. This context imposes certain constraints. In particular:

"*The SCHOLNET project will extend the basic service provided by the ERCIM Technical Reference Digital Library (ETRDL) [ETRDL] with tools that implement a new set of services to handle multimedia digital objects and to provide a collaborative working environment.*
ETRDL was implemented using the Dienst technology[Dienst], developed by a US Consortium led by Cornell University. Dienst version 4.1.9 provided functionality for archiving, access, discovery and browsing. ETRDL added capability for on-line document submission/withdrawal, subject classification, multiple language indexing and search, and on-line administration."

The design of the Scholnet architecture has been constrained by the ETRDL basic architecture. The design decisions were made in accordance with the existing ETRDL underlying design principles, and its architectural and protocol choices.

"*SCHOLNET will experiment an open digital library infrastructure. The openness permits the partitioning of the total system functionality into a set of well-defined services. Each of these services is accessible via a well-defined protocol – a set of service requests – that defines the public interface to this service. Furthermore, each of these service requests is documented with respect to the format of the request, the format of the possible responses and exceptions, and the semantics of the request"*.

"A service is instantiated by a server module, which implements the set of service requests defined for the service. The actual implementation of a server is opaque and irrelevant from the perspective of interoperability, the ability of the service to communicate via its protocol with other services, and clients. In addition, an individual server may be distributed or replicated, the nature of which is also opaque from the perspective of its use. The exact topology of the SCHOLNET servers is a design choice and will be decided during the system specification phase of the project."

Openness, i.e. the possibility of easily integrating other services in the architecture, is one of the main design principles of ETRDL and has been strictly maintained in the design of the Scholnet architecture. Scholnet will be a provider of a core set of services that, if required, can be further enriched in order to satisfy the needs of specific communities.

"An extensive experimentation activity will be conducted in order to assess the degree to which the SCHOLNET system functionality and performance meet the requirements of a large scholarly community. In this activity, the ERCIM networked scholarly community extended by the academic members of the DELOS Network of Excellence (NoE) on Digital Libraries, recently approved for funding by the 5th FP/IST, will be involved."

Great care has been taken to maintain the compatibility with the ETRDL document repositories in order to set up a valuable experimentation. In this way, all the documents accessible through the ETRDL search and browse services will also be accessible through the analogous Scholnet services.

"Dienst will constitute the reference architecture used by the Open Archive initiative."

At the time of the presentation of the Scholnet proposal, the Open Archives Initiative(OAI)[OAI] had adopted a subset of Dienst as the protocol for harvesting data from different heterogeneous archives. As a consequence of this choice, all the Scholnet repositories would have been automatically exposed to all the OAI compliant services.
Recently, however, the rules established by the Open Archives Initiative(OAI) have been modified: a new protocol, called Open Archives Initiative Protocol for Metadata Harvesting has been

specified. This is a minimal protocol that supplies and promotes an application-independent interoperability framework that can be used by a variety of communities engaged in publishing content on the Web. Each archive that implements the protocol becomes accessible by services implemented by third-parties through the invocation of protocol requests. This protocol has great potentiality, since on one hand the information stored in an archive is made accessible through several services, on the other hand the services developed on top of the protocol can provide access to a growing number of archives.
The Scholnet architecture has thus been designed so that Scholnet repositories can easily be transformed into OAI compliant ones.

As can be seen from a statement of the Standardization Forum of the DELOS NoE, interoperability is becoming one of the most important quality issues of a digital library.

"A number of emerging Web standards will provide much of the basic architecture for digital libraries (RDF, Dublin Core, INDECS, DIENST protocol, UNICODE, XML, Z39.50, etc.). Many of these standards have just begun to move from research to deployment. Implementers need to follow the progress of research, while researchers need to monitor the experience of early adopters. The refinement of a stable architecture for digital libraries will require several iterations as these standards are adapted for various applications." (http://www.ercim.org/delos/).

The Scholnet architecture has been designed taking into account the emerging proposals for interoperability standards in the digital library area.

3. FROM ETRDL TO SCHOLNET

The original idea for the Scholnet project arose from our experience in designing and constructing the ETRDL digital library. The aim of the project was to extend ETRDL with new services able to provide a better support to scholarly communication and collaboration activities. However, the Dienst software, which was used as a basis for building ETRDL, was found to be insufficient to support a multimedia, multilingual digital library. Therefore, CNR decided to redesign and reimplement the basic software in order to provide a better basis for Scholnet. This new software is called *OpenDLib*.
The aim of the Scholnet project is thus now to port the distinguishing features of ETRDL to OpenDLib and implement the new Scholnet-specific services. OpenDLibThe rest of this document presents the complete specification of the Scholnet architecture.

Before presenting the Scholnet architecture, two preliminary concepts have to be introduced: the document model (DoMDL) and the metadata schema application profile (OLAP). These are described in the next two sections.

4. THE DOCUMENT MODEL

Scholnet will handle multimedia structured documents. Annotations will be treated as external objects that refer the document versions with which they are associated**.**
The documents will conform to the Document Model for Digital Library (DoMDL). This document has already been described in [SCHOLNET D2.1.1]. Here, for clarity, we include a slightly enhanced version of this description. The DoMDL is illustrated in Figure 1. DoMDL represents a document through a set of entities (ovals) that model the document from different points of view, a set of attributes for each entity (not depicted into the figure), a set of relations (arrows) that link the different entities and a set of constraints.

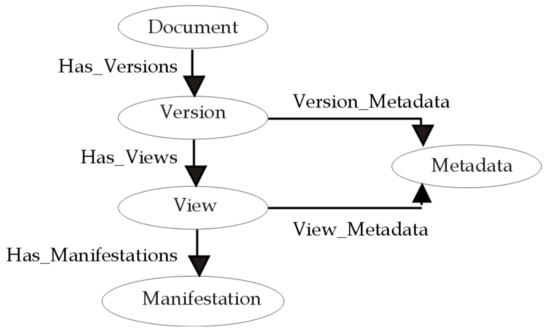


Figure 1: The Document Model for Digital Library

The following sections present a detailed description of the model components.

4.1 *ENTITIES*

Each of the entities described below models a particular aspect of a document.

Document

A document is an abstract entity that represents a distinct intellectual creation. A document is named by a URN (Unified Resource Name). Unlike a URL, a URN is location independent.

Version

We recognize a document through its individual instances along the time dimension. An instance is a given edition of a document (i.e. prepared for publication). The first instance of a document could be, for example, a document that describes an idea in a draft form; the second one could be the document reviewed after experimentation; and the last one the final published version.
An instance of a document is called a *version*. Versions are linear and numbered. The first version is version 1, subsequent versions are 2, 3, etc.

View

Each view is a *specific intellectual expression* of a document instance. The boundaries of the view are defined so as to exclude physical aspects that are not integral to the intellectual perception of a document.
Every document instance is perceived through *one or more* views. A document instance may be perceived by a user, for example, through its description, or its summary, or its full content.

In DoMDL we distinguish between two different classes of view: *Metadata*, and *Content*. The content view can be a *Body* a *Reference* or a *Choice* view. Each of these classes models specialised views that have different constraints and relationships (see Figure 2).

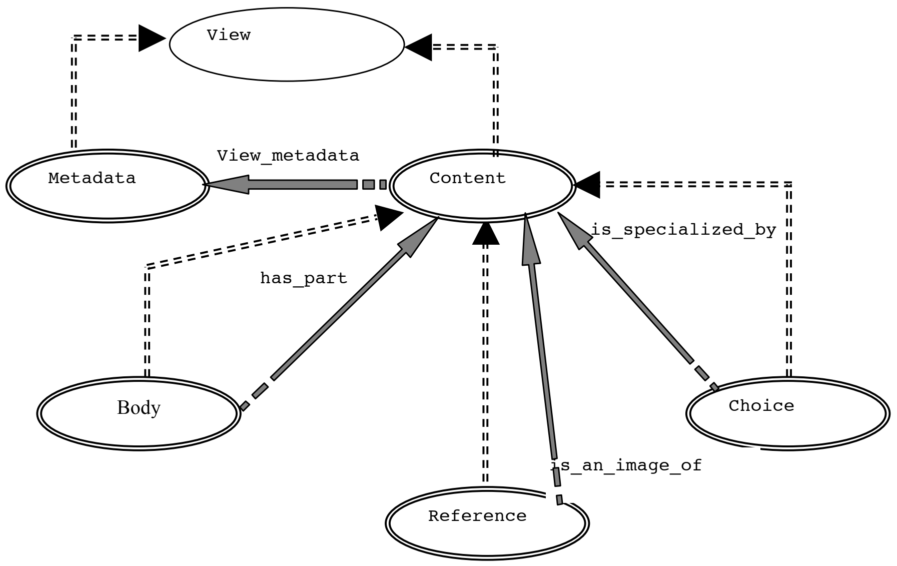


Figure 2: The View Entity, its Specialised Entities and its Relationships

Metadata View
The Metadata View provides descriptive information about the document instance and its views. Typically, index services or other discovery or browsing services use this cataloguing information to facilitate location of digital objects in a digital library.

Content View
The Content View models the specific intellectual form taken by the content of the document in order to be perceived by the user. This view may be used to represent, for example, perceptions of the same text in different languages, a silent perception of an audio-video document or its audio summary, etc.

Body View
The Body View represents the intellectual form of documents either as a whole or as the aggregation of other views. In the former case it is associated with at least one manifestation while in the second case its manifestations are derived by aggregating the manifestation of its parts.

Reference View
A Reference View is a content view that has all the properties and relations of an already existing view. For example, the view ArticleOne of an electronic journal might be exactly the same as the view ArticleFive published in the proceedings of a conference. A Reference View has no assigned explicit property except the URL of the existing view. This provides all the information that is required to retrieve the complete description of the view modeled as reference. Its manifestations can be derived by the manifestations associated with the existing view.

Choice View
A Choice View is a content view, which may be further specialized. For example, if a ChapterView is represented as a Choice View then this means that there are other more specialized views of it. These might be a PagedChapterView, a view of the chapter decomposed in pages, or a Section View, a view of the chapter decomposed in sections.

Manifestation

A manifestation *is the physical form* of an intellectual perception of the document.
Each manifestation may be disseminated under the form of either a single item or a set of items. The items are identified as a sequence with an ordinal attribute. The content type of the items in the set must be the same. A content type is expressed as a MIME (Multi-purpose Internet Mail Extension) type such as *image/tiff* or *text/html*.

4.2 *RELATIONS*

The entities described above are linked by the following relations.

has_version
The domain of the *has_version* relation is the *document* entity and its range is the *version* entity.
A specific document has at least one version and the same version cannot be the instance of two different documents.

version_metadata
The domain of this relation is the *version* entity and its range is the *metadata* view entity.
The version_metadata relation is used to associate metadata with a specific version in order to describe it.

has_view
The domain of the *has_view* relation is the *version* entity and its range is the *view* entity.
The *has_view* relation is used to express how a document instance may be perceived by a user.
Every document instance is perceived through one or more views. A book, for example, may be perceived by means of its metadata descriptions or by means of two or more content views in different languages.

view_metadata
The domain of this relation is the *content* view entity and its range is the *metadata* view entity.
The *view_metadata* relation is used to associate metadata with a specific content view in order to describe it.

has_part
This is a relation from *body to content.*
The *has_part* relation makes it possible to manage views that are realized as a hierarchy of parts that are slices of a single intellectual creation. It describes a view as an aggregation of a set of other views: for example, a book view of a document may be perceived as organized into a set of parts (chapters) where each of these parts, in turn, is a set of parts (paragraphs).
Sharing of parts is not possible. For example, a specific paragraph cannot be part of two different chapters.

is_an_image_of
This is a relation from *reference to content.*
The *is_an_image_of* relation is used to indicate that one view is the image of another. As such, it has exactly the same metadata description, structure and relations as the view to which it refers. For example, if each of the articles of a journal has been registered as an independent document, the

journal view may be defined as the aggregation of the set of image views referring the existing views of the articles in the journal.
Due to its definition, a view can be the image of at most one view; it does not have its own metadata, parts, specialisation nor manifestation.

is_specialized_by
This is a relation from *choice to content.*
A view element can be perceived by the user in various intellectual forms which may differ with respect to the level of detail. The *is_specialized_by* relation links a view to its specialization. A journal, for example, may be perceived as a set of articles or as a set of issues: the two sets can be modeled as specialised views of the journal view.

has_manifestation
The domain of this relation is the *view* entity and its range is the *manifestation* entity.
The *has_manifestation* relation is used to express the relationship between the intellectual and the physical form of a document. For example, a textual view of a document may have two different manifestations: one in Adobe's Portable Document Format (pdf) format and another in postscript format (ps).
Each view with no parts has at least one manifestation.

4.3 *ATTRIBUTES*

Each entity of the model has an attribute *name* which is an alphanumeric string. The names of the objects that are related to the same object must be different. For example, the views of a givendocument instance must have different names and the parts of a given view must have different names.
Other specific attributes are reported in the table below.

Entity	Attribute		
	Name	Type	Description
Version	*ordinal*	Integer (mandatory)	Specifies the number of the version.
View	*display*	String (optional)	Brief human readable description of the content of the view.
	ordinal	Integer (optional)	Can be used to establish an ordering among different parts of a same view, e.g. the chapters of a book, the paragraphs in a chapter.
Manifestation	*content- type*	MIME type (mandatory)	Specifies the content type of the manifestation.
	display	String (optional)	Brief human readable description of the manifestation.
	multiple	Boolean (optional)	This flag is “on” when the manifestation is available as a set of files, it is off when the manifestation is available as a single file.

	uri	String (optional)	Unique Resource Identifier of the manifestation. It can be used to specify the name of a file, or the complete ftp URL where the file can be downloaded, or the specific URN used to identify the manifestation.

5. THE DOCUMENT METADATA DESCRIPTION

The content of Scholnet documents will be described by metadata records according to the OpenDLib Application Profile (OLAP). An application profile is defined as a metadata schema which consists of data elements drawn from one or more namespaces, combined by implementors, and optimised for a particular local application.
The idea of application profiles grew out of UKOLN's work on the DESIRE project [DESIRE][Heery00] and is currently one of the most well accepted methodological solutions for the definition of new metadata schemas. Application profiles allow implementors to declare how they are using standard schemas.

The namespaces used in OLAP are the Dublin Core namespace, the Dublin Core Qualified namespace, and the OpenDLib Metadata Set namespace (OLMS). This last metadata set has been introduced to support the specific needs of the OpenDLib Architecture services.
Appendix B contains the complete OLAP and an XML DTD that specifies it.

The ETRDL repositories contain metadata descriptions given in a subset of the RFC1807 format extended with a few additional fields for handling multilinguality. A mapping between the ETRDL metadata format and OLAP has been established in order to be able to render these metadata descriptions available in the Scholnet digital library. This mapping is given in Appendix C.
Interoperability with applications that require Dublin Core metadata descriptions (such as those that are Open Archives compliant) can be obtained simply by restricting the metadata record to only those fields that belongs to the “dc” and “dcq” namespace and removing the qualifiers.
The system must also maintain a description of the document structure. In Scholnet, this structure will be described according to the Structure Metadata Set (SMS). The SMS will specify how many versions are available for each document, their views and, for each view, the corresponding structure. The SMS DTD is given in Appendix D.

6. THE SCHOLNET ARCHITECTURE

6.1 *THE SCHOLNET SERVICES*

Scholnet will be built as a distributed digital library, according to the notion of individually defined services located anywhere on the Internet. When combined, these services constitute a digital library. The functionality of the Scholnet digital library includes the storage of and access to multimedia and multilingual resources, annotations on these resources, cross-language search and browsing, user registration and personalized information dissemination of new incoming documents.

The Scholnet federation of services will communicate through an established protocol. Figure 2 shows a conceptual model that specifies the notion of "Scholnet service". The ovals represent classes of services, the double arrows the specialisation relationship. A Scholnet architecture consists of a set of instances of the leaf classes (service types)of this model.

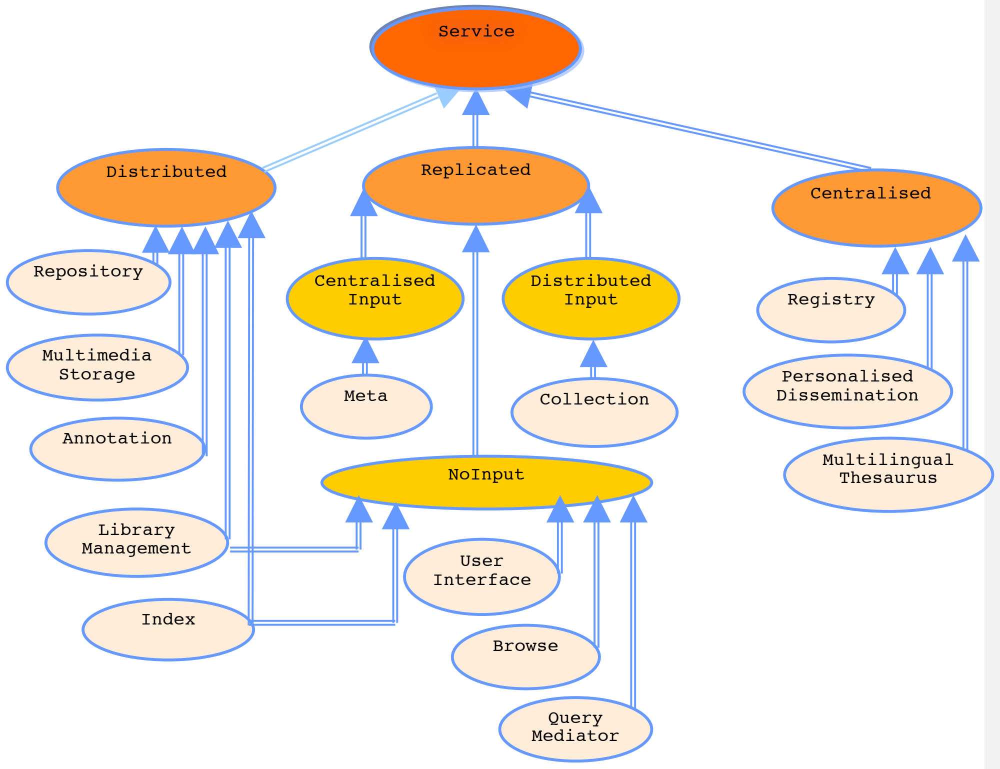


Figure 2. The Scholnet Service Conceptual Schema

This model is central to the digital library system design since it highlights the properties (descriptive metadata) that define each service. Each service instance is known by the other instances through the values of these properties, which are disseminated on demand. These properties are modeled using structured attributes that are associated with each entity of the model. As a natural consequence of the relation between entities, a child entity inherits all attributes of its parent entity. The whole set of attributes characterize an instance of a service in the Scholnet architecture. The list of these attributes is given below.

A generic service of the Scholnet architecture is modeled by the entity *Service*. A service can be distributed over different servers, replicated, or if necessary centralized. The model has an entity for each service type.

Distributed services are those that implement the same service through multiple instances, each of which manages data stored on a different server. The data are distributed according to a set of criteria that may differ from service to service. Note that each instance of a distributed service does not need to know anything about the other instances of the same service. In the version of Scholnet that we are illustrating, the services that are distributed are those that maintain a huge amount of data and/or those that are strictly related to the document publishing institutions. These institutions usually prefer to maintain their own documents on their own server to have a physical control over them. Moreover, each institution usually has its own rules for document submission/withdrawal, or content management, and therefore prefers to maintain these procedures also in a common shared environment.

The replicated services are those implemented by a set of service instances, possibly located on different servers, where each instance is able to cover completely the service functionality over the entire set of data. Scholnet distinguishes three kinds of replicated services: *NoInput*, *CentralisedInput*, or *DistributedInput*. A service is of NoInput type if it is instantiated by pure replications, i.e. the different instances are never distinguishable since they handle the same data and behave in the same way. A service of CentralisedInput or DistributedInput type has one replication, which acts as a master, and a set of replicates which act as slaves. In the case of the CentralisedInput, the master is a special instance of the service whose only purpose is to maintain and distribute on demand an updated version of the information handled by the service. The slave instances update their content by periodically invoking the master. Both the master and slave replicates of a DistributedInput service can accept new information and serve information requests. The master maintains the global state of the service information: each time a slave updates its local information, the slave communicates the change to the master which merges the new information with its own information. Periodically, each slave updates its state by invoking the master. The role of the instances of a replicated service, i.e. master and slave, is not statically assigned but can be changed in order to achieve a better connectivity or to overcome temporary crash. In the present version of Scholnet we have chosen to replicate those services that are either not constrained by any proprietary (see distributed services), security or privacy constraint (see centralized). This replication makes it possible to improve service efficiency and to increase its robustness. This is, for example, the case of the replication of indexes to improve content access, or the replication of meta information to improve digital library service access, and so on.

A centralized service has always a single instance in the digital library. Each time the security and the privacy of the content of a service is an issue, the centralized solution is preferred to the distributed and replicated ones.

Although the presence of multiple instances of a service increases fault tolerance, reduces the overload of each instance, and makes it possible to dynamically reorganize the environment when a server hosting a service instance is not reachable, the replication and distribution of the services is not mandatory and therefore each of the services outlined in Figure 2 can be instantiated as a single instance. This means that the level of distribution and replication, and the physical location of the service instances may be freely chosen to better satisfy the needs of the specific digital library context

Each class of service has a set of characterising attributes:

Class			Attributes
Service			serviceName, olp_base_url, textualDescription, submissionProcedure, harvestingProcedure, metaServiceURL, useRestriction, protocolVersion, adminEMail, verbsInfo
	Centralised		
	Distributed		AuthorityNames
	Replicated		regionName, prioritySet
		NoInput	
		CentralisedInput	masterYesNo, masterURL
		DistributedInput	masterYesNo, masterURL, copyURLset

The leaf classes inherit the properties of their ancestors and have their own additional characterising properties (see below).

The functionality of each service type is the following:

- *Repository* - stores documents that conform to the DoMDL document modelIt can be distributed on multiple servers because a Repository server can store documents published by different authorities and different authorities can be hosted by different Repository servers.
- *Multimedia Storage* - stores video documents (according to the DoMDL document model), and supports their dissemination either as whole documents or as aggregations of scenes, shots and frames. It can be distributed on multiple serversbecause a Multimedia Storage server can store documents published by different authorities and different authorities can be hosted by different Multimedia Storage servers.
- *Library Management* - supports the submission, withdrawal, and replacement of documents. It can be distributed on multiple servers because a Library Management server can manage documents of different authorities published in different Repositories and different authorities can be managed by different Library Management servers. It is replicated because the same authorities can be managed by different LibMgt servers.
- *Index* - accepts queries and returns lists of document identifiers matching those queries. It can be distributed and replicated on multiple servers because an Index server can index documents published by different authorities stored in different Repositories and because the document published by different authorities can be indexed by different Index servers.
- *Query Mediator* - dispatches queries to appropriate index servers. It can be replicated on multiple servers.
- *Browse* - supports the construction of browsing indexes and the actual browsing of those indexes on library contents. It can be replicated on multiple servers.
- *Registry* - supports the storage and access of information about authors, individual users, and user groups. Hosted by one server.
- *Personalised Dissemination* - supports the storage and execution of persistent queries, subscriptions, on a collection. Hosted by one server.
- *Annotation* - stores annotations on documents and makes them available to authorised users. It can be distributed on multiple servers because an Annotation server can store annotations on documents published by different authorities that are stored in different repositories.

- *Multilingual Thesaurus* - maintains language resources, such as multilingual thesauri, which are required to support cross-language functionality. Hosted by one server.
- *Regional Meta Service* maintains the information pertaining to the regions (see below). It is replicated on multiple servers, one for each region.
- *Collection Service* - provides a virtual organisation of the documents stored in the repositories. It supplies the information necessary to manage these virtual document aggregations. This information is used by the other services in order to handle the collection objects allowing, for example, the Query Mediator to perform a query on a specified collection, or the Browse to perform a browse on a collection, and so on. It can be replicated on multiple servers.
- *User Interface*- mediates human interaction with these services and their protocols.

Each type of service has descriptive attributes that specify its content. The table below lists the specific attributes associated with each service type:

Service type	Specific Attributes
Repository	contentDescription, authorities, sets, metadataFormats, documentStructure
Library Management	authority,set,documentStructure, metadataFormat
Index	metadataFormat, indexedFields, resultFormats, language
Multimedia Storage & Delivery	compositeDocumentFormats, nonCompositeDocumentFormats, multimediaDocumentFormats
Query Mediator	searchMethods, resultFormats
Regional Meta Service	serviceDescriptionFormat
Collection	collectionMetadataFormat
Browse	metadataFormat, browsableFields, resultFormats
User Interface	frameSet
Registry	userProfileFormat, groupProfileFormat
Personalized Dissemination	topicThesaurus
Annotation	authorities
Multilingual Thesaurus	topicThesaurus, languageCovered

The services provided in a Scholnet digital library are instantiations of the listed service types. Each service has thus all the attributes declared for its type.

An Scholnet digital library is implemented as a federation of services that may be located on different servers. The federation is instantiated when the digital library is created and can change dynamically during the digital library lifetime. For example, a new server may be added or removed, a Query Mediator can send its search requests to a different Index, a service instance can change its role from master to slave, etc.

A Scholnet digital library architecture is described by the model introduced below. This model specifies the legal configurations of the Scholnet digital library system, i.e. the digital libraries that can be created with this system, and their possible evolutions.

A Scholnet digital library architecture model uses three concepts: *Service* (and its specializations, as introduced in the previous section), *Server*, and *Region*. A Server is a network device that is able to provide services to the network users by managing shared resources. It can host different service instance types. A Region is an abstract notion which stands for a dynamic set of service instances which cover the complete functionality of the digital library and which represent the optimal choice with respect to some set of optimization criteria, e.g. their mutual connectivity. A region thus consists of the entire set of centralized and distributed service instances and a set of instances of the replicated services, one for each service type. For each region, the set of replicated service instances changes over time in order to always implement the best choice, e.g. according to the state of the connection.

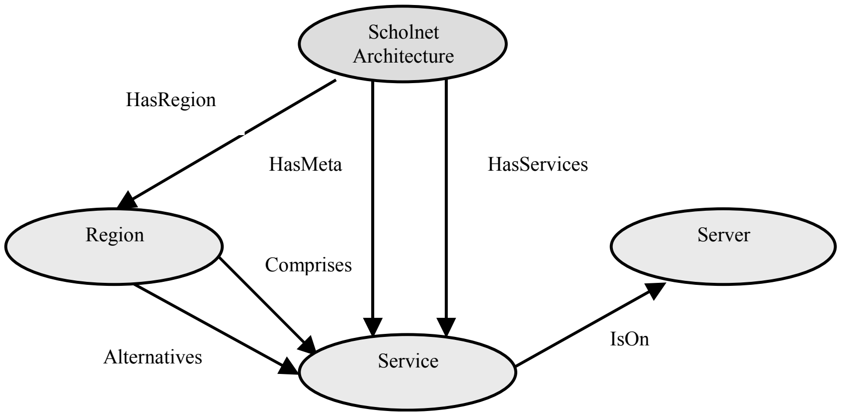


Figure 3: The OpenDLib architecture model

The Scholnet architecture model is shown in Figure 3. Any configuration that agrees with this model is a legal configuration of an Scholnet digital library system, i.e. an Scholnet digital library.
The *Service* entity and its specializations have been described above. Note that, in in order to simplify Figure 3, we have not included the whole service hierarchy. Each of the *Server* and *Region* entities has an attribute, Address and Name, respectively, which identifies it. Specific relationships link a service instance with the server that hosts it (*IsOn*), and a region with both its current service instances (*Comprises*) and with the possible alternative replicated service instances (HasAlternatives). This relationship indicates the service instances that can be used to replace the current ones when the optimization criteria are not met anymore. For each region, more than one alternative of the same service type can be indicated. A priority value is associated with each pair (region, service instance) as a measure of the quality of participation of the service in the region with respect to the set of optimization criteria selected. The service with the highest priority belongs to the region.
Note that the same service instance can belongs to, and be an alternative in, more than one region. This means that the number of replications can be chosen freely and is not constrained by the number of established regions.
The *Scholnet Architecture* entity models a digital library architecture created by instantiating the Scholnet system. Each digital library has a name and participates in three relationships which specify the digital library functionality and distribution. The relation *HasService* expresses the composition of the federation, i.e. the service instances that working together provide the digital library functionality; HasRegion models the organization of the instances into a cluster of services that satisfy optimization criteria; *HasMeta* identifies the services that control of the architecture and are responsible over time for its consistency.
Note that the only constraint on the number of participants in a relationship is that they must be sufficient to cover the digital library functionality. This means that an Scholnet digital library architecture is completely flexible in terms of the number of instances, regions and servers. These can be freely chosen when the digital library is created and they can be modified dynamically

during the digital library lifetime. Note also that the part of the Scholnet model depicted in Figure 3 does not impose any constraint on the type of services implemented by the federation. In presenting the model we have assumed the service types implemented by the current version of Scholnet, however other choices are equally supported.
The entities introduced above are subjected to a number of rules that must be satisfied in any state of any Scholnet digital library architecture. Two rules are reported below as an illustrative example.

Rule 1:
The Query Mediator service is parametric with respect to the type of search supported, the query language, the metadata formats, and the format of the returned result set. All the replicated Query Mediator instances usable in a region must select the same search types, result set formats, and the same sub-set of the metadata formats that are handled by the Repository service instances. As a consequence of this, different Query Mediator instances can only offer different search types, or return different result set formats only if they belongs to different regions;

Rule 2:
The Index service is distributed and is parametric with respect to the distribution criteria, to the retrieval engine supported, to the metadata formats, etc. Clearly an Index Service instance has to manage a metadata format that is used by the whole set of Repository instances identified by its distribution criteria. Nevertheless, as it is replicated, all Index instances that are usable in a region must refer to the same result set formats, and to the same indexed fields. For example, if an Index Service instance indexes documents in Italian, it will use a stop-word list for Italian, and Italian stemming rules, As a consequence, only an Index Service instance with the same configuration can be set-up as its replication.

Figure 4 and 5 show an example of interaction among the Query Mediator, Index, Browse, Repository, and Library Management services driven by the Meta Server. Following the indication reported in the map harvested from the Meta Server, the User Interface selects the appropriate servers to which to send its requests. By changing the map information, the Meta server can modify the routing of any request in order to improve connectivity between servers.
Figure 4 shows the allocation and the communication paths among the service instances at time T0. According to the instruction received by the Meta service, the User *UI 1* uses the query mediator hosted by the *Server 1* in order to process the user queries. At the same time, this user interface uses the browse service hosted by the same server.

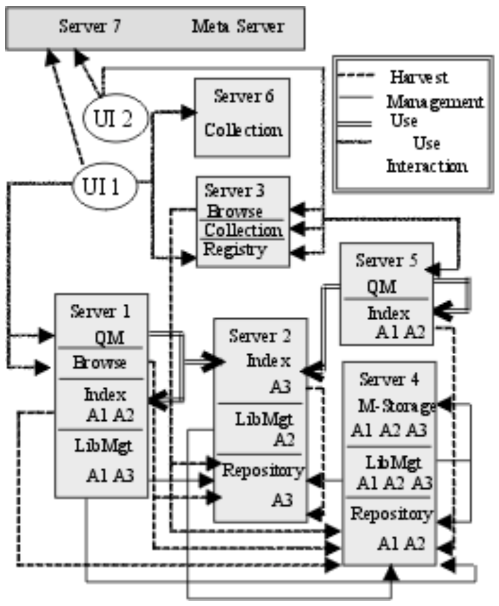


Figure 4: Dynamic routing adapting to overcome a server crash

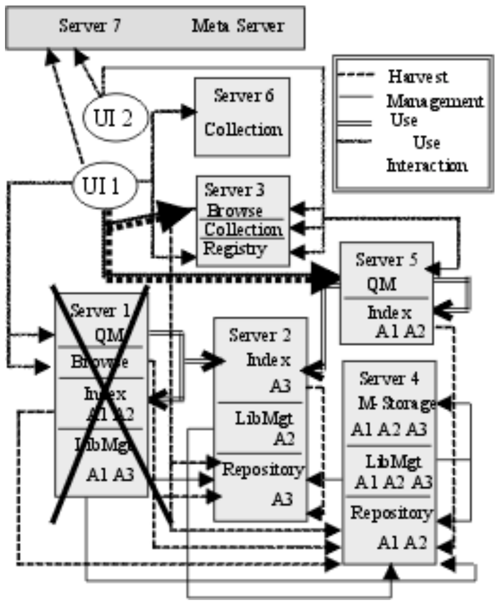


Figure 5: Dynamic routing adapting to overcome a server crash

At time T1 a temporary crash of the *Server 1* is recognized by the *UI 1* that reports the detected fault to the Meta Server. Using this information, the Meta signals alternative servers for each service instance hosted by that server.
Figure 4 shows the digital library at time T1: the UI 1 uses the query mediator hosted by *Server 5* to process the queries, and *Server 3* to allow the users to browse the library. When the temporary crash of the server is remedied, the Meta retracks the routing in order to re-optimize the communication between services.

7. THE OPENDLIB PROTOCOL

Communication with and among individual Scholnet services takes place via the OpenDLib Protocol (OLP). The OpenDLib Protocol is an evolution of the Dienst protocol. It inherits from Dienst the basic rules and conventions, and many protocol requests.

This section illustrates the protocol rules inherited from Dienst. A detailed description of the protocol requests that are served by each single service is given in the next sections.

7.1 *VERBS AND VERSIONS*

OLP protocol requests are called "verbs". Each service supports a set of verbs.
A service may support more than one version of a verb, and each version may differ in syntax or semantics. A version takes the form of two integers, separated by a period. This version applies to the individual verb, not the protocol as a whole. Including a version number in the message allows for backward-compatible extensions to the OLP system.
A server offering one or more OLP services might support verbs in various versions. A service receiving a message with a version number which is previous to the current one must reply either using the pertinent syntax and semantics, or with an error. If a service receives a message with a newer version number, then it must return an error.
Software supporting the OLP protocol may or may not be versioned. If a software version number exists, that number is independent of the OLP protocol verbs and versions of those verbs that the software supports. OLP protocol requests are expressed as URLs embedded in HTTP requests[1]. Except where noted, these are all HTTP GET requests. A typical implementation uses a standard Web server, such as Apache, that is configured to dispatch OLP URLs to the appropriate OLP service. The remainder of this section describes the aspects of the protocol that are specific to the HTTP embedding.

Message format

All messages are encoded in URLs where the path portion of the URL consists of the following tokens, in the following order:

`OLP:` This token appears literally in the URL.

`Service Name:` The name of the service that can handle the message, e.g. Repository.

`Version:` The version of the verb being invoked.

`Verb:` This is the name of the message, e.g. Structure. A verb is unique within a Service.

`Fixed arguments:` Each verb can have a certain number of fixed arguments, which must always be supplied, and must appear in the order cited.
`Fixed_post arguments:` Each verb can have a certain number of fixed arguments, which must always be passed as HTTP POST request. The fixed_post syntax arguments take the form key=value.

`Optional arguments:` the optional syntax arguments take the form key=value. If there is more than one optional argument, they are separated by an ampersand. Arguments may appear in any order. Unless otherwise specified, optional arguments are always optional and need not be repeated.

[1] As a future extension we are evaluating the possibility of embedding in HTTPS those requests that imply a user identification. Our preliminary analysis indicates that the existing software framework can support such an extension.

`Optional_post arguments:` the optional_post syntax arguments take the form key=value. Arguments may appear in any order and must always be passed as an HTTP POST request. Unless otherwise specified, optional_post arguments are always optional and need not be repeated.

The separator between tokens in the path is the slash, except for the separator before the Optional arguments, which is a question mark.

Example:
If the *Repository Service* implemented the Structure verb, and if version 1 of that verb accepted one fixed argument and two optional optional arguments (`version` and `view`), then an example request is:

/OLP/Repository/1.0/Structure/handlecorp/docid?version=2&view=book.

The full URL for this request at a particular Web server might be:

http://xx/OLP/Repository/1.0/Structure/handlecorp/docid?version=2&view=book.

Special characters

The syntax rules for URIs[2] give special roles to a few characters in certain contexts. If these characters are used in any other way they must be written as an escape sequence: a percent sign followed by the character code in hexadecimal. The special characters are.

Character	Role	Escape Sequence
/	Path Component Separator	%2F
?	Query Component Separator	%3F
#	Fragment Identifier	%23
=	Name/Value Separator	%3D
&	Argument Separator in Query Component	%26
:	Host Port Separator	%3A
;	Authority/Set Namespace Separator	%3B

The space character may not appear anywhere in a URL. It must be written with a "+" (or with the percent sign escape sequence %20.)
Note that in the examples used throughout this document, special character escaping is shown.

Message Responses

Responses to messages are formatted as HTTP responses, with appropriate HTTP header fields. The return type specified for each message in this document will, therefore, be the MIME type included in the HTTP Content-Type header field (if a wrapper is applied to the content, the Content-Type will correspond to the type of the wrapper). Likewise, any encoding applied to the content will be included in the HTTP Content-Encoding header field.

MIME Types

Responses to OLP protocol requests vary among the following MIME types:

- text/plain is used for responses that contain unstructured information.

[2] For a detailed description of the syntax, see http://www.ietf.org/rfc/rfc2396.txt

- text/xml is used for responses that contain structured information (such as the verb requesting the internal structure of a digital object). This document lists the DTD (Document Type Definition) for every verb that returns a text/xml response. All XML responses to OLP protocol requests have the following uniform features.
  The first tag output is an XML declaration where the version is always 1.0 and the encoding is always UTF-8. The remaining content is enclosed in a root element that has the same name as the verb of the respective request. The element has a single attribute named version, which has a value that is the version of the verb of the respective request. For example, a Disseminate verb with version 2.0 will produce text/xml content with a tag like <Disseminate version="2.0">
- text/html is used in response to user interface service requests (that are intended for rendering by a browser)

Content specific types such as application/postscript and image/gif are reserved for disseminations from digital objects.

Status Codes

Status codes and error returns correspond to those defined for HTTP (see the following list). A normal response from an OLP message in HTTP is signaled with the 200 reply code. Error returns are signaled with the appropriate 4xx or 5xx code as specified in the HTTP protocol. The use of HTTP error codes is as follows:

`400` - if the OLP request is malformed; for example, illegal arguments or the values of arguments are invalid.

`401` - if the client is unauthorized to make the request.

`404` - if a document specified in a OLP repository request is not in the repository.

`415` - if the format, encoding, or binder requested for a document is not available or cannot be generated.

`501` - if the OLP service, verb, or version is not supported by this server.

`503` - if the server is able to automatically generate a requested content type, but does not have a copy on hand, and does not wish to keep the connection open while it is generated, it should return the HTTP status code 503 (Service Unavailable), along with a suitable Retry-After header.

For each error return, the HTTP reason-phrase returned with the code should provide additional useful information to a human reader.

7.2 *DATES*

All dates in the protocol requests and responses are encoded using the "Complete date" variant of ISO8601. This format is CCYY-MM-DD where CC is the century, YY is the year, MM is the month of the year between 01 (January) and 12 (December), and DD is the day of the month between 01 and 31.

7.3 *SERVICE INFORMATION VERBS*

Each Scholnet service implements three verbs that provide information about the service to the rest of the architectural components: *Identify*, *ListVerbs* and *DescribeVerb*. These are termed Service Information verbs.

Identify returns the name of the service and other specific information about the service.

ListVerbs lists the name of the verbs defined by that service.

DescribeVerb takes as input a verb of the service and returns a description of it.

These verbs have been introduced in order to allow each service to “introduce itself” to other services that may want to use it. They are intended as a way of supporting the openness of the architecture and the reusability of the single architectural components.

8. REPOSITORY SERVICE

A Repository stores and makes available both metadata content and structural descriptions of documents. It also maintains the content of textual documents. The content of multimedia documents is stored separately by the Multimedia Storage service.

A repository is divided into sets. Each set stores documents, as defined by the DoMDL, and provides verbs to deposit documents, discover their structure, and obtain their dissemination. A number of dissemination variables affect the nature of a particular dissemination request.

8.1 *STATE*

This section describes the abstract data structures managed by the Repository Service.

Authority

A naming authority is an entity that is authorised to create new handles. Naming authorities are hierarchically organised, with periods as separators. For example, CNR, CNR.IEI, CNR.IEI.MultimediaDepartment.

Set

A set is an administrator-defined subset of the repository. Each set has one token name and a (possibly) longer description. Depending on the policy of a repository an individual document may exist in one or more sets (the determination of one or more sets in which a document is located is made in the *Submit* request).
Note that there is, in general, no way to predict in which set a document appears from its handle.
A repository may have one or more set hierarchies. For example, an administrator may decide to establish two hierarchies in a repository, one based on institutional affiliation and one based on subjects as follows:

- Institutions
    - CNR - Consiglio Nazionale delle Ricerche
    - IEI - Istituto di Elaborazione della Informazione
    - IAT - Istituto per le Applicazione Telematiche
    - FhG – Fraunhofer GesellschaftIPSI - Integrated Publication and Information Systems Institute
    - FIRST - Institute for Computer Architecture and Software Technology
- Subjects
    - SE - Software Engineering
    - ISP - Image and Signal Processing
    - CSD - Computer System Design
    - IE - Information Engineering

Note that each document belonging to the *IE* set also belongs to the *Subjects* set, i.e a child set in the hierarchy contains a subset of the documents contained in the father set.

The set hierarchies in a repository are available via the *ListSets* request.

Set specifications:
Set specifications are expressed in the following grammar in which set name is the short single token name for the set:

```
setspec := setlist
setlist := setname | setname;setlist
setname := [A-Za-z0-9-_]+
```

Example: Institutions;CNR;IEI

Where CNR is the short name for “Consiglio Nazionale delle Ricerche” and IEI is the short name for “Istituto di Elaborazione della Informazione”.

Repository

A repository is a set of documents. These documents are virtually organised in:

- partitions, each of which contains the documents created by an authority;
- containers, each of which contains the documents virtually stored in a set.

Document

The Repository Service manages documents in accordance with the DoMDL document model. This logical document model is instantiated in the protocol as reported below.
A document is named by a handle, which is a kind of URN. Unlike a URL, a handle is location independent.
A handle has two parts, a *naming authority* and a *string*. It is written with these two parts separated by a slash, for example CNR.IEI /doc1. The character set for handles used in OLP is restricted to alphanumeric characters, underscore, period, and hyphen (except for the slash separator). Case is not significant in handles.

Version

Each document may have multiple versions. Versions are linear and numbered. The first version is version 1, subsequent versions are 2, 3, etc. A specific version of a document is called ***document instance***. OLP considers the most recent version as the default document version, used to return information for a document when no version is specified.

View

Each element that belongs to the view entity is managed in the OLP protocol with a set of attributes that have the following constraints:

Attribute	Constraint	Type	Value	Note
name	Mandatory	String		Alphanumeric characters plus underscore
type	Mandatory	String	‘Metadata’	Usable when the view belongs to the range of the relation “has_metadata”
		String	‘Body’	Usable when the view belongs to the domains of the relation “has_part” or “has_manifestation”
		String	‘Reference’	Usable when the view belongs to the domain of the relation “is_an_image_of”
		String	‘Choice’	Usable when the view belongs to the domain of the relation “is_specialized_by”
		String	‘Annotation’	Usable when the view belongs to the range of the relation “has_annotation”. This relation indicatess that a document instance, or a specific view, has been annotated.
display	Optional	Phrase		Usable only if the view type is “body”, “metadata”, or “choice”.

				Used to describe briefly the content of the view.
min	Optional	Integer		Usable only if the view type is "body" or "choice". Used to specify the ordinal of the first page.
max	Optional	Integer		Usable only if the view type is "body" or "choice". Used to specify the ordinal of the last page.
ord	Optional	Integer		Used to transform a set of parts into a sequence of parts (for example, if we specify the 'Ord' attribute we can transform a sequence of chapters into a book).
ref-handle	Mandatory	Handle		Usable only if the view type is "Reference". Used to specify the handle of the related document.
ref-version	Mandatory	Integer		Usable only if the view type is "Reference". Used to specify the version of the related document.
ref-view	Mandatory	String		Usable only if the view type is "Reference". Used to specify the view name of the related document.
downloading	Mandatory	Binary	Yes/No	Used to specify if the manifestations related to the view can be downloaded by the user.
transcoding	Mandatory	Binary	Yes/No	Used to specify if the Repository can convert the manifestations related to the view from the original encoding scheme to a different one.
delivering	Mandatory	Binary	Yes/No	Used to specify if the user can distribuite the manifestation related to the view.

Manifestation

In the OLP protocol, each element that belongs to the manifestation entity has a set of attributes with the following constraints:

Attribute	Constraint	Type	Note
name	Mandatory	String	Alphanumeric characters plus underscore
content-type	Mandatory	Mime type	Specifies the content-type of the manifestation in accordance with the Mime type specification
display	Optional	String	Usable only as brief human readable description of the manifestation
uri	Optional	String	Used to specify the name of the file associated with the manifestation, or the ftp URL where the file can be downloaded, or a specific URN used by the system to identify a video/composite manifestation.

type	Optional	String	Used to specify the "inside" or "outside" place where the document is stored. If the value is "inside", the physical file associated with the manifestation is stored locally in the Repository. Otherwise if the "outside" value is specified, the physical file is not managed by the Repository, which reports only the URI needed to request it. Note that if the "outside" value is specified, the URI attribute cannot be empty.
min	Optional	Integer	Used to specify the ordinal of the first file in a multiple manifestation.
max	Optional	Integer	Used to specify the ordinal of the last file in a multiple manifestation.
multiple	Optional	Boolean	This flag indicates that the manifestation is available as a set of single files. Each file may be submitted by the user or generated automatically by the system. All the files are in the same content-type and are ordered numerically.
size	Optional	Integer	Used to specify the size of the manifestation.

The information related to the views and manifestations of a document are reported in accordance with the Structure Metadata Set (SMS). The format of this metadata is given in Appendix D by means of a Data Type Definition (DTD), named SMS DTD.
Although each view may have multiple manifestations, two different views cannot have the same content type attribute value. Therefore, a textual view of a document, for example, cannot have the following two different manifestations: postscript in "A4 format" and postscript in "Letter format".
The document model features are expressed in the protocol by means of:

- The *ListVersions* verb which displays the versions available for a document.
- The *Structure* verb which displays the logical structure (views and manifestations of a version) of a document.
- The *Manifestations* verb, which displays the content-types that can be requested from a document.
- The *Disseminate* verb which allows a client to request *disseminations* (manifestation of the content in the requested digital object) by specifying a particular version, structural component, and content-type.

The syntax used to request the structure of a document instance is given in the documentation for the *Structure* verb and the syntax for requesting disseminations based on that structure is given in the *Disseminate* verb.

8.2 *REPOSITORY VERBS*

This section presents the set of the repository verbs. This set is divided into three sub-sets: Service Information Verb, Cognitive Verb, and Handle Specific Verb as reported in the following table:

Service Information Verb	**Cognitive Verb**	**Handle Specific Verb**
Identify	**ListAuthorities**	**ListVersions**
ListVerbs	**ListSets**	**Terms**
DescribeVerb	**ListBinders**	**Structure**
	ListEncodings	**Manifestations**
	ListMetaFormats	**Disseminate**

	ListSubmissionFormats	**Submit**
	ListMDFTransformations	**NewVersion**
	ListContents	**Withdraw**
	UpdateRepository	**Annotate**
	LoadHandlesDbs	
	BuildMetadataFormats	

Identify

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name and other information about the service.

Example Request:
/OLP/Repository/1.0/Identify

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<Identify version="1.0">
  <serviceName>Experimental OLP Repository Server</serviceName >
  <olp_base_url>http://labserv.iei.pi.cnr.it:8119</olp_base_url>
  <metaServiceURL>http://labserv.iei.pi.cnr.it:8221</metaServiceURL>
  <textualDescription></textualDescription>
  <protocolVersion>1.0</protocolVersion>
  <adminEmail>pagano@iei.pi.cnr.it</adminEmail>
  <verbsInfo>
    <verbsSupported>
        <description>   </description>
        <olp_base_url>http://labserv.iei.pi.cnr.it:8119/OLP/Repository/1.0/ListVerbs</olp_base_url>
    </verbsSupported>
    <verbsDescription>
        <description>   </description>
        <olp_base_url>http://…:8119/OLP/Repository/1.0/DescribeVerb/DescribeVerb</olp_base_url>
    </verbsDescription>
  </verbsInfo>
  <submissionProcedure>
    <description>Submissions accepted from Computer Science Department affiliated researchers
    </description>
      <olp_base_url>http://…:8119/OLP/LibMgt/1.0/UserSubmitForm</olp_base_url>
  </submissionProcedure>
  <harvestInformation>
    <description>
     Return a structured list of the bibliographic metadata for documents stored in this repository service
     </description>
     <olp_base_url>http://labserv.iei.pi.cnr.it:8119/OLP/Repository/1.0/ListContents</olp_base_url>
  </harvestInformation>
  <useRestrictions>Terms and conditions are provided with each document.</useRestrictions>
  <contentInfo>
    <contentDescription>
     a human readable description of the content stored in the repository
    </contentDescription>
    <authorities>
     <description>
       A naming authority is an entity that is authorised to create new handles. Naming authorities are
       hierarchically organised, with periods used as the separator
     </description>
     <olp_base_url>http://labserv….:8119/OLP/Repository/1.0/ListAuthorities</olp_base_url>
    </authorities>
```

```
        <sets>
          <description>
            A set is an administrator-defined subset of the repository.  Each set has one token name and a
            (possibly) longer description.  Depending on the policy of a repository, an individual document may
            exist in one or more sets
          </description>
          <olp_base_url>http://labserv.iei.pi.cnr.it:8119/OLP/Repository/1.0/ListSets</olp_base_url>
        </sets>
        <metadataFormats>
          <description>
            A metadata description is associated with every document in accordance with the OpenDLib
            Application Profile (OLAP)
          </description>
          <olp_base_url>http://labserv.iei.pi.cnr.it:8119/OLP/Repository/1.0/ListMetaFormats</olp_base_url>
          <olp_dtd_url>http://labserv.iei.pi.cnr.it:8119/OLP/htdocs/DTD/olap.dtd</olp_dtd_url>
        </metadataFormats>
        <documentStructure>
          <description>
            The Repository Service manages documents in accordance with the DoMDL document model.
            This logical document model is instantiated in the protocol as reported in the sms dtd .
          </description>
          <olp_base_url>http://labserv.iei.pi.cnr.it:8119/OLP/htdocs/domdl.html</olp_base_url>
          <olp_dtd_url>http://labserv.iei.pi.cnr.it:8119/OLP/htdocs/DTD/sms.dtd</olp_dtd_url>
        </documentStructure>
    </contentInfo>
</Identify>
```

ListVerbs

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name of the verbs defined by this service.

Example Request:
/OLP/Repository/1.0/ListVerbs

Example Response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ListVerbs version="1.0">
     <verb>BuildFormats</verb>
     <verb>DescribeVerb</verb>
     <verb>Disseminate</verb>
     <verb>Identify</verb>
     <verb>List2Archive</verb>
     <verb>ListAuthorities</verb>
     <verb>ListBinders</verb>
     <verb>ListContents</verb>
     <verb>ListEncodings</verb>
     <verb>ListMDFTransformations</verb>
     <verb>ListMetaFormats</verb>
     <verb>ListSets</verb>
     <verb>ListSubmissionFormats</verb>
     <verb>ListVerbs</verb>
     <verb>ListVersions</verb>
     <verb>Manifestations</verb>
     <verb>NewVersion</verb>
     <verb>Structure</verb>
     <verb>Submit</verb>
```

```
    <verb>Terms</verb>
    <verb>UpdateRepository</verb>
    <verb>Withdraw</verb>
</ListVerbs>
```

DescribeVerb

```
Version: 1.0
Fixed args: verb
Optional Args: version
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on a version of the specified verb that is supported by this service. The following information may be provided at the verb or version level:

description, description of the verb or a specific version
note, information pertaining to the verb or a specific version

Each element of the list contains the following information:

- *version,* number of the verb.
- *arguments*, a list of the names of the fixed and optional arguments, if any, accepted by the verb in that version.
- *example* template of request to this repository, with fixed arguments indicated in brackets
- *returns*, optional, contains information about response format.

Note that a service may implement more than one version of a verb.

Example Request:
/OLP/Repository/1.0/DescribeVerb/Manifestations

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <DescribeVerb version="1.0">
    <verb name="Manifestations">
      <description>
            Returns a structured response indicating the formats of disseminations available for this
            document
      </description>
      <versions>
        <version id="1.0">
              <example>http://../OLP/Repository/1.0/Manifestations/<handle></example>
          <arguments>
            <fixed>
              <arg name="handle" />
            </fixed>
            <optional>
              <arg name="version" />
              <arg name="view" />
            </optional>
          </arguments>
        </version>
        <version id="2.0">
          <note>Deprecated</note>
          <example>http://../OLP/Repository/2.0/Manifestations/<handle></example>
          <arguments>
            <fixed>
              <arg name="handle" />
            </fixed>
          </arguments>
        </version>
      </versions>
    </verb>
  </DescribeVerb>
```

ListAuthorities

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured list of the authorities stored in this repository. For each authority the list of sets that can store its documents is indicated.

Example Request:
/OLP/Repository/1.0/ListAuthorities

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
 <ListAuthorities version="1.0">
   <authority name="scholnet.one" display="Scholnet Project Repository">
      <allowed-sets>
         <set name="cs" />
         <set name="cs;trs" />
         <set name="test" />
      </allowed-sets>
   </authority>
   <authority name="scholnet.two" display="Scholnet Project Test Repository">
      <allowed-sets>
         <set name="test" />
      </allowed-sets>
   </authority>
 </ListAuthorities>
```

ListSets

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured list of the administrator-determined sets for documents stored in this repository service. The list contains the hierarchy of sets and sub-sets. For each set, both the short name and long description is returned. For each set the list of authorities that are authorised to publish in it is also indicated.

Example Request:
/OLP/Repository/2.0/ListSets

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
 <ListSets version="1.0">
   <set name="cs" display="Computer Science">
     <allowed-authorities>
       <authority name="scholnet.one" />
     </allowed-authorities>
     <set name="trs" display="Technical Reports">
       <allowed-authorities>
         <authority name="scholnet.one" />
```

```
        </allowed-authorities>
      </set>
    </set>
    <set name="test" display="Computer Science: Test area">
      <allowed-authorities>
        <authority name="scholnet.one" />
        <authority name="scholnet.two" />
      </allowed-authorities>
    </set>
  </ListSets>
```

ListBinders

```
     Version: 1.0
     Fixed args: none
     Optional Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400, 501
```

Returns a structured list of the types of binders (e.g. tar) available in this repository.

Example Request:
/OLP/Repository/1.0/ListBinders

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <ListBinders version="1.0">
    <binder>tar</binder>
  </ListBinders>
```

ListEncodings

```
     Version: 1.0
     Fixed args: none
     Optional Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400, 501
```

Returns a structured list of the types of encoding (e.g zip) available in this repository.

Example Request:
/OLP/Repository/1.0/ListEncodings

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <ListEncodings version="1.0">
    <encoding>gzip</encoding>
  </ListEncodings>
```

ListMetaFormats

```
     Version: 1.0
     Fixed args: none
     Optional Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400, 501
```

Returns a structured response indicating the metadata formats that are supported by this repository service. Note that the fact that a metadata format is supported does not mean that it is available for all documents in that repository. For each metadata format, the following information is returned:

- the name, which is the identifier for the metadata format that can be used in other OLP protocol requests;

- the namespace ID, which is a URL that refers to a document describing the metadata format.

Example Request:
/OLP/Repository/1.0/ListMetaFormats

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<ListMetaFormats version="1.0">
    <meta-format name="olms" dtd="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/Dtd/olap.dtd">
        <namespace name="dc" uri="http://purl.org/dc/elements/1.1/"/>
        <namespace name="olms" uri="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"/>
        <namespace name="dcq" uri="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"/>
    </meta-format>
    <meta-format name="bib">
        <namespace name="bibns" uri=" ftp://nic.merit.edu/document/rfc/rfc1807.txt"/>
    </meta-format>
    <meta-format name="rfc1807">
        <namespace name="rfc1807" uri="ftp://nic.merit.edu/document/rfc/rfc1807.txt"/>
    </meta-format>
</ListMetaFormats>
```

ListSubmissionFormats

```
Version: 1.0
Fixed args:
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured list of the MIME types of content streams that are accepted by this server in the Submit verb. For each manifestation format, the following information is returned:

- the name, which is the identifier for the manifestation format that has to be used in other OLP protocol requests.
- the content-type identifier as specified in RFC 1521 (Mechanisms for Specifying and Describing the Format of Internet Message Bodies) and 1522 (Message Header Extensions for Non-ASCII Text)

Example Request:
/OLP/Repository/1.0/ListSubmissionFormats

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
  <ListSubmissionFormats version="1.0">
    <manifestation name=”postscript” content-type=”application/postscript” />
    <manifestation name=”xml” content-type=” text/xml ” />
    <manifestation name=”html” content-type=” text/html ” />
  </ListSubmissionFormats>
```

ListMDFTransformations

```
Version: 1.0
Fixed args:
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response indicating the metadata formats that are acceptable in the submission phase (source metadata format) and the available transformations to other formats (destination metadata format) that are supported by this repository service instance. Note that the user, at submission time, can refuse the transcoding of his/her bibliographic record into other formats. For

this reason the fact that a metadata format is supported does not mean that it is available for all documents in that repository.
For each source metadata format, the list of available destination metadata formats is reported.

Example Request:
/OLP/Repository/1.0/ ListMDFTransformations

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<ListMDFTransformations version="1.0">
  <source value="bib">
    <destination value="rfc1807" />
    <destination value="olms" />
    <destination value="dc" />
  </source>
  <source value="olms">
    <destination value="dc" />
    <destination value="rfc1807" />
  </source>
</ListMDFTransformations>
```

ListContents

```
Version: 1.0
Fixed args: none
Optional Args: authorities, partitionspec, file-after, file-before, meta-
format
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured list of the handles for documents stored in this repository service. Without any arguments the list includes all stored documents.
The meaning of the Optional Arguments is as follows:

- authorities limits the returned handles to those specified in the authorities list. The authorities available for a repository are returned from the ListAuthorities request.
- partitionspec limits the returned handles to those in the specified set specification. The sets available for a repository are returned from the ListSets request.
- file-after limits the list to those handles for documents that were added or modified since time, a universal time expressed in ISO 8601 format. If the server is not able to determine date of modification to the resolution of a day, or if the server is not able to selectively extract records on a time scale of a day, the server may return additional records, e.g. all those modified during the week, month, or even century containing the date.
- file-before limits the list to those handles for documents that were added or modified prior to time, a universal time expressed in ISO 8601 format. If the server is not able to determine date of modification to the resolution of a day, or if the server is not able to selectively extract records on a time scale of a day, the server may return additional records, e.g. all those modified during the week, month, or even century containing the date.
- meta-format returns, in addition to the handle, metadata for each document in the specified format. The metadata is empty for any document that does not have metadata in that format. Legal values for this argument are one of the metadata formats (as returned from the ListMetaFormats request).

Example Request: List the handles in the Technical Reports (trs) set within the Computer Science (cs) set.
/OLP/Repository/1.0/ListContents?partitionspec=cs;trs

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<ListContents count="8" version="1.0">
```

```
    <document handle="ncstrl.scholnet.one.trs.good/2001-TR-009" version="1"/>
    <document handle="ncstrl.scholnet.one.trs/2001-TR-005" version="1"/>
    <document handle="ncstrl.scholnet.one.trs/2001-TR-021" version="1"/>
    <document handle="ncstrl.scholnet.one/2001-TR-002" version="1"/>
    <document handle="ncstrl.scholnet.one/2001-TR-142" version="1"/>
    <document handle="ncstrl.scholnet.one/2001-TR-144" version="1"/>
    <document handle="ncstrl.scholnet.two.bad/2001-TR-001" version="1"/>
    <document handle="ncstrl.scholnet.two/Test-001" version="1"/>
</ListContents>
```

List the OLMS metadata along with the handles

Example Request:
/OLP/Repository/1.0/ListContents?partiotionspec=cs;trs&meta-format=olms

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListContents count="8" version="1.0">
    <document handle="ncstrl.scholnet.one.trs.good/2001-TR-009" version="1">
        <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
            xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
            xmlns:dc="http://purl.org/dc/elements/1.1/">
            <dc:title>A document test</dc:title>
            <dc:creator>Pagano, Pasquale</dc:creator>
            <dcq:description.abstract>This document has been inserted to test the repository functionality
            </dcq:description.abstract>
            <dcq:date.available>February 12, 2001</dcq:date.available>
            <dcq:date.issued>February 12, 2001</dcq:date.issued>
        </olms:ol>
    </document>

……
</ListContents>
```

-- --

```
     Version: 1.1
     Fixed args: none
     Optional Args: authorities, partitionspec, file-after, file-before, meta-
     format, resumption-token
     Return MIME type: text/xml
     Return Status Codes: 200, 400, 501
```

Returns a structured list of the handles for documents stored in this repository service. Without any arguments the list includes only N stored documents. This is because the lists may be very large and it may be practical to partition them among a series of requests and responses as follows:

- A repository replies to a ListContents 1.1 request with a list and a resumption-token (means that the list is incomplete), or with a list without resumption-token (means that no other informations need to be added to the other results);
- In order to complete the response list, the client of the request will have to send one or more requests with resumption-token as arguments. The complete list then consists of the concatenation of the incomplete lists from the sequence of requests.

The meaning of the Optional Arguments is as follows:

- authorities limits the returned handles to those specified in the authorities list. The authorities available for a repository are returned from the ListAuthorities request.
- partitionspec limits the returned handles to those in the specified set specification. The sets available for a repository are returned from the ListSets request.
- file-after limits the list to those handles for documents that were added or modified since time, a universal time expressed in ISO 8601 format. If the server is not able to determine date of

modification to the resolution of a day, or if the server is not able to selectively extract records on a time scale of a day, the server may return additional records, e.g. all those modified during the week, month, or even century containing the date.

- file-before limits the list to those handles for documents that were added or modified prior to time, a universal time expressed in ISO 8601 format. If the server is not able to determine date of modification to the resolution of a day, or if the server is not able to selectively extract records on a time scale of a day, the server may return additional records, e.g. all those modified during the week, month, or even century containing the date.
- meta-format returns, in addition to the handle, metadata for each document in the specified format. The metadata is empty for any document that does not have metadata in that format. Legal values for this argument are one of the metadata formats (as returned from the ListMetaFormats request).
- resumption-token is used to partition the results. Note that when a resumption-token is specified all other parameters will be ignored.

ListVersions

```
Version: 1.0
Fixed args: handle
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 404, 501
```

Returns a structured response describing the current versions (*document instances*) available for the requested handle. The information returned consists of:

- id: the version number, a positive integer
- date: the submission date, expressed as in ISO 8601, or * if this cannot be determined
- comment: a comment or description. A single line of text only.

Example Request:
/OLP/Repository/1.0/ListVersions/handlecorp%2fTR010101

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <ListVersions version="1.0">
    <version id="2">
      <date>1999-02-01</date>
      <comment>this version</comment>
    </version>
    <version id="1">
      <date>1998-02-01</date>
      <comment>that version</comment>
    </version>
  </ListVersions>
```

Terms

```
Version: 4.0
Fixed args: handle
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 404, 501
```

Returns a natural language statement of the terms and conditions for use of the object. The intention is not to create a contract, but only to advise a human of the reasonable expectations of the server or author.

Example Request:
/OLP/Repository/4.0/Terms/CNR.IEI/970101

Example Response:
```
<Terms version="4.0">
    <text>Copyright 1999 by the CNR-IEI, Pisa Italy
    </text>
</Terms>
```

Structure

```
Version: 1.0
Fixed args: handle
Optional Args: version, view
Return MIME type: text/xml
Return Status Codes: 200, 400, 404, , 501, 503
```

This verb returns a structured response that describes the decompositions available for a document instance. A client may use the decomposition information as the basis for document requests using the *Disseminate* verb.
The meaning of the Optional Arguments is as follows:

- *version* specifies the document instance for which the structural information is requested. If omitted, it provides information about the latest document instance.
- *view* specifies the view of the document instance for which structural information is requested. If omitted, it provides information about the entire document instance.

The response lists the views, and manifestations available for a document instance.

Example Request:
/OLP/Repository/1.0/Structure/scholnet.test/2002-test-003?version=1.0

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<document handle="scholnet.test/2002-test-003" version="1">
  <view name="Annotation" type="annotation" display="Annotation posted to the document">
      <manifestation content-type="text/xml" name="ref_annotation" type="outside" uri="Annotation_0006" />
      <view name="Ann_2002-test-003-6433214" type="annotation" display="Annotation">
          <manifestation content-type="text/xml" name="ref_annotation" type="outside" uri="003-6433214" />
      </view>
      <view name="Ann_2002-test-003-5646225" type="annotation" display="Annotation">
          <manifestation content-type="text/xml" name="ref_annotation" type="outside" uri="003-5646225" />
      </view>
  </view>
  <view name="bibdata" type="metadata" display="Bibliographic record">
      <manifestation content-type="text/xml" name="olms" />
      <manifestation content-type="text/xml" name="dc" />
 </view>
  <view min="1" max="5" name="Part_1" type="body" delivering="yes" downloading="yes" transcoding="yes"
display="The Scholnet architecture">
      <manifestation content-type="application/postscript" name="postscript" />
      <manifestation content-type="application/pdf" name="pdf" />
      <view name="Annotation" type="annotation" display="Annotation posted to a view of the document">
          <manifestation content-type="text/xml" name="ref_annotation" type="outside" uri="Annotation_06" />
      </view>
 </view>
</document>
</Structure>
```

This response says that the document instance can disseminate two metadata formats at the top level, DC and OLMS. The document instance also has one body view called “Part_1”. This view has two manifestations (postscript and pdf).

Example Request:
/OLP/Repository/1.0/Structure/handlecorp/TR010101

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE structure SYSTEM "http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/Dtd/olap.dtd">
<Structure version="1.0">
    <document handle=" handlecorp/TR010101" version="1.0">
        <view name="bibdata" type="metadata" downloading=”yes” transcoding="yes” delivering=”yes”>
            <manifestation name="olms" content-type="text/xml"/>
            <manifestation name="dc" content-type="text/xml"/>
        </view>
        <view name="book" type="body" downloading=”yes” transcoding="yes” delivering=”yes”>
            <manifestation name="gif" min="1" max="150"  multiple=”true” content-type="image/gif"/>
            <manifestation name="postscript-pageimage" min="1" max="150" multiple=”true” content-
type="application/postscript"/>
            <manifestation name="postscript" content-type="application/postscript"/>
        </view>
    </document>
</Structure>
```

This is the "shorthand" response for documents with simple structure. The response indicates one two metadata types for the document instance, DC and OLMS, and a single body view, book, with a set of sequential pages numbered from 1 to 150 inclusive, available in gif and postscript formats. The body view is also available in postscript format.

Example Request:
/OLP/Repository/1.0/Structure/handlecorp/TR010101?view=book

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE structure SYSTEM "http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/Dtd/olap.dtd">
<Structure version="1.0">
    <document handle=" handlecorp/TR010101" version="1.0">
        <view name="book" type="body" downloading=”yes” transcoding="yes” delivering=”yes”>
            <manifestation name="gif" min="1" max="150"  multiple=”true” content-type="image/gif"/>
            <manifestation name="postscript-pageimage" min="1" max="150" multiple=”true” content-
type="application/postscript"/>
            <manifestation name="postscript" content-type="application/postscript"/>
        </view>
    </document>
</Structure>
```

This request, and the associated response, demonstrate the use of the view argument in the request, which specifies that only structure information for the specified view should be returned.

Example Request:
/OLP/Repository/1.0/Structure/handlecorp/TR010101?view=DC
This request specifies that only the metadata formats for the document should be returned.

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?
<!DOCTYPE structure SYSTEM "http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/Dtd/olap.dtd">
<Structure version="1.0">
    <document handle=" handlecorp/TR010101" version="1.0">
        <view name="DC" type="metadata" downloading=”yes” transcoding="yes” delivering=”yes”>
            <manifestation name="dc" content-type="text/xml"/>
            <manifestation name="dc_html" content-type="text/html"/>
        </view>
    </document>
```

```
</Structure>
```

This response says that the DC metadata can be disseminated in html or in xml format.

Example Request:
/OLP/Repository/1.0/Structure/handlecorp/2001-TR-125?version=1

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE structure SYSTEM "http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/Dtd/olap.dtd">
  <Structure version="1.0">
    <document handle=" handlecorp/2001-TR-125" version="1.0">
      <view name="bibdata" type="metadata" downloading="yes" transcoding="no" delivering="yes">
          <manifestation name=" rfc1807"  content-type="text/xml" />
          <manifestation name=" dc"  content-type="text/xml" />
      </view>
      <view name="journal" type="body" downloading="yes" transcoding="yes" delivering="yes">
          <view name="part-1" type="body" ord="1" downloading="yes" transcoding="no" delivering="no">
              <view name="art-1" type="reference"
                          ref-handle="handlecorp/doc_id"
                          ref-version="version_number"
                          ref-view="name_of_view"
              </view>
              <view name="art-2" type="reference"
                          ref-handle="handlecorp/doc_id"
                          ref-version="version_number"
                          ref-view="name_of_view"
              </view>
              …………………
          </view>
          <view name="part-2" type="body" ord="2"  downloading="yes" transcoding="no" delivering="no">
              <view name="art-1" type="reference"
                          ref-handle="handlecorp/doc_id"
                          ref-version="version_number"
                          ref-view="name_of_view"
              </view>
               …………………
          </view>
      </view>
    </document>
  </Structure>
```

This response says that the document instance can disseminate two metadata formats at the top level, RFC1807 and DC. The document instance also has one body view called journal. This view is structured into body views named part-xx. Each part contains different reference views: each is a link to a specific view of a document instance of an article.

Example Request:
/OLP/Repository/1.0/Structure/handlecorp/2001-TR-125?version=1

Example Response:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE structure SYSTEM "http://labserv.iei.pi.cnr.it/olp/htdocs/olms/dtd/olap.dtd">
<Structure version="1.0">
  <document handle=" handlecorp/2001-TR-125" version="1.0">
    <view name="bibrecord" type="metadata" display="bib rfc1807 textual record" downloading="yes"
transcoding="no" delivering="yes">
        <manifestation name="bib" content-type="text/x-rfc1807" />
    </view>
    <view name="body" type="choice" display="the document itself" min="1" max="150" downloading="yes"
transcoding="yes" delivering="yes">
```

```
        <view name="body-english" type="body" display="the english version" downloading="yes"
transcoding="yes" delivering="yes">
            <manifestation name="postscript" content-type="application/postscript" />
            <manifestation name="pdf" content-type="application/pdf" />
            <manifestation name="postscript-pageimage" content-type="application/postscript" multiple="true"
                              min="1" max="150" />
            <manifestation name="inline" content-type="image/gif" min="1" max="150" multiple="true"/>
        </view>
        <view name="body-italian" type="body" display="the italian version" downloading="yes"
transcoding="yes" delivering="yes">
            <view name="part1" display="..." type="body" min="1" max="50" ord="1" downloading="yes"
transcoding="yes" delivering="yes">
                <manifestation name="postscript" content-type="application/postscript"/>
                <manifestation name="postscript-pageimage" content-type="application/postscript"
                              multiple="true" min="1" max="50"/>
                <manifestation name="inline" content-type="image/gif" multiple="true" min="1" max="50"/>
                <view name="chapter1" type="body" min="1" max="14" paged="true" ord="1"
downloading="yes" transcoding="yes" delivering="yes">
                    <manifestation name="postscript" content-type="application/postscript" />
                    <manifestation name="postscript-pageimage" content-type="application/postscript"
                              multiple="true" min="1" max="14"/>
                    <manifestation name="inline" content-type="image/gif" multiple="true" min="1" max="14"/>
                </view>
                <view name="chapter2" type="body" min="15" max="50" ord="2" downloading="yes"
transcoding="yes" delivering="yes">
                    <manifestation name="postscript" content-type="application/postscript" />
                    <manifestation name="postscript-pageimage" content-type="application/postscript"
                              multiple="true" min="15" max="50"/>
                    <manifestation name="inline" content-type="image/gif" multiple="true" min="15" max="50"/>
                </view>
            </view>
            <view name="part2" display="..." type="body" min="51" max="150" ord="2" downloading="yes"
transcoding="yes" delivering="yes">
                ……………………..
            </view>
        </view>
    </view>
  </document>
</Structure>
```

This response says that the user may view the document instance through the metadata, or through two different structured versions. The first one in English through the "postscript"/"pdf" manifestations, or as sets of pages. The second one allows the document to be perceived as two structured parts, each of which has multiple chapters (available in postscript format or as a set of pages).

Manifestations

```
Version: 4.0
Fixed args: handle
Optional Args: version, view
Return MIME type: text/xml
Return Status Codes: 200, 400, 404, 501, 503
```

Returns a structured response indicating the dissemination formats available for this document instance.
The available arguments are:

- *version* specifies the document instance for which the format information is requested. If omitted, it provides information about the latest document instance.

- *view* specifies the view of the document instance for which format information is requested. If omitted, it provides information about the entire document instance.

The response includes a list of manifestations with the following information for each one:

- *name*, is the string that can be used in a *Disseminate* request.
- *content-type*, is the corresponding MIME type of this manifestation.
- *size*, is an estimate of the number of bytes of the dissemination, expressed in that content-type. This is returned only if the size can be determined.
- *multiple*, identifies manifestations available as a set of single numbered files.
- *min,* identifies the ordinal of the first file
- *max,* identifies the ordinal of the last file

```
Example Request:
/OLP/Repository/1.0/Manifestations/CNR.IEI/970101?version=1&view=chapt1
```

```
Example Response:
<?xml version="1.0" encoding="UTF-8"?>
   <Manifestations version="1.0">
      <document handle=”CNR.IEI/970101” version=”1.0”>
        <manifestation name=”postscript” content-type=”application/postscript” size="25555" />
       <manifestation name=”xml” content-type=”text/xml” size="3258" />
      </document>
   </Manifestations>
```

Disseminate

```
Version: 1.0
Fixed args: handle, view, manifestation
Optional Args: version, binder, encoding, items
Return MIME type: dependent on dissemination requested.
Return Status Codes: 200, 400, 404, 415, 501, 503
```

Requests a dissemination of a digital object. The characteristics of the dissemination that can be requested (the arguments of the *Disseminate* verb), are determined by the responses to the *ListVersions* and *Structure* verbs (or from the *Formats* verb) for the specific digital object. The response is a MIME-typed byte stream. The MIME type of the byte stream is either the MIME type of the specified content-type or, if a binder is specified, the MIME type of that binder. Refer to the examples below for more information on the stream that is returned.

In addition to the handle, the required fixed arguments are:

- *view* specifies the view of the document instance from which a dissemination is requested. This argument is one of the available body or metadata views for the document instance. For example: book;chapter-1 specifies the chapter1 part of the book view.
- *manifestation* is a required argument that specifies the MIME-type tag of the content in the dissemination. The list of available content-types for a document instance is indicated by the response to the *Manifestations* or *Structure* request. Note that the value supplied for the argument is not the actual MIME-type, but the manifestation name as indicated in the response to the *Structure* request or the tag name as indicated in the response to the *Manifestations* request (because of the complications with the "/" character in HTTP requests). If there is no binder argument, then the manifestation is the MIME-type of the stream disseminated to the client.

The Optional Arguments are:

- *version* specifies the document instance from which a dissemination is requested. If omitted, the latest document instance is used by default.
- *binder* specifies the encapsulating binder for the dissemination. The MIME type of the stream disseminated to the requesting client is then the output of the binder application. A

binder argument is required if the dissemination contains multiple logical objects (e.g., multiple images in image/tiff). The list of Binders supported by a repository is available through the *ListBinders* protocol request.

- *encoding* specifies the compression encoding scheme that should be applied to the dissemination. The encoding applied is reflected in the HTTP Content-encoding header returned from the *Disseminate* request. The list of encodings supported by a repository is available through the *ListEncodings* protocol request.
- *items* specifies the ordinal, or the interval, of the requested file, files, in accordance with the following syntax:
  items=xx[-yy] where xx and yy are the numbers identifying an item.
  If the *items* argument for a multiple manifestation is not specified, the binder argument become mandatory.
  If the *items* argument for a non-multiple manifestation is specified, the argument is ignored.

Note that if a Disseminate request specifies an outside manifestation (a manifestation that has set the *type* attribute to the *outside* value) the 404 error is generated. The Repository only manages manifestations that are stored locally (inside manifestations). Other services can access the outside manifestations directly through the address specified by the attribute (if it is a URL) or through the Multimedia Storage Service (if the value is a URN).
Note also that the Repository disseminates only views, through their manifestations, that have set the attribute "downloading" to "yes". If the value for this attribute is set to "no" the 401 error code is generated.

Example Request:
OLP/Repository/1.0/Disseminate/handlecorp/TR010101/book/postscript

Example Response:
The PostScript rendition (MIME-type postscript) of the "book" view of the document.

Example Request:
OLP/Repository/1.0/Disseminate/handlecorp/TR010101/book;gif/gif(3)

Example Response:
Page 3 of the book view of the document instance as a GIF file (MIME type image/gif).

Example Request:
OLP/Repository/1.0/Disseminate/handlecorp/TR010101/book;chapter3/gif
?binder=tar&encoding=gzip

Example Response:
Chapter3 of the book view of the document instance as a series of GIF files, bound together using tar, and then compressed using gzip. The MIME type of the disseminated byte stream is than compressed/gzip.

Submit

```
Version: 1.0
Fixed args: none
Fixed_post args: file
Optional Args: id, partitionspec, authority
Return MIME type: text/xml
Return Status Codes: 200, 400, 401, 501
Note: authorized client only
```

Submit a new document to be stored in the repository. This is transmitted as an HTTP POST request where the input stream is a MIME type multipart/mixed. This input stream has multiple parts ordered as follows:

- a structure file with MIME type text/xml;

- a metadata file with MIME type text/xml;
- one or more files with the appropriate MIME types each of which is a manifestation in a format that this repository can accept as an input format. Refer to the *Submit-Formats* verb for how to retrieve the input formats acceptable to a repository.

The meaning of the arguments is as follows:

- id is a *suggested* doc_id for this document. The repository may reject the submission because the handle is not unique, or does not conform to the naming conventions for the repository, or if the repository service does not allow client-specified names. If the id argument is omitted then the name of the repository service will be assigned, if allowed.
- partitionspec is a list of sets identifying the sets in the repository in which the document should be deposited. The ListSets verb provides information on the available sets in the repository. This argument is *required* if the repository has multiple sets.
- authority is a *required* argument if the Repository manages multiple authorities. It indicates the naming authority to be associated with the document. The repository must be configured to support the authority specified for this document.

Example of the structure file passed to the Repository:

```
<?xml version="1.0" encoding="UTF-8"?>
<Structure version="1.0">
<view name="bibrecord" type="metadata" display="OpenDLib Metadata Record" downloading=”yes”
transcoding=”yes” delivering=”yes”>
        <manifestation name="dc" content-type="text/xml" filename="metadata.xml" />
    </view>
    <view name="body" type="body" display="The document itself" min="1" max="150" downloading=”yes”
transcoding=”yes” delivering=”yes”>
        <manifestation name="postscript" content-type="application/postscript" filename="body.ps" />
    <view name="part1" display="Part I: Engineering of the OLP protocol" type="body" min="1" max="50"
downloading=”yes” transcoding=”yes” delivering=”yes”>
        <view name="chapter1" type="body" min="1" max="14" downloading=”yes” transcoding=”yes”
delivering=”yes”/>
        <view name="chapter2" type="body" min="15" max="50" downloading=”yes” transcoding=”yes”
delivering=”yes”/>
    </view>
    <view name="part2" type="body" display="Part II: Implementing the Engineered protocol" min="50"
max="150" downloading=”yes” transcoding=”yes” delivering=”yes”/>
        <view name="chapter1" type="body" min="51" max="64" downloading=”yes” transcoding=”yes”
delivering=”yes”/>
        <view name="chapter2" type="body" min="64" max="75" downloading=”yes” transcoding=”yes”
delivering=”yes”/>
        <view name="chapter3" type="body" min="76" max="97" downloading=”yes” transcoding=”yes”
delivering=”yes”/>
        <view name="chapter4" type="body" min="97" max="150" downloading=”yes” transcoding=”yes”
delivering=”yes”/>
    </view>
</Structure>
```

The above structure file indicates to the Repository Service how to use the MIME multi-part file and how to generate the parts of the documents. The user submits a single postscript file and provides some information on the structure of the document. The repository automatically generates all other manifestations according to the information given.

If the submit request is successful the structured return contains:

- handle of the submitted document.
- partiotionspec indicating the set specification for each of the sets to which the document was submitted.

***Note on Access Control*:** the repository service accepts Submit requests only from recognized LibMgt Service authorized to handle the specified authority.

Example Request:
/OLP/Repository/1.0/Submit?authority=CNR.IEI&id=TR010101&partitionspec=compsci;ai,eng

In addition an input stream should be supplied with the POST that includes this request.

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
  <Submit version="1.0">
    <handle>CNR.IEI/TR010101</handle>
    <set name="compsci">
      <display> Computer Science </display>
      <set name="ai">
        <display> Artificial Intelligence </display>
      </set>
    </set>
    <set name="eng">
      <display> Engineering Department </display>
    </set>
  </Submit>
```

NewVersion

```
Version: 1.0
Fixed args: handle
Fixed_post args: file
Optional Args: comment
Return MIME type: text/xml
Return Status Codes: 200, 400, 401, 404, 501
Note: authorized client only
```

Submit a new version of a document with an identifier handle currently stored in the repository. The version number of the new version is one greater than the previous most current version (and becomes the default version for *Disseminate* and *Structure* requests made without version arguments).

***Note on Access Control*:** the repository service accepts Submit requests only from recognized LibMgt Service authorized to handle the specified authority. In this case, it is the client's responsibility to verify that the party requesting a new version is authorised to do so.

This is transmitted as an HTTP POST request where the input stream is a MIME type multipart/mixed. This input stream has two parts ordered as follows:

- a metadata file with MIME type text/xml;
- a structure file with MIME type text/xml;
- one or more files with the appropriate MIME types, each of which is a manifestation in a format that this repository is able to accept as an input format. Refer to the *Submit-Formats* verb for how to retrieve the input formats acceptable to a repository.

The meaning of the arguments is as follows:

- *comment* - a free text description of the new version.

If the new version request is successful the structured response contains:

- *version* number of the latest version;
- *date* of the submitted version;
- *comment* for this version.

Example Request:
/OLP/Repository/1.0/NewVersion?description=fixed+wrong+algorithm
In addition an input stream should be supplied with the POST that includes this request.

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
   <NewVersion version="1.0">
      <version id="2">
         <date>1999-02-01></date>
         <comment>fixed wrong algorithm</comment>
      </version>
   </NewVersion>
```

Withdraw

```
Version: 1.0
Fixed args: handle
Optional Args: version, reason, nosave, delete
Return MIME type: text/xml
Return Status Codes: 200, 400, 401, 404, 501
Note: authorized client only
```

Removes a document from a repository. The following arguments are available:

- version allows to specify which version are to be withdrawed. If no version is psecified, the latest one is assumed.
- *reason* is a text string specifying the reason for withdrawal. A common use of this string is to state a new access point for the document.
- *nosave* is a suggestion to the repository that the deleted content should not be archived (e.g., maintained by the repository but not accessible through protocol).
- *delete* specifies that the information in the bibliographic record (metadata) for the document should be deleted. If not specified, then the metadata will remain but the content will be deleted.

Note that if a delete option is not specified, a withdraw request does not remove the document but only manifestations of the document. All information reported in the metadata and structured files is kept in the repository.
If the withdraw request is successful the structured response contains: handle of the withdrawn document.
***Note on Access Control*:** the repository service accepts Submit requests only from recognized LibMgt Service authorized to handle the specified authority. In this case, it is the client's responsibility to verify that the party requesting a withdrawal is authorised to do so.

Example Request:
/OLP/Repository/1.0/Withdraw/handlecorp/TR010101?reason=I+was+wrong.+The+world+is+not+flat

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
   <Withdraw version="1.0">
      <handle>handlecorp/TR010101</handle>
   </Withdraw>
```

Annotate

```
Version: 4.0
Fixed args: handle, op, ann_handle
Optional Args: version, view
Return MIME type: text/xml
Return Status Codes: 200, 400, 404, 501
```

Pasquale Pagano 6/21/0221 19:47
Deleted: text/plain

Allows an annotation flag to be set to/removed from to a specific document version or its view.
The meaning of the arguments is as follows:

- *handle* identifies the document;
- *ann_handle* allows to identify the annotation stored outside the Repository service;
- *version* specifies the document instance. If omitted, the Repository service assumes the latest document instance;
- *view* specifies the view of the document instance to which to link the annotation view. If omitted, the annotation is linked to the document instance;
- *op* assumes one of the two following values: 'set' or 'remove'. Specifying 'set', an annotation view is linked to the document instance (or its view) if it is not yet in the document structure information. Specifying 'remove', the annotation view is removed from the document instance.

Pasquale Pagano 6/21/0221 19:49
Formatted: Bullets and Numbering

Returns 200 if the operation concludes successfully, the appropriate error code otherwise.

Example Request:
/OLP/Repository/1.0/Annotate/handlecorp/TR010101/set/ann_001

Pasquale Pagano 6/21/0221 19:47
Deleted: /OLP/Repository/1.0/Annotate/handlecorp/TR010101/remove

Example Response:
```
 <?xml version="1.0" encoding="UTF-8" ?>
    <Annotate version="1.0">
        <document handle="scholnet.test/2002-test-002" version="1">
            <annotation_set>
                <annotation handle=" ann_001" />
            </annotation_set>
        </document>
    </Annotate>
```

Example Request:
/OLP/Repository/1.0/Annotate/handlecorp/TR010101/remove/ann_001

Example Response:
```
 <?xml version="1.0" encoding="UTF-8" ?>
    <Annotate version="1.0">
        <document handle="scholnet.test/2002-test-002" version="1">
            <annotation_remove>
                <annotation handle=" ann_001" />
            </annotation_remove>
        </document>
    </Annotate>
```

BuildMetadataFormats

```
Version: 1.0
Fixed args: none
Optional Args: handle, version, authorities
Return MIME type: none
Return Status Codes: 200, 400, 501
Note: authorized client only
```

This verb updates the repository adding any missing manifestations of a metadata view. A specif handle, with its version, or a generic list of authorities could be specified.
This verb has to be used when a set of documents are imported or when some data are corrupted or need to be upgraded.
Note that when an handle is specified the authorities list will be ignored. Note also that if a version is not specified, the latest one is assumed.

Example Request:

/OLP/Repository/1.0/BuildMetadataFormats?authorities=test test1 test2

LoadHandlesDbs

```
Version: 1.0
Fixed args: none
Optional Args: authorities, rebuild, nosave
Return MIME type: none
Return Status Codes: 200, 400, 501
Note: authorized client only
```

This verb updates the databases used to store information about documents. It can be used specifying a list of authorites.
This verb has to be used when a set of documents are imported or when some data are corrupted or need to be upgraded.
Note that if rebuild is not specified, the Repository service only checks and updates the databases. If the rebuild options is setted, then databases will be copied in a backup directory and new instances of the databases will be created. Specifying the option nosave, the backup copy will be omitted.

Example Request:
/OLP/Repository/1.0/ LoadHandlesDbs?authorities=test test1 test2

UpdateRepository

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: none
Return Status Codes: 200, 400, 501
Note: authorized client only
```

This verb updates the repository built with a generic version of Dienst in order to re-use it with the more sophisticated OpenDLib Repository.
In particular it:

- scans any old style authority based directories and sets handle_mapping_db (handle_partition and partition_handle mappings) according to virtual partitions
- can traverse partition sub-directories and set handle_mapping_db and reverse mapping accordingly
- adapts the old file-system structure to support more than 32767 documents per authority. The new limit offor publications is now associated with the document identifier sub-parts. For example, let us suppose that an authority has configured the Repository Service in order to support a document identifier composed of three parts: the year, the type of document (TR, technical report; TH, thesis; etc), and a progressive ordinal. Examples are 2000-TR-009, 2001-TH-1234, etc. The limit of publications for this authority is now 32767 raised to third (???) (more than 35000 * 10^9 documents).

The handle_mapping_db (which also manages handle_partition, partition_handle tables) stores relations between documents and sets. These relations are maintained in a database so that the Repository Service can support in an efficient way any service requests that include the set specifications.

Example Request:
/OLP/Repository/1.0/UpdateRepository

9. MULTIMEDIA DOCUMENT STORAGE AND DELIVERY SERVICE

The Multimedia Document Storage and Delivery Service (MDS) supports the storage and both the streaming (real-time) and the download delivery of the stored multimedia documents.

9.1 STATE

In this section we describe the data structure, its abstract data type, and a brief textual description for each object managed by the Multimedia Storage Service. Also, whenever necessary, a short explanation of the main ideas behind certain functions is given for the interested reader.

9.2 DEFINITIONS

Composite Documents

Definition: A composite document is a document that consists of several files whereby one of these files is the "header file" that contains references to the other files.

Non Composite Documents

Definition: A non composite document is a document with a single file representation.

Example 1: A web page which contains two pictures and some html is a composite document made of three files (two files with pictures and one html file, the html file is the "header file").
Example 2: A web page which contains only embedded information without any references to other files is a non composite document.

Multimedia Documents

The main structures the MDS deals with are:

1. Non composite documents represented as continuous files that can be downloaded and stored on a local file system or transmitted in real-time as a stream and that can be played with suitable player applications (e.g. MPEG-1, QuickTime, Real video)
2. Composite documents in SMIL format
3. Non composite documents that can be downloaded and stored on a local file system but can't be streamed (e.g. bitmaps in .bmp Format)

Note: Although the MDS' internal architecture is optimised for the handling of multimedia documents like audio and video streams in certain well known standards it is also possible to store any other kinds of documents with single file representations in the MDS' data vaults. The MDS provides only an ftp based download facility for these documents as long as none of the MDS streaming servers is able to stream them. Usage of the MDS for non multimedia files is not recommended.

Note: Any document's original version will always be available for downloading. It will also be available for streaming, if it is in a stream able format and the format is understood and can be streamed by one of the MDS streaming servers.

Uploading of documents

The remote user of this service must use the file transfer protocol (ftp) for uploading files to the vaults of the MDS.

9.3 MULTIMEDIA STORAGE VERBS

In this section, we present the set of the MDS verbs. This set is divided into four sub-sets: Service Information Verb, Service Specific Verb, Service Specific Verb (restricted) and Service Specific Verb (administrative) as reported in the following table:

Service Information Verb	Service Specific Verb	Service Specific Verb (restricted)	Service Specific Verb (administrative)
Identify	GetListOfDocuments	ApproveDocument	
ListVerbs	EndSession	WithdrawDocument	
DescribeVerb	RequestUpload		
	DiscardUpload		
	ProcessUpload		
	GetDocumentInfo		
	GetUserRole		
	UserLoginForm		
	ApproverLoginForm		
	AdminLoginForm		

Identify

```
    Version: 1.0
    Fixed args: none
    Keyword Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name and other information about the service.

Example Request:
/OLP/MDS/1.0/Identify

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<Identify version="1.0">
<serviceName>Multimedia Document Storage and Delivery Service</serviceName>
<baseURL>http://pc-snaredrum.ipsi.fhg.de:8283</baseURL>
<protocolVersion>1.0</protocolVersion>
<adminEmail>jlang@ipsi.fhg.de</adminEmail>
<descriptions />
</Identify>
```

ListVerbs

```
    Version: 1.0
    Fixed args: none
    Keyword Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

Returns a structured response containing the name of the verbs defined by this service.

Example Request:
/OLP/MMS/1.0/ListVerbs

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListVerbs version="1.0">
<verb>AdminLoginForm</verb>
<verb>ApproverLoginForm</verb>
<verb>ApproveDocument</verb>
<verb>DescribeVerb</verb>
<verb>DiscardUpload</verb>
<verb>EndSession</verb>
<verb>GetDocumentInfo</verb>
<verb>GetListOfDocuments</verb>
<verb>Identify</verb>
<verb>ListVerbs</verb>
<verb>ProcessUpload</verb>
<verb>RequestUpload</verb>
<verb>UserLoginForm</verb>
<verb>WithdrawDocument</verb>
</ListVerbs>
```

DescribeVerb

```
Version: 1.0
Fixed args: verb
Keyword Args: version
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Returns a structured response that contains a list, where each element of the list provides information of a version of the specified verb that is supported by this service. The following information may be provided at the verb or version level:

- *description*, description of the verb or a specific version
- *note*, information pertaining to the verb or a specific version

Each element of the list contains the following information:

- *version* number of the verb.
- *arguments*, a list of the names of the fixed and keyword arguments, if any, accepted by the verb in that version.
- *example* template of request to this repository, with fixed argument indicated in brackets
- *returns*, optional, contains information about response format.

Note that a service may implement more than one version of a verb.

Example Request:
/OLP/MMS/1.0/DescribeVerb/UserLoginForm

Example Response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<DescribeVerb version="1.0">
<Verb name="UserLoginForm">
  <description>Allows the user to log in the user environment. If the authentification was successful a unique session id will be given back.</description>
<versions>
<version id="1.0">
  <example>http://pc-snaredrum.ipsi.fhg.de:8283/OLP/MDS/1.0/UserLoginForm</example>
<arguments>
<optional_post>
<arg name="login" />
<arg name="password" />
```

```
</optional_post>
</arguments>
<returns note="unstructured text/html" />
</version>
</versions>
</Verb>
</DescribeVerb>
```

UserLoginForm

```
    Version: 1.0
    Post args: username, password
    Keyword Args: -
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

Before any other MDS service specific verb can be used the user has to pass the authentification procedure. By calling “**UserLoginForm**” a request for a “normal” session with ordinary user rights will be asked. If the authentification was successful a unique session id will be given back.

Note: The username/password combination must resemble a valid **Registry Service** user entry with sufficient rights for usage of this service verb!

Note: This verb can be called interactively (not recommended) or by using POST parameters similar to the LibMgt UserLoginForm verb.

Example:

Example Request:
/OLP/MDS/1.0/UserLoginForm

Example Response (success):

```
<?xml version="1.0" encoding="UTF-8"?>
<UserLoginForm version="1.0" status="OK">
<sessionid>tqidmqb3khqxhc45w3jj2j45</sessionid>
<role>user</role>
<ipaddress>141.12.12.27</ipaddress>
<grantedtime>1/6/2003 4:24:27 PM</grantedtime>
<validuntiltime>1/7/2003 4:24:27 PM</validuntiltime>
</UserLoginForm>
```

ApproverLoginForm

```
    Version: 1.0
    Post args: username, password
    Keyword Args: -
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

Before any other MDS service specific verb can be used the user has to pass the authentification procedure. By calling “**ApproverLoginForm**” a request for an “approver” session with sufficient rights to approve documents will be asked. If the authentification was successful a unique session id will be given back.

Note: The username/password combination must resemble a valid **Registry Service** user entry with sufficient rights for usage of this service verb!

Note: This verb can be called interactively (not recommended) or by using POST parameters similar to the LibMgt UserLoginForm verb.

Example:

Example Request:
/OLP/MDS/1.0/ApproverLoginForm

Example Response (success):
```
<?xml version="1.0" encoding="UTF-8"?>
<ApproverLoginForm version="1.0" status="OK">
<sessionid>tqidmqb3khqxhc45w3jj2j45</sessionid>
<role>approver</role>
<ipaddress>141.12.12.27</ipaddress>
<grantedtime>1/6/2003 4:24:27 PM</grantedtime>
<validuntiltime>1/7/2003 4:24:27 PM</validuntiltime>
</ApproverLoginForm>
```

AdminLoginForm

```
    Version: 1.0
    Post args: username, password
    Keyword Args: -
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

Before any other MDS service specific verb can be used the user has to pass the authentification procedure. By calling "**AdminLoginForm**" a request for an "administrative" session with sufficient rights to administrate the MDS will be asked. If the authentification was successful a unique session id will be given back.

Note: The username/password combination must resemble a valid **Registry Service** user entry with sufficient rights for usage of this service verb!

Note: This verb can be called interactively (not recommended) or by using POST parameters similar to the LibMgt UserLoginForm verb.

Example:

Example Request:
/OLP/MDS/1.0/AdminLoginForm

Example Response (success):
```
<?xml version="1.0" encoding="UTF-8"?>
<AdminLoginForm version="1.0" status="OK">
<sessionid>tqidmqb3khqxhc45w3jj2j45</sessionid>
<role>administrator</role>
<ipaddress>141.12.12.27</ipaddress>
<grantedtime>1/6/2003 4:24:27 PM</grantedtime>
<validuntiltime>1/7/2003 4:24:27 PM</validuntiltime>
</AdminLoginForm>
```

EndSession

```
    Version: 1.0
    Fixed args: sessionid
    Keyword Args: -
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

Ends the session that was started formerly by calling “**UserLoginForm**”, “**ApproverLoginForm**” or “**AdminLoginForm**”.

Note: For security reasons the session id “sessionid” will not be assigned or used anymore by the MDS after successful completion of this operation.

Example:

Example Request:
/OLP/MDS/1.0/EndSession/tqidmqb3khqxhc45w3jj2j45

Example Response (success):
<?xml version="1.0" encoding="UTF-8"?>
<EndSession version="1.0" sessionid=" tqidmqb3khqxhc45w3jj2j45" status="OK"/>

Example Response (failure):
<?xml version="1.0" encoding="UTF-8"?>
<EndSession version="1.0" sessionid=" tqidmqb3khqxhc45w3jj2j45" status=" Error: Session cannot be ended. Service is still processing."/>

RequestUpload

```
Version: 1.0
Fixed args: sessionid, pathname
Keyword Args: -
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Notifies the MDS that the user wants to upload one or more files on the MDS’ ftp server. The user has to provide the filename of the “header” file he wants to upload using the “pathname” fixed argument. In return the user receives a user id and a unique document id from the MDS for the file(s) he wants to upload. The user id should then be used as a login on the MDS ftp server.

Note: “pathname” must be enclosed in quotation marks!

Note: After the retrieval of the user id the ftp upload can be started. MDS ensured that everything necessary on the ftp server is set up and ready to go.

Note: Every assigned user id will be unique and only used once.

Note: Every assigned document id will be unique and only used once.

Note: Usage of the assigned user id for login on the server is possible as long as the upload was not declared completely obsolete by using the “**DiscardUpload**” verb or completely successful by using the “**ProcessUpload**” verb.

Note: If the MDS can’t handle the upload request, it will return a meaningful error message.

Example (1):

Example Request:
/OLP/MDS/1.0/ RequestUpload/urnxqv45kpdoj3jcrq43wla1/"long name/256x192.smi"

Example Response (success):
```
<?xml version="1.0" encoding="UTF-8"?>
<RequestUpload version="1.0" status="OK">
<sessionid>urnxqv45kpdoj3jcrq43wla1</sessionid>
<userid>U00ghcugv552s4os455</userid>
<uploadurl>ftp://U00ghcugv552s4os455:U00ghcugv552s4os455@pc-
snaredrum.ipsi.fhg.de:4543</uploadurl>
<documentid>D00ghcugv552s4os455q5mqju45</documentid>
<referencename>long name/256x192.smi</referencename>
</RequestUpload>
```

Example Response (failure):
```
<?xml version="1.0" encoding="UTF-8"?>
<RequestUpload version="1.0" status="Error: Invalid sessionid. Please start a valid session before using this
service verb." />
```

DiscardUpload

```
Version: 1.0
Fixed args: sessionid, userid
Keyword Args: -
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Discards the upload with the upload id “userid”. This id must be an id which was requested while using a former “**RequestUpload**”. All files that were uploaded during the upload session with this “userid” will be deleted by the MDS.

Note: The document id tied to the discarded upload will be declared obsolete and will never be used again.

Note: This verb allows the user of the MDS to delete broken/unsuccessful uploads.

Note: For security reasons the user id “userid” will not be used and assigned to users anymore by the MDS after successful completion of this operation.

Note: Only uploads that were not processed by proper completion of the “**ProcessUpload**” verb can be withdrawn this way. Uploaded documents that already have been processed can only be discarded one by one by using the (restricted) “**WithdrawDocument**” verb.

Example:

Example Request:
```
/OLP/MDS/1.0/ DiscardUpload/3l2cpqzwpebikj55a5fnyo45/U005y5z3r55qhl1sx45
```

Example Response (success):
```
<?xml version="1.0" encoding="UTF-8"?>
<DiscardUpload version="1.0" status="OK">
<sessionid>3l2cpqzwpebikj55a5fnyo45</sessionid>
<userid>u005y5z3r55qhl1sx45</userid>
<documentid>D005y5z3r55qhl1sx45h42hl1ao</documentid>
<referencename>long name/256x192.smi</referencename>
<size>0</size>
<documentstatus>discarded</documentstatus>
</DiscardUpload>
```

ProcessUpload

```
Version: 1.0
Fixed args: sessionid, userid
Keyword Args: -
Return MIME type: text/xml
Return Status Codes: 200, 400
```

This verb must be used to complete the upload procedure with the given upload id “userid”. This id must be an id which was requested while using a former “**RequestUpload**”. All files that were uploaded during the upload session with this user id will now be processed by the MDS. After processing of the files has finished, a response to the caller will be given back.

Note: The response will contain the unique document id of the formerly uploaded files as given by the “RequestUpload” response.

Note: For security reasons the user id “userid” will not be used and assigned to users anymore by the MDS after successful completion of this operation.

Note: If the operation completes with a non fatal failure then it is possible to take the appropriate measures (e.g. retransmission of the not correctly processed files to the MDS ftp-server) and to call “**ProcessUpload**” again until the successful completion of the operation could be accomplished. Alternatively discarding the whole upload by calling “**DiscardUpload**” is also possible.

Note: Processed uploads can’t be discarded. The documents of a processed upload have to be discarded by using the (restricted) “**WithdrawDocument**” verb.

Example Request:
/OLP/MDS/1.0/ ProcessUpload/dzoetr55x4er3o55ucyze355/U00ijqg5ei3wszladvs

Example Response (success):
```
<?xml version="1.0" encoding="UTF-8"?>
<ProcessUpload version="1.0" status="OK">
<sessionid>dzoetr55x4er3o55ucyze355</sessionid>
<userid>u00ijqg5ei3wszladvs</userid>
<documentid>D00ijqg5ei3wszladvshue2oy45</documentid>
<referencename>long name/256x192.smi</referencename>
<size>17558497</size>
<documentstatus>processed</documentstatus>
</ProcessUpload>
```

GetDocumentInfo

```
Version: 1.0
Fixed args: sessionid, documentid
Keyword Args: -
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Retrieves detailed information about the different versions of the document with the id “documentid” that are available. This information may and should be used for retrieval purposes.

Note: It is mandatory to use “**GetDocumentInfo**” whenever a new request for a document with a certain “documentid” comes up, because the returned information of a former “GetDocumentInfo” call with the same “documentid” could be obsolete shortly after the MDS’ response (e.g. because of load balancing on the video servers).

Note: "**GetDocumentInfo**" may also provide information for administrative tasks like usage statistics and reference counters.

Example:

Example Request:
/OLP/MDS/1.0/GetDocumentInfo/s004325/d00018

Example Response (success):
```
<?xml version="1.0" encoding="UTF-8"?>
<GetDocumentInfo version="1.0" sessionid="s004325" documentid="d00018" status="OK">
	<variant url="ftp://ftp.scholnet.org/s1.rm" status="OK" type="Realmedia8" bitrate="450kpbs"/>
	<variant url="http://play.scholnet.org/?url=swave/scholnet/g2demand/020/s1.rm" status="OK"
type="Realmedia8" bitrate="20kpbs"/>
	<variant url="http://play.scholnet.org/?url=swave/scholnet/g2demand/450/s1.rm" status="OK"
type="Realmedia8" bitrate="450kpbs"/>
</GetDocumentInfo>
```

Example Response (failure):
```
<?xml version="1.0" encoding="UTF-8"?>
<GetDocumentInfo version="1.0" sessionid="s004325" documentid="d00018"  status="Error: Document id is
not valid."/>
```

WithdrawDocument

```
Version: 1.0
Fixed args: sessionid, documentid
Keyword Args: -
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Deletes all variants (including the original version) of the document with the id "documentid" that are available on the MDS servers.

Note: Use with caution! Withdrawn documents cannot be restored by the MDS administrators.

Note: The usage of this verb may be restricted to users with administrative MDS access rights.

Note: The document id tied to the withdrawn document will be declared obsolete and will never be used again.

Note: Only documents that are already processed by proper completion of the "**ProcessUpload**" verb can be withdrawn this way. Uploaded documents that haven't been processed can only be discarded by using the "**DiscardUpload**" verb.

Note: Only documents that are not tied to composite documents can be withdrawn. For proper withdrawal of referenced documents the referencing document such as the composite document's master document (smil) has to be withdrawn first or an error will happen. Information about the number of references a document has can be retrieved by using the "**GetDocumentInfo**" verb.

Example:

Example Request:
/OLP/MDS/1.0/WithdrawDocument/s004325/d00018

Example Response (success):
<?xml version="1.0" encoding="UTF-8"?>
<WithdrawDocument version="1.0" sessionid="s004325" documentid="d00018" status="OK"/>

Example Response (failure):
<?xml version="1.0" encoding="UTF-8"?>
< WithdrawDocument version="1.0" sessionid="s004325" documentid="d00018" status="Error: Document reference counter is not zero – Document wasn't withdrawn."/>

ApproveDocument

```
Version: 1.0
Fixed args: sessionid, documentid
Keyword Args: -
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Approves all variants (including the original version) of the document with the id "documentid" that are available on the MDS servers.

Note: The usage of this verb may be restricted to users with administrative MDS access rights.

Note: Only documents that are already processed by proper completion of the "**ProcessUpload**" verb can be approved.

Example:

Example Request:
/OLP/MDS/1.0/ApproveDocument/s004325/d00018

Example Response (success):
<?xml version="1.0" encoding="UTF-8"?>
<ApproveDocument version="1.0" sessionid="s004325" documentid="d00018" status="OK"/>

Example Response (failure):
<?xml version="1.0" encoding="UTF-8"?>
< ApproveDocument version="1.0" sessionid="s004325" documentid="d00018" status="Error"/>

GetUserRole

```
Version: 1.0
Fixed args: sessionid
Keyword Args: -
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Retrieves detailed information about the user's role. Three different roles are currently available:

Administrator - user is allowed to administrate the service
Approver - user is allowed to call restricted verbs ("ApproveDocument" and "WithdrawDocument")
User - user is allowed to call normal service verbs

Example:

Example Request:
/OLP/MDS/1.0/GetUserRole/s004325

Example Response (success):
```
<?xml version="1.0" encoding="UTF-8"?>
<GetUserRole version="1.0" sessionid="s004325" status="OK" username="johndoe" role="Administrator"/>
```

Example Response (failure):
```
<?xml version="1.0" encoding="UTF-8"?>
<GetUserRole version="1.0" sessionid="s004325" status=" Error: GetUserRole failed."/>
```

GetListOfDocuments

```
    Version: 1.0
    Fixed args: sessionid
    Keyword Args: -
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

Retrieves a list of all available original documents and their current status.

Example:

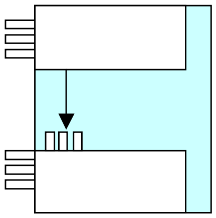
Example Request:
/OLP/MDS/1.0/GetListOfDocuments/s004325

Example Response (success):
```
<?xml version="1.0" encoding="UTF-8"?>
<GetListOfDocuments version="1.0" sessionid="s004325" status="OK">
        <document  documentid="d00018" name="s1.rm" size="456789" status="approved"/>
        <document  documentid="d00019" name="s2.ra" size="943554" status="pending"/>
</GetListOfDocuments>
```

10. LIBRARY MANAGEMENT SERVICE

The Library Management Service supports the submission, withdrawal, and replacement of documents from repositories.
It receives submission, withdrawal, editing of document metadata and document object requests submitted by a registered user, stores these requests in a temporary area, and allows the administrator to approve or to reject them.
The LibMgt administrators are those users authorised by the Registry to manage one or more authorities. An administrator is associated with a set of authorities and, in the Scholnet environment, different administrators manage different authorities.

The Library Management Service (LibMgt) is logically divided into two components as reported below:



The communication with the LibMgt Service takes place via the OpenDLib Protocol (OLP). Note that other services can call either the LibMgt GUI verbs or the LibMgt Engine verbs in accordance with the service specification reported in this paragraph. The communication between LibMgt *GUI* and LibMgt *Engine* takes place via internal API. This architectural choice makes it possible:
1. to use the LibMgt Service functionality with a third party user interface;
2. to easily change the LibMgt GUI to fit the specific needs.

10.1 *STATE*

This section describes the data structure, its abstract type, and gives a brief textual description for each object managed by the Library Management Service.

Authority

A naming authority is an entity that is authorised to create new handles. Naming authorities are hierarchically organised, with periods used as the separator. For example, CNR, CNR.IEI, CNR.IEI.MultimediaDepartment.
The list of authorities managed by a LibMgt Service is set up at service configuration time. This list is harvested from the Meta Service if the LibMgt service is configured as "networked", otherwise the list defined in the service configuration file is take into account.

Set

A set is an administrator-defined subset of the Repository. Each set has one token name and a (possibly) longer description. Depending on the policy of a Repository, an individual document may belong to one or more sets (the decision as to in which sets a document is located is made in the *Submit* request). Note that there is, in general, no way to predict in which set a document appears from its handle.
The list of Repository Sets in which an authority is authorised to publish documents is dynamically calculated pulling information from the appropriate Repository Service instances. These instances are dynamically and automatically selected in accordance with the defined list of authorities.

Document

The LibMgt Service manages documents in accordance with the DoMDL. The information describing the document model is represented in accordance with the Structure Metadata Set (SMS), and the documents are described in accordance with the metadata format specified at service start-up time.

10.2 *LIBMGT VERBS*

In this section, we present the set of the LibMgt verbs. This set is divided into three sub-sets: Service Information Verb, Graphical User Interface Verb, and Service Specific Verb, as reported in the following table:

Service Information Verb	GUI Verb	Service Specific Verb
Identify	AdminLoginForm	AdminLogin
ListVerbs	IncomingSubmitForm	ApproveSubmit
DescribeVerb	IncomingWithdrawForm	ApproveWithdraw
	RejectForm	Reject
ListAuthorities		
	ShowBibRecord	
	ShowStructureRecord	
	UserSubmitForm	UserSubmit
	UserEditForm	UserSubmitNewVersion
	UserWithdrawForm	UserWithdraw

All GUI verbs return an html page that is designed to be displayed by a browser.

Identify

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name and other information about the service.

Example Request:
/OLP/LibMgt/1.0/Identify

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<Identify version="1.0">
   <serviceName>Experimental OLP LibMgt Server</serviceName >
   <olp_base_url>http://labserv.iei.pi.cnr.it:8230</olp_base_url>
   <metaServiceURL>http://labserv.iei.pi.cnr.it:8221</metaServiceURL>
   <textualDescription></textualDescription>
   <protocolVersion>1.0</protocolVersion>
   <adminEmail>pagano@iei.pi.cnr.it</adminEmail>
   <verbsInfo>
      <verbsSupported>
            <description>   </description>
            <olp_base_url>http://labserv.iei.pi.cnr.it:8119/OLP/LibMgt/1.0/ListVerbs</olp_base_url>
      </verbsSupported>
      <verbsDescription>
            <description>   </description>
            <olp_base_url>http://…:8119/OLP/LibMgt/1.0/DescribeVerb/DescribeVerb</olp_base_url>
      </verbsDescription>
```

```
    </verbsInfo>
    <submissionProcedure>
        <description>Submissions accepted from Computer Science Department affiliated researchers
        </description>
            <olp_base_url>http://…:8119/OLP/LibMgt/1.0/UserSubmitForm</olp_base_url>
    </submissionProcedure>
    <harvestInformation>
          <description>
                No information is disseminated by the service
          </description>
          <olp_base_url></olp_base_url>
    </harvestInformation>
    <useRestrictions>Terms and conditions are …...</useRestrictions>
    <contentInfo>
        <contentDescription>
          a human readable description of the content stored in the LibMgt
        </contentDescription>
        <authorities>
          <description>
                A naming authority is an entity that is authorised to create new handles. Naming authorities are
                hierarchically organised, with periods as separators.
                This LibMgt instance allows submitting, editing or deleting of documents in a subset of the
                overall set of authorities.
          </description>
          <olp_base_url>http://labserv….:8119/OLP/LibMgt/1.0/ListAuthorities</olp_base_url>
        </authorities>
        <metadataFormats>
          <description>
                A metadata description is associated with every document in accordance with the OpenDLib
                Application Profile (OLAP)
          </description>
          <olp_base_url>http://labserv.iei.pi.cnr.it:8119/OLP/Repository/1.0/ListMetaFormats</olp_base_url>
          <olp_dtd_url>http://labserv.iei.pi.cnr.it:8119/OLP/htdocs/DTD/olap.dtd</olp_dtd_url>
        </metaFormats>
        <documentStructure>
          <description>
                The Repository Service manages documents in accordance with the DoMDL document model.
                This logical document model is instantiated in the protocol as reported in the sms dtd
          </description>
          <olp_base_url>http://labserv.iei.pi.cnr.it:8119/OLP/htdocs/domdl.html</olp_base_url>
          <olp_dtd_url>http://labserv.iei.pi.cnr.it:8119/OLP/htdocs/DTD/structure.dtd</olp_dtd_url>
        </documentStructure>
    </contentInfo>
</Identify>
```

ListVerbs

```
    Version: 1.0
    Fixed args: none
    Optional Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name of the verbs defined by this service.

Example Request:
/OLP/LibMgt/1.0/ListVerbs

DescribeVerb

```
    Version: 1.0
```

```
Fixed args: verb
Optional Args: version
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on a version of the specified verb that is supported by this service. The following information may be provided at the verb or version level.

- *description*, description of the verb or a specific version
- *note*, information pertaining to the verb or a specific version

Each element of the list contains the following information:

- *version* number of the verb.
- *arguments*, a list of the names of the fixed and optional arguments, if any, accepted by the verb in that version.
- *example* template of request to this LibMgt, with fixed arguments indicated in brackets
- *returns*, optional, contains information about response format.

Note that a service may implement more than one version of a verb.

Example Request:
/OLP/LibMgt/1.0/DescribeVerb/AdminLoginForm

ListAuthorities

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured list of the authorities managed by this LibMgt. The short and long name is indicated for each authority.

Example Request:
/OLP/LibMgt/1.0/ListAuthorities

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <ListAuthorities version="1.0">
    <authority name="scholnet.one" display="Scholnet Project Repository"/>
    <authority name="scholnet.two" display="Scholnet Project Test Repository"/>
  </ListAuthorities>
```

AdminLoginForm

```
Version: 1.0
Fixed args: none
Optional_post Args: login, password
Return MIME type: text/html
Return Status Codes: 200, 400, 501
```

This page allows the administrator to login to the administration environment.

If the login and password are specified as optional_post arguments, passed as HTTP POST requests, the LibMgt GUI service starts the session without showing the form on which these values can be input. In this way the verb can also be called by other services that have already authenticated the user as LibMgt administrator.

After the identification of the LibMgt administrator, the verb displays the administration home environment. From this environment, the administrator can access all the functionality provided.

Example Request:
/OLP/LibMgt/1.0/AdminLoginForm

IncomingSubmitForm

```
    Version: 1.0
    Fixed args: none
    Optional Args: none
    Return MIME type: text/html
    Return Status Codes: 200, 400, 501
```

This page shows all incoming requests to submit/edit a new/existing document submitted to the authorities administered by the given administrator. The set of authorities is managed as parameter of the session.
This verb is usable only after the identification of the administrator. The login and password of the administrator are managed as parameters of the session and used to verify that only authorised users access this verb. If a session has not yet been initialized, the verb automatically calls the AdminLoginForm.

For each incoming request, this page shows the following information:

- the date of the submission
- the author of the request
- a link to the bibliographic record that describes the document
- a link to the structure metadata record that represents the document
- the authority and the set in which to publish the document

The link to the bibliographic/structure metadata is:

- a pointer to the incoming area of the LibMgt for the submit new document request
- a request to the Repository that stores the document for the edit existing document request.

The administrator can approve or reject any shown requests.

Example Request:
/OLP/LibMgt/1.0/IncomingSubmitForm

IncomingWithdrawForm

```
    Version: 1.0
    Fixed args: none
    Optional Args: none
    Return MIME type: text/html
    Return Status Codes: 200, 400, 501
```

This page shows all incoming requests to withdraw/delete an existing document submitted to the authorities administered by the given administrator. The set of authorities is managed as parameter of the session.
This verb is usable only after the identification of the administrator. The login and password of the administrator are managed as parameters of the session and used to verify that only authorised users access this verb. If a session ihas not yet been initialized, the verb automatically calls the AdminLoginForm.

For each incoming request, this page shows the following information:

- the date of the submission
- the author of the request
- a link to the bibliographic record that describes the document
- a link to the structure metadata record that represents the documents
- the authority and the set in which the document is published.

The link to the bibliographic/structure metadata is always a request to the Repository that stores the document.

The administrator can approve or reject any showed requests.

Example Request:
/OLP/LibMgt/1.0/IncomingWithdrawForm

RejectForm

```
Version: 1.0
Fixed args: handle, version, user
Optional Args: none
Return MIME type: text/html
Return Status Codes: 200, 400, 501
```

This page allows the administrator to specify the reason for rejection. The system automatically sends an e-mail message to the author as response to the incoming request.
This verb is usable only after the identification of the administrator. The login and password of the administrator are managed as parameters of the session and make it possible to verify that only authorised users can access this verb.
For each incoming request, this page allows the following information to be specified:

- the date of the rejection
- the reason for rejection

The administrator also decides whether he/she want to keep/delete the incoming request. In the first case, the author of the request can correct the request without completely re-submitting it.

Example Request:
/OLP/LibMgt/1.0/RejectForm

ShowBibRecord

```
Version: 1.0
Fixed args: handle
Optional Args: type, modality
Return MIME type: text/html
Return Status Codes: 200, 400, 404, 501
```

The page shows the bibliographic metadata record in accordance with the OLAP application profile.

The *type* argument can assume one of the following values:

- *incoming*, means that the document has not yet been approved and therefore all files about it are maintained by the LibMgt Service

- *repository*, means that the document is already approved and therefore all files about it are maintained by the Repository Service

If the *type* argument is not specified, the *incoming* mode is assumed as default.

The *modality* argument can assume one of the following values:

- *view*, means that the bibliographic record is shown only in view mode. Changes to the bibliographic record are not allowed.
- *edit*, means that the bibliographic record is shown in edit mode allowing changes to the values reported in the bibliographic record.

If the *modality* argument is not specified, the *view* mode is assumed as default.

Example Request:
/OLP/LibMgt/1.0/ShowBibRecord/CNR.IE/TR-103?type=incoming&modality=view

ShowStructureRecord

```
Version: 1.0
Fixed args: handle
Optional Args: modality, type
Return MIME type: text/html
Return Status Codes: 200, 400, 404, 501
```

This page shows the structure metadata record in accordance with the SMS dtd.

The *type* argument can assume one of the following values:

- *incoming*, means that the document has not yet been approved and therefore all files about it are maintained by the LibMgt Service
- *repository*, means that the document is already approved and therefore all files about it are maintained by the Repository Service

If the *type* argument is not specified, the *incoming* mode is assumed as default.

The *modality* argument can assume one of the following values:

- *view*, means that the bibliographic record is shown only in view mode. Changes to the bibliographic record are not allowed.
- *edit*, means that the bibliographic record is shown in edit mode allowing changes to the values reported in the bibliographic record.

If the *modality* argument is not specified, the *view* mode is assumed as default.

Example Request:
/OLP/LibMgt/1.0/ShowStructureRecord/CNR.IEI/TR-103?type=incoming&modality=view

UserSubmitForm

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/html
Return Status Codes: 200, 400, 501
```

This page allows the user to fill in all information in accordance with the OLAP application profile and with the SMS DTD.
The user has to submit the following information:

- user identification and password
- the bibliographic record in accordance with the OLAP application profile
- the logical structure of the document. The structure is automatically composed by the service in accordance with the steps performed by the user. This structure can be edited in order to add new logical views of the documents (for which the user does not have to submit a related manifestation) or to submit manifestations related to a specific view. The structured information is validated in accordance with the SMS DTD. When a user specifies a manifestation he/she has to specify:
  - the path location of the file if it is not a video or composite manifestation;
  - an FTP location where the system can download the file or its file name, if it is a video or composite manifestation.

    The LibMgt service sends a message to the Multimedia Storage Service indicating the file name and receives an URN for the resource and a user-id. The URN is associated with the manifestation, and the user-id is used to upload the file to the MDS, if the user has specified an FTP location, or is given to the user that uses it to upload the file directly to the MDS.

    Note that the LibMgt cannot access or download video manifestations, which are directly downloaded by the Multimedia Storage Service. This is because a video/composite manifestation has normally a dimension of some hundreds mega-bytes.

Note that the user can also indicate the bibliographic record and the logical structure as XML files. The information extracted from these files allows the LibMgt service to pre-load the LibMgt GUI environment.
Using this additional functionality, the user can prepare both document descriptive files with another tool (such as the Metadata Editor, or the SMIL editor) and then complete the operation with the LibMgt service. The use of a specialised tool can be helpful when managing special document types such as composite documents or video documents.

Example Request:
/OLP/LibMgt/1.0/UserSubmitForm

UserEditForm

```
Version: 1.0
Fixed args: step
Optional Args: none
Return MIME type: text/html
Return Status Codes: 200, 400, 501
```

This page allows the user to change the metadata and the content associated with an existing document.
The LibMgt Service manages this verb in multiple steps.
In the first step the user has to submit the following information (if not already provided):

- The user identification and password.

In the second step:

- the user can select the handle of the document that he/she wants to change. Note that a user can only ask to change those documents of which he/she is owner
- the user specifies a comment that reports the changes with respect to the current version.

The handle allows the LibMgt to identify whether the document is stored in the incoming area of the service or in the Repository Service.
In the third step the user can modify the following information:

- the bibliographic record shown

- the logical structure shown for the document. This structure can be edited in order to:
  - add new logical views to document (for which the user does not have to submit a related manifestation)
  - submit related manifestations to a specific view
  - change existing manifestations
  - delete a view/manifestation
  - change information related to a view/manifestation

The structured information is validated in accordance with the SMS dtd.
Steps 2 and 3 are can only be performed after the user has been identified. The login and password of the registered user are managed as parameters of the session and used to verify that only authorised users access this verb.

Example Request:
/OLP/LibMgt/1.0/UserEditForm/1

UserWithdrawForm

```
Version: 1.0
Fixed args: step
Optional Args: none
Return MIME type: text/html
Return Status Codes: 200, 400, 501
```

This page allows the user to request the withdrawal or deletion of a document. A withdraw request, if approved, makes only the bibliographic/structure metadata of the document accessible. A delete request, if approved, deletes all information about the document. The delete request is supported but is discouraged.
In the first step, the user has to submit the following information (if not already provided):
- The user identification and password.

In the second step:
- the user can select the handle of the document that he/she wants to withdraw/delete. Note that a user can only submit a request for those documents of which he/she is owner.

In the third step the user has to specify the following information:
- the kind of the request (withdraw, delete)
- the reason
- optionally, the new address of the document.

Steps 2 and 3 can only be performed after the identification of the user. The login and password of the registered user are managed as parameters of the session and used to verify that only authorised users access this method.

Example Request:
/OLP/LibMgt/1.0/UserWithdrawForm/1

AdminLogin

```
Version: 1.0
Fixed_post args: login, password
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

This is used to verify the identification of an administrator. It checks that the login and password, passed as HTTP POST request, are correct and returns the list of authorities that the user is authorised to administer.

```
Example Request:
/OLP/LibMgt/1.0/AdminLogin

Example Response:
<?xml version="1.0" encoding="UTF-8"?>
<AdminLogin version="1.0">
   <session-id value=""/>
   <login>test</login>
   <authorities>
      <authority name="……." />
      <authority name="……." />
      ……………….
   </authorities>
 </AdminLogin>
```

ApproveSubmit

```
Version: 1.0
Fixed args: type, doc-id, partiotionspec, authority
Optional Args: comment
Optional_post Args: session-id, login, password
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

This is used to complete the document submission phase, improving the request submitted by the user.
This verb can only be used after the identification of the administrator. The login and password of the administrator are managed as parameters of the session and used to verify that only authorised users access this method. If a session has not yet been initialized, the verb can be used specifying the login and password of a LibMgt administrator as *optional arguments*. In this way the verb is usable either by the GUI, that creates a session with the Engine, or by other services.

The *type* argument can assume one of the following values:

- *submit*, means that the administrator has approved the publication of a new document
- *edit,* means that the administrator has approved the editing of a published document.

The doc-*id* argument specifies the document identifier. The *partiotionspec* argument specifies the set in which the document has to be published. The authority makes it possible to verify that the user/administrator can submit/approve a document in/for this authority.
The verb verifies that the following information is available in the LibMgt incoming area for the specified document:

- the structure metadata file with MIME type text/xml in accordance with the SMS dtd,
- the bibliographic metadata file with MIME type text/xml in accordance with the OLAP dtd,
- one or more files with the appropriate MIME types, each of which is a manifestation in a format that the repository is able to accept as an input format.

The optional comment argument is mandatory if the type is “edit”.

If the document contains external, i.e. video or composite, manifestations submitted by the user, an additional step is performed:

- a session is started with the Multimedia Document Storage Service (MDS) in order to confirm approval of the document. In this session, ProcessUpload and then ApproveDocument are invoked using the appropriate parameters.

If the submit request is successful the structured return contains:

- handle of the submitted document
- partiotionspec indicates the set specification for each of the sets to which the document was submitted.

Example Request:

/OLP/LibMgt/1.0/ApproveSubmit/submit/TR-103/cs;ai/CNR.IEI

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <ApproveSubmit version="1.0">
    <handle>CNR.IEI/TR-103</handle>
    <set name="cs">
      <display>Computer Science</display>
      <set name="ai">
        <display>Artificial Intelligence</display>
      </set>
    </set>
  </ApproveSubmit>
```

ApproveWithdraw

```
Version: 1.0
Fixed args: doc-id, authority, version
Optional Args: reason, type, nosave
Optional_post Args: session-id, login, password
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

This is used to complete the document withdraw phase, improving the request submitted by the user.
This verb can only be used after the identification of the administrator. The login and password of the administrator are managed as parameters of the session and used to verify that only authorised users access this method. If a session has not yet been initialized, the verb can be used specifying the login and password of a LibMgt administrator as *optional arguments*. In this way, the verb is usable either by the GUI, which creates a session with the Engine, or by other services.

The doc-*id* argument specifies the document identifier. The authority verifies that the administrator can withdraw/delete documents of this authority.

The *reason* is a text string specifying the reason for withdrawal. A common use of this string is to state a new access point for the document.
The *type* argument can assume one of the following values:

- *nodelete*, means that the administrator has requested that manifestations of the document are removed. However, all information reported in the metadata and structured files are maintained by the repository.
- *delete* specifies that the bibliographic metadata record and the structure metadata record for the document should also be deleted.

If the type argument is not specified, the *nodelete* option will be considered as default.

The *nosave* argument is a suggestion to the repository that the deleted content should not be archived (e.g., maintained by the repository but not accessible through the protocol).

If the document contains external, i.e. video or composite, manifestations submitted by the user, an additional step is performed:

- a session is started with the Multimedia Document Storage Service (MDS) in order to confirm the document withdrawal. In this session, the WithdrawDocument method is invoked using the appropriate parameters.

If the withdraw request is successful the structured response contains the handle of the withdrawn document.

Example Request:
/OLP/LibMgt/1.0/ApproveWithdraw/TR010101/CNR.IEI?reason=the document has been published in DLib

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
   <ApproveWithdraw version="1.0">
      <handle>CNR.IEI/TR010101</handle>
   </ApproveWithdraw>
```

Reject

```
Version: 1.0
Fixed args: type, user-id
Optional Args: reason, title, date
Optional_post Args: session-id, login, password
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

The system automatically sends an e-mail message to the author as response to the incoming request.
This verb can only be used after the identification of the administrator. The login and password of the administrator are managed as parameters of the session and used to verify that only authorised users access this method. If a session has not yet been initialized, the verb can be used specifying the login and password of a LibMgt administrator as *optional arguments*. In this way, the verb is usable either by the GUI, which creates a session with the Engine, or by other services.

The *user-id* argument specifies the owner of the request.
The type argument can assume one of the following values:

- *keep*, the request is not deleted and the owner of the request may correct the request without completely re-submiting it.
- *delete*, the request is deleted from the incoming area

The *reason* argument specifies the comment of the administrator. The *title* argument specifies the title of the document, object of the request. The *date* argument specifies the date of rejection.

If the document contains external, i.e. video or composite, manifestations submitted by the user an additional step is performed:

- a session is started with the Multimedia Document Storage Service (MDS) in order to confirm the document rejection. In this session, the DiscardUpload method is invoked using the appropriate parameters.

If the request is successful, confirmation is returned to the client.

Example Request:
/OLP/LibMgt/1.0/Reject/keep/pagano01?title=The OLP protocol&date=21-04-2001
Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <Reject version="1.0">
    <title> The OLP protocol </title>
    <type>keep</type>
  </Reject>
```

UserWithdraw

```
Version: 1.0
Fixed args: type, handle
Optional Args: new-addr, reason
Optional_post Args: session-id, login, password
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Stores a withdraw request in the incoming area.
This verb can only be used after the identification of the user. The login and password of the registered user are managed as parameters of the session and used to verify that only authorised users can access this method. If a session has not yet been initialized, the verb can be used specifying the login and password as *optional arguments*.
The *handle* argument specifies the authority and identifier of the document object of the request.
The *type* argument can assume one of the following values:

- *withdraw*, means that the user has requested to remove the manifestations of the document. All information reported in the metadata and structured files can be maintained by the repository.
- *delete* specifies that also the bibliographic metadata record and the structure metadata record for the document should be deleted.

The *new-addr* argument specifies the new address of the document, if this exists. The *reason* argument allows the user to justify the reason for the request.
If the request is successful, confirmation is returned to the client.

Example Request:
/OLP/LibMgt/1.0/UserWithdraw/delete/CNR.IEI/TR-01-01
Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <UserWithdraw version="1.0">
    <handle>CNR.IEI/TR-01-01</handle>
    <type>delete</type>
  </UserWithdraw>
```

UserSubmit

```
Version: 1.0
Fixed args: none
Fisex_post Args: file
Optional Args: none
Optional_post Args: session-id, login, password
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Stores a submit request in the incoming area.
This verb can only be used after the identification of the user. The login and password of the registered user are managed as parameters of the session and used to verify that only authorised

users access this method. If a session has not yet been initialized, the verb can be used specifying the login and password as *optional arguments*.
All information is passed from the UI to the LibMgt engine as an HTTP POST request.
The verb produces the following information in the LibMgt incoming area for the document object of the request:

- the structure metadata file with MIME type text/xml in accordance with the SMS dtd,
- the bibliographic metadata file with MIME type text/xml in accordance with the OLAP dtd,
- a MIME multipart/mixed type that contains all inside manifestations submitted by the user.

If the document contains external, i.e. video or composite, manifestations submitted by the user, an additional step is performed:

- a session is started with the Multimedia Document Storage Service (MDS) in order to confirm approval of the document. In this session, the RequestUpload method is invoked using the appropriate parameters.

If the request is successful, aconfirmation is returned to the client. The confirmation message reports:

1. the temporal doc-id assigned to the document. This number can be used to simplify communication with the administrator.
2. the user-id returned by the MDS, if the document contains external manifestations. The user has to use this user-id in order to upload the document to the MDS ftp server.

```
Example Request:
/OLP/LibMgt/1.0/UserSubmit
Example Response:
<?xml version="1.0" encoding="UTF-8"?>
   <UserSubmit version="1.0">
      <doc-id>213-1313-3314</doc-id>
   </UserSubmit>
```

UserSubmitNewVersion

```
Version: 1.0
Fixed args: none
Fisex_post Args: file
Optional Args: none
Optional_post Args: session-id, login, password
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Stores a submit new version request in the incoming area.
This verb can only be used after the identification of the user. The login and password of the registered user are managed as parameters of the session and used to verify that only authorised users access this method. If a session has not yet been initialized, the verb can be used specifying the login and password as *optional arguments*.
All information is passed from the UI to the LibMgt engine as an HTTP POST request.
The verb produces the following information in the LibMgt incoming area for the document object of the request:

- the structure metadata file with MIME type text/xml in accordance with the SMS dtd,
- the bibliographic metadata file with MIME type text/xml in accordance with the OLAP dtd,
- a MIME multipart/mixed type that contains all inside manifestations submitted by the user.

If the document contains external, i.e. video or composite, manifestation submitted by the user, an additional step is performed:

- a sessionis started with the Multimedia Document Storage Service (MDS) in order to confirm approval of the document. In this session, the RequestUpload method is invoked using the appropriate parameters.

If the request is successful, confirmation is returned to the client. The confirmation message reports:

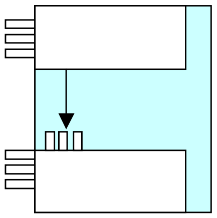
1. the temporal doc-id assigned to the document. This number can be used to simplify communication with the administrator.
2. the user-id returned by the MDS, if the document contains external manifestations. The user has to use this user-id in order to upload the document to the MDS ftp server.

```
Example Request:
/OLP/LibMgt/1.0/UserSubmitNewVersion
Example Response:
<?xml version="1.0" encoding="UTF-8"?>
    <UserSubmitNewVersion version="1.0">
        <doc-id>213-1313-3314</doc-id>
    </UserSubmitNewVersion>
```

11. REGISTRY SERVICE

The Registry Service of the Scholnet system handles registration of users and groups. A user is identified by a unique login, while a group is identified by a groupname. Every group is represented by its owner; this is specified (changed) together with other settings through the group profile.
The Digital Library administrator is established at service start-up time and can assign special rights to the registered users in order to give them the appropriate administration rights on those services of the architecture that need an administrator. In the current version of the Scholnet system, these services are the Registry, the Repository, the LibMgt and the Collection Service.
A Repository administrator can assign document submission rights in his/her authorities to any registered user. Such users become "submitters" and can also edit or delete documents.
A LibMgt administrator can approve or reject any submission, edit or withdraw requests submitted by a submitter to his/her authorities.
A Collection Service administrator can create collection, and edit or delete his/her own collections.
A Thesaurus Service administrator can manage the service in order to release a new thesaurus; create, update, delete, or rename terms an existing thesaurus; create, delete, or move hierarchies in a thesaurus; create a (new) facet of the thesaurus.
A Registry administrator can assign and remove user rights.

The Registry is logically divided in two components as reported below:



Communication with the Registry Service takes place via the OpenDLib Protocol (OLP). Note that other services can call either the Registry GUI verbs or the Registry Engine verbs in accordance with the service specification reported in this section. The communication between Registry *GUI* and Registry *Engine* takes place via internal API. This architectural choice makes it possible:
1. to use the Registry Service functionality with a third party user interface;
2. to easily change the Registry GUI to fit the specific needs.

11.1 *STATE*

User Profile

The Registry service manages user profiles in accordance with the UserProfile format that is reported in Appendix E (UserProfile DTD). The user profile has three different parts: the first reports information on the user identity and can be edited by the user; the second reports information on the documents submitted by the user and is updated in the Scholnet environment only by the LibMgt Service; the last reports information on the user rights and can only be updated using the Registry UI or the Registry Engine.
A user with a known e-mail address is identified by a unique login. All the other settings in the user profile are optional.

	Attribute	Obligation	Type	Constraint/Comment
SETTINGS	Login	Mandatory	String	alphanumeric characters plus underscore, unique in the system
	Password	Mandatory	String	user's password to the system
	FullName	Mandatory	String	user's first name(s) and surname
	Email	Mandatory	String	valid e-mail contact for the user
	Institution	Optional	String	user's institution
	Address	Optional	String	contact address for the user
	Phone	Optional	String	contact phone for the user
	Comment	Optional	String	any text
DATA	HandleList	Optional	Set	set of document handles associated with the user (the user is the owner of these documents)
RIGHTS	Authorities	Optional	Set	set of authorities in which the user can submit documents. If the set is empty the user cannot submit documents.
	Collection-Adm	Optional	Boolean	specifies whether the user can administer the Collection Service
	LibMgt-Adm	Optional	Set	specifies the set of authorities that can be administered with the Library Management Service
	Repository-Adm	Optional	Set	specifies the set of authorities that can be administered
	Registry-Adm	Optional	Boolean	specifies whether the user can administer the Registry Service
	Thesaurus-Adm	Optional	Boolean	specifies whether the user can administer the Thesaurus Service

Group Profile

The Registry service manages group profiles in accordance with the GroupProfile format that is reported in the Appendix F (GroupProfile DTD).
A group is identified by a unique groupname. A group must have an owner specified (a user already registered in the system) and a short description - the other settings in the group profile are optional.
A group can be defined public or private. A public group is visible to all registered users and each registered user can ask to join the group sending a message to the administrator. A private group (introduced to support a project or a restricted team of users) is only visible to its members.

	Attribute	Obligation	Type	Constraint/Comment
SETTINGS	GroupName	Mandatory	String	alphanumeric characters plus underscore, unique in the system
	Description	Mandatory	String	short description of the group
	Owner	Mandatory	String	login of the group owner, must already be registered

	UserList	Optional	Set	login of each registered user belonging to the group
	Comment	Optional	String	any text
	Public	Mandatory	Boolean	default value is true.

11.2 *REGISTRY VERBS*

In this section, we present the set of the registry verbs. This set is divided into three sub-sets: Service Information Verbs, User Verbs, and GroupVerbs as reported in the following table:

Service Information Verbs	User Verbs	Group Verbs
Identify	AddUser	AddGroup
ListVerbs	AuthenticateUser	
DescribeVerb	DisplayUserInfo	DisplayGroupInfo
	ModifyUserInfo	ModifyGroupInfo
	DeleteUser	DeleteGroup
	AddUserHandle	AddUserToGroup
	AuthenticateUserHandle	AuthenticateGroupUser
	DeleteUserHandle	DeleteUserFromGroup
	ListUsers	ListGroups
	ListUserHandles	ListGroupUsers
		ListUserGroups
	EditUserRights	
	EditUserRightsForm	
	AdminLogin	
	AdminLoginForm	

Identify

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name and other information about the service.

Example Request:
/OLP/Registry/1.0/Identify

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<Identify version="1.0">
    <serviceName>Experimental OLP Registry Server</serviceName >
    <olp_base_url>http://labserv.iei.pi.cnr.it:8230</olp_base_url>
    <metaServiceURL>http://labserv.iei.pi.cnr.it:8221</metaServiceURL>
    <textualDescription></textualDescription>
    <protocolVersion>1.0</protocolVersion>
    <adminEmail>pagano@iei.pi.cnr.it</adminEmail>
    <verbsInfo>
        <verbsSupported>
```

```
            <description>   </description>
            <olp_base_url>http://labserv.iei.pi.cnr.it:8230/OLP/Registry/1.0/ListVerbs</olp_base_url>
        </verbsSupported>
        <verbsDescription>
            <description>   </description>
            <olp_base_url>http://…:8230/OLP/Registry/1.0/DescribeVerb/DescribeVerb</olp_base_url>
        </verbsDescription>
    </verbsInfo>
    <submissionProcedure>
        <description>The service supports public registration of the user. The registered user can access
                additional services that include Group Management, Annotation, and Personalised
                Information Dissemination. Special rights can also be assigned to registered users by an
                administrator in order to allow them to submit documents to the library
        </description>
          <olp_base_url>http://…/OLP/UI/1.0/GenerateRegistrationForm</olp_base_url>
    </submissionProcedure>
    <harvestInformation>
        <description>
                The individual users and groups list is available
          </description>
          <olp_base_url>http://…:8230/OLP/Registry/1.0/ListUsers</olp_base_url>
          <olp_base_url>http://…:8230/OLP/Registry/1.0/ListGroups</olp_base_url>
    </harvestInformation>
    <useRestrictions>Terms and conditions are …...</useRestrictions>
    <contentInfo>
        <userProfile>
            <description>
                A metadata description is associated with every user in accordance with the OpenDLib
                UserProfile format.
            </description>
            <olp_base_url>http://labserv.iei.pi.cnr.it:8119/OLP/htdocs/userprofile.html</olp_base_url>
            <olp_dtd_url>http://labserv.iei.pi.cnr.it:8119/OLP/htdocs/DTD/userprofile.dtd</olp_dtd_url>
        </userProfile>
        <groupProfile>
            <description>
                A metadata description is associated with every group in accordance with the OpenDLib
                GroupProfile format.
            </description>
            <olp_base_url>http://labserv.iei.pi.cnr.it:8119/OLP/htdocs/groupprofile.html</olp_base_url>
            <olp_dtd_url>http://labserv.iei.pi.cnr.it:8119/OLP/htdocs/DTD/groupprofile.dtd</olp_dtd_url>
        </groupProfile>
    </contentInfo>
</Identify>
```

ListVerbs

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Returns a structured response containing the name of the verbs defined by this service.
Example Request:
/OLP/Registry/1.0/ListVerbs

DescribeVerb

```
Version: 1.0
Fixed args: verb
```

```
Optional Args: version
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Returns a structured response that contains a list in which each element of the list provides information on a version of the specified verb that is supported by this service. The following information may be provided at the verb or version level:

- *description*, description of the verb or a specific version,
- *note*, information pertaining to the verb or a specific version.

Each element of the list contains the following information:

- *version,* number of the verb,
- *arguments*, a list of the names of the fixed and optional arguments, if any, accepted by the verb in that version,
- *example,* template of request to this repository, with fixed arguments indicated in brackets,
- *returns*, optional, contains information about response format.

Note that a service may implement more than one version of a verb.

Example Request:
/OLP/Registry/1.0/DescribeVerb/AddUser

AddUser

```
Version: 1.0
Fixed_post args: uploaded_file
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 401
```

A new user is added to the Registry.
The user-settings are transmitted as XML file in an HTTP POST request.
The *fullname, password,* and *email* arguments are mandatory.
Returns a structured response that contains at least:

- login, a new unique login name assigned to the user by the system,
- code, 200 if the addition was successful, an error code (400, 401) otherwise.

Example Request:
/OLP/Registry/1.0/AddUser/%s
Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<AddUser version="1.0">
	<login>newuser</login>
	<code>200</code>
</AddUser>
```

AuthenticateUser

```
Version: 1.0
Fixed_post args: login, password
Optional Args: service
Return MIME type: text/xml
Return Status Codes: 200, 400, 401
```

Checks whether the user is already registered and has a password to the system.
The *login* and *password* arguments are transmitted as an HTTP POST request.

If the service argument is specified, the Registry service only returns information on the service required. The accepted values are the following: submit, collection, libmgt, repository, registry.
Returns error code 400 if the user is not authenticated, otherwise returns an xml response that contains the user rights.

Example Request:
/OLP/Registry/1.0/AuthenticateUser/%s/%s

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<AuthenticateUser version="1.0">
    <login>test</login>
    <authorities>
        <authority>ercim.cnr.iei</authority>
        <authority>ercim.gmd</authority>
    </authorities>
    <collection-adm value="0" />
    <libmgt-adm>
        <authorities>
            <authority>ercim.cnr.iei</authority>
        </authorities>
    </libmgt-adm>
    <repository-adm>
        <authorities>
            <authority>ercim.gmd</authority>
        </authorities>
    </repository-adm>
    <registry-adm value="0" />
 </AuthenticateUser>
```

DisplayUserInfo

```
Version: 1.0
Fixed args: login
Optional_post: password
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Returns a structured response that contains all settings of the given user:
- login, user's login in the registry,
- full name, user's first name(s) and surname,
- email, user’s contact e-mail ,
- institution, user's institution,
- address, user's contact address,
- phone, user's contact phone,
- comment, any text.

Example Request:
/OLP/Registry/1.0/DisplayUserInfo/user

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<DisplayUser version="1.0">
            <login>test</login>
            <fullname>Test User</fullname>
            <email>test@scholnet.org</email>
            <institution>Scholnet</institution>
            <address>wherever he lives 21</address>
            <phone>+666999666</phone>
            <comment></comment>
</DisplayUser>
```

ListUserHandles

```
     Version: 1.0
     Fixed args: login
     Optional Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

Returns a structured response that contains a list of handles, each of which is related to a document, submitted by the user:

Example Request:
/OLP/Registry/1.0/ListUserHandles/user

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListUserHandles version="1.0">
    <documents>
        <document handle="scholnet.test/2002-test-012" version="1" />
        <document handle="scholnet.test/2002-NewTest-002" version="1" />
        <document handle="scholnet.test/1999-isl-21" version="1" />
        <document handle="scholnet.test/2002-test-022" version="1" />
        <document handle="scholnet.test/2002-NewTest-006" version="1" />
    </documents>
</ListUserHandles>
```

ModifyUserInfo

```
     Version: 1.0
     Fixed_post args: DisplayUserInfo
     Optional Args: none
     Return MIME type: text/plain
     Return Status Codes: 200, 400, 401
```

Changes settings of a given user. Only the user can change his/her settings.
The user-settings are transmitted as an XML file in an HTTP POST request.
The *fullname, password,* and *email* arguments are mandatory.
Returns a structured response that contains at least:

- login, a new unique login name assigned to the user by the system,
- code, 200 if the addition was successful, an error code (400, 401) otherwise

Example Request:
/OLP/Registry/1.0/ModifyUserInfo/%s

DeleteUser

```
     Version: 1.0
     Fixed args: none
     Fixed_post args: login, password
     Optional Args: none
     Return MIME type: text/plain
     Return Status Codes: 200, 400
```

Deletes the user (identified by the login) from the Registry and removes him/her from all groups, where he/she is a member.
The *login* and *password* arguments are transmitted as an HTTP POST request.
Returns 200 if the user can be deleted, the 400 error code otherwise.

Example Request:
/OLP/Registry/1.0/DeleteUser/%s/%s

AddUserHandle

```
     Version: 1.0
     Fixed args: none
     Fixed_post args: owner, admin, adminpass, handle
     Optional Args: version
     Return MIME type: text/plain
     Return Status Codes: 200, 400, 401
```

The registry maintains a relationship between users and their documents. This relationship allows an owner to be associated with a document in order to check withdraw, delete, or new-version requests. This verb allows a document handle to be associated with a specific user that becomes the owner of the document.
The *owner, admin, adminpass, handle* arguments are transmitted as an HTTP POST request. The admin and the admpass are userid and password of the LibMgt administrator that has approved the submission of a new document. The LibMgt service checks that:

- the password of the administrator is correct;
- the administrator can approve a submission for the authority specified.

Returns 200 if no error is generated, or an error code (400 or 401) otherwise.

Example Request:

/OLP/Registry/1.0/AddUserHandle/%s/%s/%s/%s

AuthenticateUserHandle

```
     Version: 1.0
     Fixed args: none
     Fixed_post args: owner, handle
     Optional Args: version
     Return MIME type: text/plain
     Return Status Codes: 200, 400
```

Verifies whether a user is the owner of a document.
The *login,* and *handle* arguments are transmitted as an HTTP POST request.
Returns 200 if authenticated, or the error code number otherwise.
Example Request:
/OLP/Registry/1.0/AuthenticateUserHandle/%s/%s

DeleteUserHandle

```
     Version: 1.0
     Fixed args: none
     Fixed_post args: owner, admin, adminpass, handle
```

```
    Optional Args: none
    Return MIME type: text/plain
    Return Status Codes: 200, 400, 401
```

Removes the document handle from the list of handles owned by the user.

The *owner, admin, adminpass, handle* arguments are transmitted as an HTTP POST request. The admin, adminpass are the login and the password of the LibMgt administrator that has approved the deletion of the document. The LibMgt service checks that:

- the password of the administrator is correct;
- the administrator can approve the user delete request for the authority specified.

Returns 200 if removal was successful or an error code (400 or 401) otherwise.

Example Request:
/OLP/Registry/1.0/DeleteUserHandle/%s/%s/%s/%s

ListUsers

```
    Version: 1.0
    Fixed args: none
    Optional Args: format
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

Returns a structured response indicating the login and possibly other settings from the user's profile. *Format* can be either 'short', which returns only login for each user, or 'long', which returns all setting attributes. The default value for format is 'short'.
Example Request:
/OLP/Registry/1.0/ListUsers?format=long

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<ListUsers version="1.0" format="long">
        <user>
          <login>test</login>
          <fullname>Test User</fullname>
          <email>test@scholnet.org</email>
          <institution>Scholnet</institution>
          <address>wherever he lives 21</address>
          <phone>+666999666</phone>
          <comment></comment>
        </user>
        <user>
          ...
        </user>
</ListUsers>
```

AdminLoginForm

```
    Version: 1.0
    Fixed args: none
    Optional_post Args: login, password
    Return MIME type: text/html
    Return Status Codes: 200, 400
```
This page allows the administrator to login in the Registry administration environment.

If the login and password are specified as optional arguments, passed as an HTTP POST request, the Registry GUI service starts the session without showing a form in which these values can be entered. In this way the verb can be called also by other services that have already authenticated the user as the Registry administrator.
After the identification of the Registry administrator, the verb displays the administration home environment. Using this environment the administrator can access all the functionality provided.

Example Request:
/OLP/Registry/1.0/AdminLoginForm

EditUserRightsForm

```
Version: 1.0
Fixed args: step
Optional Args: none
Return MIME type: text/html
Return Status Codes: 200, 400
```

The page allows the Registry administrator to manage the rights of a user.
In the first step, the administrator has to submit his/her identification and password, if not previously provided.
In the second step, the administrator can select the login of a given user.
In the third step, the administrator can change any of the user rights.
Steps 2 and 3 can only be used after the identification of the user. The login and password of the registered user are managed as parameters of the session and used to verify that only authorised users access this method. If a session has not yet been initialized, the verb automatically calls the AdminLoginForm.

Example Request:
/OLP/Registry/1.0/UserWithdrawForm/1

EditUserRights

```
Version: 1.0
Fixed args: login
Fixed_post args: uploaded_file
Optional Args: admin, admpass
Optional_post Args: session-id, login, password
Return MIME type: text/plain
Return Status Codes: 200, 400, 401
```

Used to modify the rights of the user specified as parameter.
This verb can only be used after the identification of a Registry administrator. The login and password of the Registry administrator are managed as parameters of the session and used to verify that only authorised users access this method. If a session has not yet been initialised the verb can be used specifying the login and password of a Registry administrator as *optional arguments*. In this way, the verb is usable either by the Registry GUI, that creates a session with the Registry Engine, or by other services.
All rights-configuration information is passed from the Registry UI to the Registry engine as an HTTP post request of an xml file.
Returns 200 if the rights modification was successful, an error code (400 or 401) otherwise.

Example Request:
/OLP/Registry/1.0/EditUserRights/testuser/%s

AddGroup

```
     Version: 1.0
     Fixed_post args: owner, password
     Optional_post Args: uploaded_file
     Return MIME type: text/plain
     Return Status Codes: 200, 400
```

Creates a new group in the Registry. The specified user is set as the owner of the new group and is the only user present in the group.
The group settings are transmitted as an HTTP POST request of an XML file. Returns 200 if the group is successful created, the error code 400 otherwise.

Example Request:
/OLP/Registry/1.0/AddGroup/%s/%s/%s

DisplayGroupInfo

```
     Version: 1.0
     Fixed args: groupname
     Optional Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

Returns a structured response that contains all settings of the given group:
- groupname, group's ID in the registry,
- description, short description of the group,
- owner, login of the group's owner,
- comment, any text.

Example Request:
/OLP/Registry/1.0/DisplayGroupInfo/group

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<DisplayGroup version="1.0">
        <groupname>developers</groupname>
        <description>System Developers</description>
        <owner>devadmin</owner>
        <comment></comment>
</DisplayGroup>
```

ModifyGroupInfo

```
     Version: 1.0
     Fixed_post args: groupname, owner, password
     Optional_post Args: uploaded_file
     Return MIME type: text/plain
     Return Status Codes: 200, 400
```

Changes settings of the given group.
The Registry service verifies that the user specified (*owner)* is the owner of the group specified (*groupname)* because only the owner of a group can modify the group settings.
All fixed and optional arguments are transmitted as an HTTP POST request.
Returns 200 if all required settings can be changed, the error code 400 otherwise.

Example Request:
/OLP/Registry/1.0/ModifyGroupInfo/%s/%s/%s%s

DeleteGroup

```
Version: 1.0
Fixed_post args: groupname, owner, password
Optional Args: none
Return MIME type: text/plain
Return Status Codes: 200, 400
```

Deletes a group. Its members remain as registered users, but their membership in this group is removed.
The Registry service verifies that the user specified (*owner)* is the owner of the group specified (*groupname)* because only the owner of a group can request the group deletion.
All fixed arguments are transmitted as an HTTP POST request.
Returns 200 if the deletion was successful, the error code 400 otherwise.

Example Request:
/OLP/Registry/1.0/DeleteGroup/%s/%s/%s

AddUserToGroup

```
Version: 1.0
Fixed_post args: groupname, owner, password, login
Optional Args: none
Return MIME type: text/plain
Return Status Codes: 200, 400
```

All fixed arguments are transmitted as an HTTP POST request.
The Registry service verifies that the user specified (*owner)* is the owner of the group specified (*groupname)* because only the owner of a group can add a user to a group.
Returns 0 if the user is known to the Registry and could be added to the group, a negative number otherwise.

Example Request:
/OLP/Registry/1.0/AddUserToGroup/%s/%s/%s/%s

AuthenticateGroupUser

```
Version: 1.0
Fixed args: groupname, login
Optional Args: none
Return MIME type: text/plain
Return Status Codes: 200, 400
```

Returns 200 if the user is a member of the given group, the error code otherwise.

Example Request:
/OLP/Registry/1.0/AuthenticateGroupUser/group/user

DeleteUserFromGroup

```
Version: 1.0
Fixed_post args: groupname, owner, password, login
Optional Args: none
Return MIME type: text/plain
Return Status Codes: 200, 400
```

Removes a user from the given group.

The Registry service verifies that the user specified (*owner)* is the owner of the group specified (*groupname)* because only the owner of a group can add a new user *(login)* to a group.
All fixed arguments are transmitted as an HTTP POST request.
Returns 200 if the user could be deleted, the error code otherwise.

Example Request:
/OLP/Registry/1.0/DeleteUserFromGroup/%s/%s/%s/%s

ListGroups

```
Version: 1.0
Fixed args: none
Optional Args: format
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Returns a structured response that contains a list in which each element of the list provides information about a public group from the Registry. *Format* can be either 'short', in which case the service returns only groupname for each group, or 'long', which returns all available items as specified in the DisplayGroup verb. The default value for format is 'short'.

Example Request:
/OLP/Registry/1.0/ListGroups?format=long

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<ListGroups version="1.0" format="long">
        <group>
          <groupname>developers</groupname>
          <description>
            System Developers
          </description>
          <owner>devadmin</owner>
          <comment></comment>
        </group>
        <group>
          ...
        </group>
</ListGroups>
```

ListGroupUsers

```
Version: 1.0
Fixed args: groupname
Optional Args: format
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Returns a structured response that contains a list in which each element of the list provides information on one member of the given group. *Format* can be either 'short', in which case the service only returns the login for each user, or 'long', which returns all available items as specified in the DisplayUser verb. The default value for format is 'short'.

Example Request:
/OLP/Registry/1.0/ListGroupUsers/group?format=long

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<ListGroupUsers version="1.0" group="developers" format="long">
        <user>
          <login>devadmin</login>
          <full>Developers' Administrator</full>
          <email>devadmin@scholnet.org</email>
          <institution>Scholnet</institution>
          <address>
            wherever all admins live 42
          </address>
          <phone>+9999999999</phone>
          <comment></comment>
        </user>
        <user>
          ...
        </user>
</ListGroupUsers>
```

ListUserGroups

```
Version: 1.0
Fixed args: login
Optional Args: format
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Returns a structured response that contains a list in which each element of the list provides information on one group that contains the given user as a member. *Format* can be either 'short', in which case the service only returns the groupname for each group, or 'long', which returns all available items as specified in the DisplayGroup verb. The default value for format is 'short'.

Example Request:
/OLP/Registry/1.0/ListUserGroups/user?format=long

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListUserGroups version="1.0" user="devadmin" format="long">
        <group>
          <groupname value="developers" />
          <description>
            System Developers
          </description>
          <owner value="devadmin" />
           <public value="1" />
          <comment>just a comment</comment>
        </group>
        <group>
          ...
        </group>
</ListUserGroups>
```

12. COLLECTION SERVICE

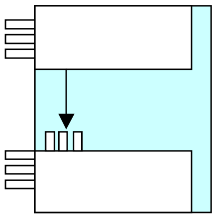
The Collection Service (CS) provides a virtual organisation of the documents stored in the repositories. It supplies the information necessary to manage this aggregation of virtual documents. This information is used by the other services in order to handle the objects in the collection objects so that, for example, the Query Mediator can perform a query on a given collection, or the Browse can perform a browsing on a collection, and so on. It can be replicated on multiple servers.

Collections are managed by one or more administrators. The administrators are registered users that have administration rights on the collection service, i.e. they can create, delete, and edit the collection metadata, creating virtual views of the information space. Theser rights are set by the Registry Administrator.

In order to automatically pre-organise the documents stored in the distributed repositories, the Collection Service exploits the repository set concept. A set is an administrator-defined subset of the Repository that is described by a name and a textual description. At the service start-up time (and then periodically) the CS dynamically pulls information about sets from each Repository Service instance and associates a collection with each discovered set. These collections are called *default collections* and can be used by the CS administrators to create other virtual views of the information space.

The CS is a replicated service. Multiple instances of the CS can be defined in the environment. The Meta Service maintains the address of each instance and defines one of these as the master CS and all others as slave CS. When a slave CS adds, deletes or edits the collection metadata for a collection, it sends a notification to the master CS that registers the event and updates the information for that collection. Periodically, each slave CS harvests the collection's metadata list from the master CS and updates its state.
The presence of multiple instances of the CS increases fault tolerance, reduces the overload of each instance, and makes it possible to dynamically reorganise the environment when a server hosting the CS is not reachable. Note that the Meta Service can dynamically change the status of a Collection Service, moving it from slave to master and vice versa. The master CS must verify that the CS candidate to the master status has updated the information before changing its state to slave in order to preserve the CS state integrity. If the master candidate has not yet been updated the CS invokes Synchronise which then forces the synchronisation between two CSs.

The Collection Service is logically divided in two components as reported below:



Communication with the Collection Service takes place via the OpenDLib Protocol (OLP). Note that other services can call either the Collection GUI verbs or the Collection Engine verbs in accordance with the service specification reported in this section. The *CS GUI* and *CS Engine* communicate via internal API. This architectural choice makes it possible:

- to use the Collection Service functionality with a third party user interface;
- to easily change the CS GUI to fit the specific needs.

12.1 *STATE*

This section presents the data structure, its abstract type, and a brief textual description for each object managed by the Collection Service.

Collection

A Collection is a virtual view of the information space, i.e. any collection may be perceived as a subset of documents published in the environment. The documents belonging to the collection are dynamically identified. Each collection has a unique *CollectionId* and the following collection metadata, in accordance with the DTD defined in the Appendix G.

- *Name*: the name of the collection
- *Description*: the textual description of the collection
- *Subject*: a list of free text optionals that specify the collection subject
- *Owner*: the login of the creator of the collection
- *Filtering Condition*: a set of pairs (Authority, Condition) with the following meaning:
  - *Authority*: authority that publishes documents in the collection
  - *Condition:* filter to select the documents of the collection among those published by the authority
- *Services*: a set of service descriptions. Each service description reports the name of the service and its verbs (with the specification of the version) that can be used on the collection.
- *Parent Collection*: the parent collection CollectionId. This information is used to build a hierarchy between collections in order to present them in a more usable and meaningful way to the end-user.

Note that the information about the available verbs of a service will be used by other services in order to present the end-user with only that functionality supported by authorities associated with the collection. For example, let us suppose that certain authorities (identified generically as Ax) indexed by certain specific index servers support a very complex query, named A, that is not supported by other authorities. The A query form is presented as available functionality to the end-user only when he/she selects a collection based on authorities extracted by the Ax group mentioned above. The same query capability is not visible to the user when he selects a collection on authorities outside Ax.

Here below is an example of the collection metadata managed by the CS.

```
<collection name="……." description="…….." subject= "……." id="…">
   <owner>
      <login>….</login>
   </owner>
   <filtering-condition>
         <pair>
            <authorities>
               <authority name="…." />
               ……
            </authorities>
            <condition value="…." />
         </pair>
      ………..
   </filtering-condition>
   <services>
      <service name="……">
         <verbs name="……" version="……">
         …….
```

```
        </service>
        ………..
     </services>
    <parent-collection id="…." />
</collection>
```

12.2 *COLLECTION VERBS*

In this section, we present the set of Collection verbs. This set is divided into three sub-sets: Service Information Verb, Graphical User Interface Verb, and Service Specific Verb, as reported in the following table:

Service Information Verb	GUI Verb	Service Specific Verb
Identify	AdminLoginForm	AdminLogin
ListVerbs		CreateDefaultCollection
DescribeVerb	CreateCollectionForm	CreateCollection
	EditCollectionForm	EditCollection
	DeleteCollectionForm	DeleteCollection
	ListCollectionsForm	ListCollections
	ShowCollectionMetadata	GetCollectionMetaData
		Synchronise
		SynchroniseCollection

Any GUI verb returns an html page that is designed for rendering by a browser.

Identify

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name and other information about the service.

Example Request:
/OLP/Collection/1.0/Identify

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<Identify version="1.0">
    <serviceName>Experimental OLP Collection Server</serviceName >
    <olp_base_url>http://labserv.iei.pi.cnr.it:8230</olp_base_url>
    <metaServiceURL>http://labserv.iei.pi.cnr.it:8221</metaServiceURL>
    <textualDescription></textualDescription>
    <protocolVersion>1.0</protocolVersion>
    <adminEmail>pagano@iei.pi.cnr.it</adminEmail>
    <verbsInfo>
        <verbsSupported>
            <description>   </description>
            <olp_base_url>http://labserv.iei.pi.cnr.it:8230/OLP/Collection/1.0/ListVerbs</olp_base_url>
        </verbsSupported>
        <verbsDescription>
            <description>   </description>
            <olp_base_url>http://…:8230/OLP/Collection/1.0/DescribeVerb/DescribeVerb</olp_base_url>
        </verbsDescription>
    </verbsInfo>
    <submissionProcedure>
```

```
        <description>The service supports public management of the collection. The authorised user can
                create, edit, and delete own collection.
        </description>
          <olp_base_url>http://…/OLP/Collection/1.0/AdminLoginForm</olp_base_url>
    </submissionProcedure>
    <harvestInformation>
        <description>
                The collections list is available
          </description>
          <olp_base_url>http://…:8230/OLP/Collection/1.0/ListCollections</olp_base_url>
    </harvestInformation>
    <useRestrictions>Terms and conditions are …...</useRestrictions>
    <contentInfo>
        <collectionProfile>
              <description>
                A metadata description is associated with every collection in accordance with the OpenDLib
                CollectionProfile format.
              </description>
              <olp_base_url>http://labserv.iei.pi.cnr.it:8119/OLP/htdocs/collectionprofile.html</olp_base_url>
              <olp_dtd_url>http://labserv.iei.pi.cnr.it:8119/OLP/htdocs/DTD/collectionprofile.dtd</olp_dtd_url>
        </collectionProfile>
    </contentInfo>
</Identify>
```

ListVerbs

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name of the verbs defined by this service.

Example Request:
/OLP/Collection/1.0/ListVerbs

DescribeVerb

```
Version: 1.0
Fixed args: verb
Optional Args: version
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on a version of the specified verb that is supported by this service. The following information can be provided at the verb and version level:

- *description*, description of the verb or a specific version
- *note*, information pertaining to the verb or a specific version

Each element of the list contains the following information:

- *version,* number of the verb.
- *arguments*, list of the names of the fixed and optional arguments, if any, that are accepted by the verb in that version.
- *example* template of request to this Collection, with fixed arguments indicated in brackets
- *returns*, optional, contains information about response format.

Note that a service may implement more than one version of a verb.

Example Request:
/OLP/Collection/1.0/DescribeVerb/AdminLoginForm

AdminLogin

```
Version: 1.0
Fixed post: none
Fixed_post args: login, password
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Allows to verify the identification of an administrator. The login and password are passed as HTTP POST request.

The method verifies that the login and password are correct and returns the list of the collections that the identified user is authorised to administrate.

Example Request:
/OLP/Collection/1.0/AdminLogin
Example Response

```
<?xml version="1.0" encoding="UTF-8"?>
<AdminLogin version="1.0">
   <session-id value=""/>
   <login>test</login>
   <collections>
      <collection name="……." description="…….." subject= "……." id="…">
         <collection name="……." description="…….." subject= "……." id="…"/>
      </collection>
      ………………
   </collections>
 </AdminLogin>
```

CreateDefaultCollections

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/plain
Return Status Codes: 200, 400, 501
```

Allows to create the *default collections* of the environment. The CS harvest information from each repository in order to discover the available sets. For each set, the CS creates a collection such that:

- The name of the collection is the name of the set;
- The description is the display field associated with the set;
- The authorities is the list of all authorities (also if they are managed by different repositories) able to publish documents in the set;
- The condition for each authority is: "setspec=*set-name*";
- The owner is the special flag "Collection Service";
- At present the Subject is empty. In the future we would like to automatically associated subjects to the default collections by analysing the documents that belong to them.
- The parent collection is calculated automatically using the structured information of the name, formally expressed in accordance with the *partiotionspec* definition rule (see Repository State paragraph).
- The services reported are the available verbs for searching and browsing the authorities specified.

The master CS automatically invokes the CreateDefaultCollection at service start-up time. Each slave CS at service start-up time requires the list of available collections from the master CS.
Note that the default collections are periodically updated because each repository can change the list of available sets or new repositories can be dynamically added to the environment.

Returns 200 if the request can be served, the error code 400 otherwise.

Example Request:
/OLP/Collection/1.0/CreateDefaultCollection

CreateCollection

```
     Version: 1.0
     Fixed args: none
     Fixed_post args: collection-specification
     Optional Args: session-id
     Optional_post Args: login, password
     Return MIME type: text/plain
     Return Status Codes: 200, 400, 501
```

Allows to create collections in the environment. The collection-specification is passed as XML file in POST mode. This specification contains two parts. The first part includes metadata that describes the collection: the name, description, and subject of the collection, its owner, its parent collection, and the list of services. The second part describes the membership condition and the available services.
Note that the collection filtering condition is built as the combination of the filtering condition of its parent collection, if this is specified, and the membership condition specified explicitly.
The membership condition is formulated according to the following BNF grammar:
Membership-Condition ::= Condition | Part
Part ::= Pair | Pair Part
Pair ::= Auths *and* Condition | Auths
Filtering-Condition ::= Leaf | Condition
Condition ::= Leaf Operator Leaf | Leaf Operator Condition | Condition Operator Leaf | Condition Operator Condition
Leaf ::= Attr Rel-Op *Value*
Operator::= *and* | *or*
Rel-Op ::= = | > | < | …
Auths ::= *Authories = (list of authorities)*
Attr ::= *Olap field*

Multiple conditions can be associated with multiple authorities set. If the user does not specify the list of authorities, all authorities are assumed as default and only a single condition can be specified.

An example of the collection-specification is reported below:

```
<?xml version="1.0" encoding="UTF-8"?>
<collection name="" description="" subject= "" id="">
    <owner>
        <login></login>
        <passwd></passwd>
    </owner>
    <services>
        <service name="search">
            <verb name="SearchFielded">
                <metadata_format name="olms">
                    <filtering name="dc:language" value="en">
                        <fields>
                            <field name="dc:creator"/>
```

```
                                ....
                            </fields>
                        </filtering>
                    </metadata_format>
                </verb>
            </service>
        </services>
        <parent-collection id="" />
        <membership meta-format="olms" filtering_name="dc_language" filtering_value="en">
            <pair>
                <authorities>
                    <authority name="ercim.cnr.iei"/>
                    <authority name="ercim.cnr.cnuce"/>
                </authorities>
                <filtering-condition>
                    <condition>
                        <leaf attribute="dc:creator" value="pagano" rel-op=""/>
                        <operator name="and"/>
                        <leaf attribute="dc:creator" value="castelli" rel-op=""/>
                    </condition>
                    <operator name="or"/>
                    <leaf attribute="dc:subject" value="digital library" rel-op=""/>
                </filtering-condition>
            </pair>
            <pair>
                <authorities>
                    <authority name="ercim.cnr.iat"/>
                </authorities>
                <filtering-condition>
                    <condition>
                        <leaf attribute="dc:creator" value="buzzi" rel-op=""/>
                        <operator name="or"/>
                        <leaf attribute="dc:subject" value="digital library" rel-op=""/>
                    </condition>
                </filtering-condition>
            </pair>
        </membership>
</collection>
```

Note that in the *service* section of the collection-specification for every service specification not explicitly reported, the default configuration, reported by the QM ListSearchMethods verb and by the Browse ListBrowsableFields verb, is assumed as default. This means that if, in the specification of the collection *A*, is not reported "<service name="search">", the search service is not usable to find documents belonging to the collection. If is reported "<service name="search" />", all search functionalities will be available on that collection. If is reported: "<service name="search"><verb name="SearchFielded"/></service>", only the SearchFielded method will be available on the collection. If is reported: "<service name="search"><verb name="SearchFielded"><metadata_format ="olms"/></verb></service>", only the SearchFielded method restricted to the *OLMS* metadata format will be available on the collection. The same rules apply to the "<service name="browse">" section, and, as for the search section, for each sub-section not explicitly reported.

The filtering condition allows to select which documents, published by specific authorities, belong to a collection.

The CS creates a CollectionId automatically and uses the collection-specification in order to build the collection metadata. At present the Membership-Condition includes information about the authorities that stores documents about the collection. If the authorities are not specified, the CS assumes that the collection documents can be published in one of the overall set of authorities.

Techniques that allow the CS to select automatically the authorities that publish documents compliant with the membership condition can be taken into account. The CS has been designed to easily permit this enhancement.

This verb can be invoked only after the identification of the administrator. The login and password of the administrator are managed as parameters of the session and allow to verify that only authorised users can access this method.
If a session is not yet initialised, the verb is usable by specifying the login and password of the Collection administrator. In this way the verb can be invoked either by the Collection GUI, that creates a session with the Collection Engine, or by the other services.

Returns 200 if the request can be served, the error code 400 otherwise.

Example Request:
/OLP/Collection/1.0/CreateCollection

EditCollection

```
Version: 1.0
Fixed args: CollectionId
Fixed_post args: collection-metadata
Optional Args: session-id
Optional_post Args: login, password
Return MIME type: text/plain
Return Status Codes: 200, 400, 501
```

Allows to edit the description, subject, parent collection, and services of an exiting collection. Only the owner of a collection can modify these collection metadata.
In order to preserve the database integrity the filtering condition cannot be modified using the EditCollection service request. The modification of the settings specified in the filtering condition might identify a completely different set of documents, thus changing completely the semantics of the collection. .

The CS uses the login and password (or the session-id) to verify that the user can edit and modify the specified collection.
The collection-metadata is passed as XML file in POST mode.

Returns 200 if the request can be served, the error code 400 otherwise.

Structure of the collection-metadata file passed to the Collection is as follow:

```
<collection name="……." description="…….." subject= "……." id="…">
   <owner>
      <login>….</login>
   </owner>
    <membership meta-format="" language="">
      <pair>
           <authorities>
               <authority name="ercim.cnr.iei"/>
               <authority name="ercim.cnr.cnuce"/>
           </authorities>
          <filtering-condition />
      </pair>
      ………..
   </filtering-condition>
   <services>
      <service name="……">
          <verbs name="……" version="……">
          …….
```

```
        </service>
        ………..
     </services>
    <parent-collection id="…." />
</collection>
```

Example Request:
/OLP/Collection/1.0/EditCollection/0000125

DeleteCollection

```
        Version: 1.0
        Fixed args: CollectionId
        Optional Args: session-id
        Optional_post Args: login, password
        Return MIME type: text/plain
        Return Status Codes: 200, 400, 501
```

Allows to delete a explicitly created collections, i.e. a collections that is not classified as default. The specified collection is removed from the list of available collections.
Before to proceed to the deletion the system checks that:

- The request is issued by the owner of the collection ;
- The specified collection is not classified as default collection. The default collection cannot be removed.

This verb is usable only after the identification of the administrator. The login and password of the administrator are managed as parameters of the session and allow to verify that only authorised users can access this verb.
If a session is not yet initialised, the verb is usable by specifying the login and password of the Collection administrator as *optional arguments*. In this way the verb is usable either by the Collection GUI, that creates a session with the Collection Engine, or by the other services.

Returns 200 if the request can be served, the error code 400 otherwise.

Example Request:
/OLP/Collection/1.0/DeleteCollection/0000125

ListCollections

```
        Version: 1.0
        Fixed args: none
        Optional Args: owner
        Return MIME type: text/xml
        Return Status Codes: 200, 400, 501
```

Returns a structured response containing the Name, Description, Subject and CollectionId of the collections defined by this service.
If the owner is specified the CS return only the list of collections owned by the specified user.

Note that the *parent-collection* attribute is used to build a structured list where a collection child is included in the tree of the collection father.

Example Request:
/OLP/Collection/1.0/ListCollections

Example Response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ListCollections version="1.0">
   <collection name="Computer Science Publications" id="0000015" description="Scholnet Computer
Science Publications" subject="computer science">
      <collection name="Operating System" id="0000024" description="Operating System Technical
Reports" subject ="Operating System"/>
      <collection name="Database" id="0000024" description="Scholnet Computer Science Technical
Reports" subject ="Database"/>
   </collection>
   <collection>
   ….
   </collection>
</ListCollections>
```

GetCollectionMetadata

```
Version: 1.0
Fixed args: CollectionId
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing all information about a specific collection. This information is used, for example, by the Query Mediator to transform a query on a collection in a query understandable by an Index Service.

Example Request:
/OLP/Collection/1.0/GetCollectionMetadata

Example Response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<collection name="……." description="…….." subject= "……." id="…">
   <owner>
      <login>….</login>
   </owner>
    <filtering-condition meta-format="" language="">
      <pair>
         <authorities>
            <authority name="ercim.cnr.iei"/>
            <authority name="ercim.cnr.cnuce"/>
         </authorities>
         <filtering-condition />
      </pair>
      ………..
   </filtering-condition>
   <services>
      <service name="……">
         <verbs name="……" version="……">
         …….
      </service>
      ………..
    </services>
   <parent-collection id="…." />
</collection>
```

Synchronise

```
Version: 1.0
```

```
     Fixed args: host, port
     Optional Args: none
     Return MIME type: text/plain
     Return Status Codes: 200, 400, 501
```

The verb is used to synchronise two or more CSs.

This method is used by a Meta Service before changing the state of CS from master to slave, or periodically by the Master Collection Service to update its state.

The methods is managed by the new master CS as follows:
1. Set a lock that interrupts any collection service request. The pending requests are completed and the new ones are refused sending a special error code that allows the client to manage the temporary interruption of the service.
2. Send a Synchronise request to all slaves CSs.
3. Require the list of available collections from any slave CS, using SynchroniseCollection verb.
4. Update the set of collection metadata.
5. Remove the lock on SynchroniseCollection request.
6. Wait that any slave CS updates its collection metadata via SynchroniseCollection.
7. Remove the lock on all methods.

The methods is managed by a slave CS as follow:
1. Set a lock that interrupts any collection service request, except for SynchroniseCollection. The pending requests are completed and the new ones are refused.
2. Wait until the master CS call the SynchroniseCollection verb.
3. Send a SynchroniseCollection request to the new master CS in order to download the list of available collections.
4. Update the set of collection metadata.
5. Remove the lock on the collection requests.

A CS accepts the Synchronise request only if the client address is known as a CS address or a Meta address.

Example Request:
/OLP/Collection/1.0/Synchronise/labserv.iei.pi.cnr.it/8031

SynchroniseCollection

```
     Version: 1.0
     Fixed args: none
     Optional Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400, 501
```

Returns a structured response containing all information about the handled collections set.
A CS accepts the SynchroniseCollection request only if the client address is known as a CS address.

Example Request:
/OLP/Collection/1.0/SynchroniseCollection

AdminLoginForm

```
     Version: 1.0
     Fixed args: none
     Optional_post Args: login, password
     Return MIME type: text/html
```

```
    Return Status Codes: 200, 400
```

Returns the page that allows the administrator to log in the administration environment.
If the login and password are specified as optional arguments, passed as HTTP POST request, the Collection GUI service starts the session without showing the form in which these values can be entered. In this way the verb can be called also by other services that have already authenticated the user as Collection administrator.
After the identification of the Collection administrator the verb shows the administration home environment. Using this environment the administrator can access all the service functionality.

Example Request:
/OLP/Collection/1.0/AdminLoginForm

ListCollectionsForm

```
      Version: 1.0
      Fixed args: none
      Optional Args: user
      Return MIME type: text/html
      Return Status Codes: 200, 400, 501
```

Returns the page that shows the structured collection list containing the Name, Description, Subject and CollectionId of the collections managed by this service.
If the user is specified only the list of collections owned by the specified user is shown.

Example Request:
/OLP/Collection/1.0/ListCollectionsForm

ShowCollectionMetadata

```
      Version: 1.0
      Fixed args: CollectionId
      Optional Args: none
      Return MIME type: text/html
      Return Status Codes: 200, 400, 501
```

Returns the page that shows the structured collection metadata containing all information about a specific collection. This information is analysed, for example, by a collection administrator in order to create a refinement of an existing collection.

Example Request:
/OLP/Collection/1.0/ShowCollectionMetadata

CreateCollectionForm

```
    Version: 1.0
    Fixed args: none
    Optional Args: none
    Return MIME type: text/html
    Return Status Codes: 200, 400
```

Returns the page that allows the administrator to fill in all information about a collection.
The administrator has to submit the following information:

- the name, the subject and the description
- the membership condition. It is automatically composed by the service in accordance with the steps performed by the user. He/she can specify:
  - The collection constraints using all OLAP fields.
  - The collection as composition of other existing collections.

- The collection as refinement of an exiting collection

This verb is usable only after the identification of the administrator. The login and password of the administrator are managed as parameters of the session and allow to verify that only authorised user can access this method.
If a session is not yet initialised the verb automatically calls the AdminLoginForm.

Example Request:
/OLP/Collection/1.0/CreateCollectionForm

EditCollectionForm

```
Version: 1.0
Fixed args: step
Optional Args: none
Return MIME type: text/html
Return Status Codes: 200, 400
```

Returns the page that allows the administrator to change the metadata associated to an existing collection.
The Collection Service in multiple steps manages the method.
After the administrator has submitted the identification and password, he/she can select the collections that he/she wants to change. Note that an administrator can only ask to change his/her own collections.In the second step the user can modify the following information:

- the showed description and subject
- the list of available services and verbs.

Note that the membership condition of the collection cannot be changed because this operation corresponds to the identification of a new set of documents.. If the administrator wants to change a filtering condition he/she has to delete the exiting collection and to build a new one.

The structured information is validated in accordance with the collection metadata format.

This verb is usable only after the identification of the administrator. The login and password of the administrator are managed as parameters of the session and allow to verify that only authorised user can access this method.
If a session is not yet initialised the verb automatically calls the AdminLoginForm.

Example Request:
/OLP/Collection/1.0/EditCollectionForm/1

DeleteCollectionForm

```
Version: 1.0
Fixed args: step
Optional Args: none
Return MIME type: text/html
Return Status Codes: 200, 400
```

Returns the page that allows the administrator to request the deletion of a document. A delete request deletes all information about the collection. The delete request is supported but it is deprecated.

After the administrator has submitted the identification and password, he/she can select the collections that he/she wants to delete. Note that an administrator can ask only for the deletion of that collection of which he/she is owner.

This verb is usable only after the identification of the administrator. The login and password of the administrator are managed as parameters of the session and allow to verify that only authorised user can access this method.
If a session is not yet initialised the verb automatically calls the AdminLoginForm.

Example Request:
/OLP/Collection/1.0/DeleteCollectionForm/1

13. BROWSE SERVICE

The Browse Service (BS) supports the construction of browsing indexes and the actual browsing of those indexes on library contents.
It receives browsing requests and returns structured list of document metadata, each of which contains a link to the document itself.
The formats of the metadata managed by the BS, the indexed fields list and of the result metadata are parametric and can be easily changed. The BS harvests these configuration parameters from the Meta Service if it is configured in networked modality, or read them from its local configuration files if it is configured in standalone modality.
In the Scholnet environment these formats are subsets of the OpenDLib Application Profile, i.e. any metadata format disseminated by the Repository Service is a potential candidate.
In order to build the browsing indexes, the service periodically harvests the available metadata from each repository, and updates its knowledge base.

The communication with the Browse Service takes place via the OpenDLib Protocol (OLP).

13.1 *STATE*

This section presents the data structure, its abstract type, and a brief textual description for each object managed by the Browse Service.

Bibliographic Record

The Browse Service manages bibliographic records in accordance with the specified metadata format.
A bibliographic metadata is named by a handle, which is a kind of URN. Unlike a URL, a handle is location independent.
A handle has two parts, a *naming authority,* and a *string*. It is written with these two parts separated by a slash, for example CNR.IEI /doc1. The character set for handles used in OLP is restricted to alphanumeric characters, underscore, period, and hyphen (except for the slash separator). Case is not significant in handles.

Browsable Fields

The Browse Service will index a subset of the fields included in the specified metadata format. This subset is determined analysing the configuration parameters at service start-up time.

Result Formats

The Browse Service will return list of documents where each element of the list contains the fields specified in one of the available result formats. These formats are determined analysing the configuration parameters at service start-up time.

13.2 *BROWSE VERBS*

In this section, we present the set of the Browse verbs. This set is divided into three sub-sets: Service Information Verb, Service Descriptive Verb, and Service Specific Verb as reported in the following table:

Service Information Verb	Service Descriptive Verb	Service Specific Verb
Identify	ListBrowsableFields	Browse
ListVerbs	ListResultFormats	BrowseUpdate
DescribeVerb		

Identify

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name and other information about the service.

Example Request:
/OLP/Browse/1.0/Identify

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<Identify version="1.0">
   <serviceName>Experimental OLP Browse Server</serviceName >
   <olp_base_url>http://labserv.iei.pi.cnr.it:8230</olp_base_url>
   <metaServiceURL>http://labserv.iei.pi.cnr.it:8221</metaServiceURL>
   <textualDescription></textualDescription>
   <protocolVersion>1.0</protocolVersion>
   <adminEmail>pagano@iei.pi.cnr.it</adminEmail>
   <verbsInfo>
       <verbsSupported>
           <description>   </description>
           <olp_base_url>http://labserv.iei.pi.cnr.it:8230/OLP/Browse/1.0/ListVerbs</olp_base_url>
       </verbsSupported>
       <verbsDescription>
           <description>   </description>
           <olp_base_url>http://…:8230/OLP/Browse/1.0/DescribeVerb/DescribeVerb</olp_base_url>
       </verbsDescription>
   </verbsInfo>
   <submissionProcedure>
       <description>The service does not support submission.
       </description>
         <olp_base_url> </olp_base_url>
   </submissionProcedure>
   <harvestInformation>
       <description>
             No information is available
         </description>
         <olp_base_url> </olp_base_url>
   </harvestInformation>
   <useRestrictions>Terms and conditions are …...</useRestrictions>
   <contentInfo>
       <browsableFields>
           <description>
             The set of fields that can be used to browse the set of documents is available.
           </description>
           <olp_base_url> http://…:8230/OLP/Browse/1.0/ListBrowsableFields</olp_base_url>
       </browsableFields>
       <resultFormats>
           <description>
             The result formats used by the service are parametric. The pertinent set of result formats is
             available.
           </description>
           <olp_base_url> http://…:8230/OLP/Browse/1.0/ListResultFormats</olp_base_url>
       </resultFormats>
   </contentInfo>
</Identify>
```

ListVerbs

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name of the verbs defined by this service.

Example Request:
/OLP/Browse/1.0/ListVerbs

DescribeVerb

```
Version: 1.0
Fixed args: verb
Optional Args: version
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on a version of the specified verb that is supported by this service. The following information may be provided at the verb or version level:

- *description*, description of the verb or a specific version
- *note*, information pertaining to the verb or a specific version

Each element of the list contains the following information:

- *version* number of the verb.
- *arguments*, a list of the names of the fixed and optional arguments, if any, accepted by the verb in that version.
- *example* template of request to this Collection, with fixed arguments indicated in brackets
- *returns*, optional, contains information about response format.

Note that a service may implement more than one version of a verb.

Example Request:
/OLP/Browse/1.0/DescribeVerb/Browse

ListBrowsableFields

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on a field available for browsing the library contents.
The available fields are identified at Browse service start-up time, analysing the configuration parameters harvested from the Meta (BS configured as networked) or read from local files (BS configured as standalone).

Example Request:
/OLP/Browse/1.0/ListBrowsableFields

Example Response:
<?xml version="1.0" encoding="UTF-8"?>

```
<ListBrowsableFields version="1.0">
    <meta-format name="olms" uri="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/">
        <field name="creator">
            <type>string</type>
        </field>
        <field name="date">
            <type>iso8610</type>
            <start_value>1995-08-01</start_value>
            <end_value>20021024</end_value>
        </field>
    </meta-format>
</ListBrowsableFields>
```

ListResultFormats

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on the format of a browse results request on the library contents.
The available formats are identified at Browse service start-up time, analysing the configuration parameters harvested from the Meta (BS configured as networked) or read from local files (BS configured as standalone). One of these formats must be indicated as default, otherwise the first one is assumed.
The name of these formats specified in the response can be used as optional arguments in the Browse request in order to select the format of the result.

Example Request:
/OLP/Browse/1.0/ListResultFormats

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListResultFormats version="1.0">
    <meta-format name="olms" uri="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/">
        <format name="short" default='yes'>
            <field>dc:title</field>
            <field>dc:date</field>
        </format>
        <format name="long">
                <field>dc:creator</field>
            <field>dc:title</field>
            <field>dc:date</field>
            <field>dc:subject</field>
        </format>
    </meta-format>
</ListResultFormats>
```

Browse

```
Version: 1.0
Fixed args: field
Optional Args: range, authority, collection, format
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on a document.

The meaning of the arguments is as follows:

- *field*. Specifies the field on which the browse request is performed.
- *range*. Specifies a range of values for which browse information should be returned.
- *authority*. If the authority option is specified, the results will be filtered so that documents returned indicate publications for the authorities specified. If no value is specified, all the authorities specified in the configuration parameters are assumed as default.
- *collection*. If collection option is specified, the results will be filtered so that documents returned indicate publications for the collections specified. If no value is specified, the all collections value is assumed as default. Note that if the collection option is specified, the *authority* option is ignored.
- *format*. Specifies the format of the browse results request. If no value is specified, the default result format value is assumed.

The name of the formats available can be selected from the list returned by the ListResultsFormat.

Example Request:
/OLP/Browse/1.0//Browse/creator/range=a-d&collection=information%20retrieval&format=short

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<Browse version="1.0">
   <document handle="ncstrl.scholnet.one.trs.good/2001-TR-009">
       <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
             xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
             xmlns:dc="http://purl.org/dc/elements/1.1/">
             <dc:title>A document test</dc:title>
             <dc:date>2001/06/12</dc:date>
       </olms:ol>
   </document>
</Browse>
```

BrowseUpdate

```
Version: 1.0
Fixed args: metadata-format
Optional Args: authority, force, verbose
Return MIME type: text/plain
Return Status Codes: 200, 400, 501
```

Used to update the browse database. This service request harvests the available new bibliographic records from remote Repository servers and adds these new document descriptions to the database.

The arguments are as follows:

- *metadata-format*. Specifies the metadata format harvested from the Repository. The metadata format managed by the Browse service is a parameter specified at service start-up time.
- *authority*. Specifies the authorities to be updated. If no value is specified, all authorities are assumed as default.
- *force*. Used to force the updating of all authorites without keeping into account the time-stamp of each authority. If no value is specified, only the authorities that have a time-stamp within the specific time (fixed as configuration parameter) will be updated.
- *verbose*. Forces the writing of a verbose log.

The following steps are performed:

1. Lock the browse database in order to update it.
2. Select the authorities that have to be updated.

3. Identify the Repositories that store the documents for those authorities.
4. Send a ListContents request to each Repository identified specifying the authority, the starting date, and the metadata-format.
5. Add bibliographic information for a document to the browse database. Note that the same document may be added more than once to account for changes to bibliographic records.
6. Unlock the database.

200 is returned if no error occurs, an error code (400 or 501) otherwise.

Example Request:
/OLP/Browse/1.0/BrowseUpdate/OLMS

14. INDEX SERVICE

The Index Service (IS) supports the construction of indexes on the contents of the library and the querying of those indexes.
It receives search requests and returns a structured list of document metadata, each of which contains a link to the document itself.
The formats of the metadata managed by the IS, of the indexed fields list and of the result metadata are parametric and can be easily changed. The IS harvests these configuration parameters from the Meta Service if it is configured in networked modality, or reads them from its local configuration files if it is configured in standalone modality.
In the Scholnet environment, these formats are subsets of the OpenDLib Application Profile, i.e. any metadata format disseminated by the Repository Service is a potential candidate.

The Index Service is also a distributed service that is driven by the Meta Service, i.e. the Meta Service assigns a set of authorities to each index server.
In order to build the indexes, the service periodically harvests the available metadata from a set of repository servers, and updates its knowledge base. The set of repository servers is identified using the authority-repository relation maintained by the Meta Service.

Communication with the Index Service occurs via the OpenDLib Protocol (OLP).

The IS is composed by a third party retrieval engine and by a gateway between the OpenDLib environment and this engine. These two modules communicate via internal API. Note that the OpenDLib gateway is completely parametric and that different retrieval engines can be used, for example, FreeWaisSf, Inquery, and Smart.

14.1 *STATE*

This section presents the data structure, its abstract type, and a brief textual description for each object managed by the Index Service.

Bibliographic Record

The Index Service manages bibliographic records in accordance with the metadata format specified.
A bibliographic metadata is named by a handle, which is a kind of URN. Unlike a URL, a handle is location independent.
A handle has two parts, a *naming authority,* and a *string*. It is written with these two parts separated by a slash, for example CNR.IEI /doc1. The character set for handles used in OLP is restricted to alphanumeric characters, underscore, period, and hyphen (except for the slash separator). Case is not significant in handles.

Indexed Fields

The Index Service will index a subset of the fields included in the metadata format specified. This subset is determined analysing the configuration parameters at service start-up time.

Result Formats

The Index Service will return a list of documents in which each element of the list contains the fields specified in one of the available result formats. These formats are determined by analysing the configuration parameters at service start-up time.

Language

Each instance of the IS is language dependent. The language is specified as a local configuration parameter. The language has to be specified because the lists of stop words, stemming rules, and many others rules used by the retrieval engine are specific for each language. For this reason, different servers will be instantiuated, each of which will able to manage different language.

14.2 *INDEX VERBS*

In this section, we present the set of the Index verbs. This set is divided into three sub-sets: Service Information Verb, Service Descriptive Verb, and Service Specific Verb, as reported in the following table:

Service Information Verb	Service Descriptive Verb	Service Specific Verb
Identify	ListIndexedFields	SearchFielded
ListVerbs	ListResultFormats	SearchSimple
DescribeVerb		SearchFeedback
		GetMetadataDocuments
		IndexUpdate

Identify

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name and other information about the service.

Example Request:
/OLP/Index/1.0/Identify

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<Identify version="1.0">
    <serviceName>Experimental OLP Index Server</serviceName >
    <olp_base_url>http://labserv.iei.pi.cnr.it:8230</olp_base_url>
    <metaServiceURL>http://labserv.iei.pi.cnr.it:8221</metaServiceURL>
    <textualDescription></textualDescription>
    <protocolVersion>1.0</protocolVersion>
    <adminEmail>pagano@iei.pi.cnr.it</adminEmail>
    <verbsInfo>
        <verbsSupported>
            <description>   </description>
            <olp_base_url>http://labserv.iei.pi.cnr.it:8230/OLP/Index/1.0/ListVerbs</olp_base_url>
        </verbsSupported>
        <verbsDescription>
            <description>   </description>
            <olp_base_url>http://…:8230/OLP/Index/1.0/DescribeVerb/DescribeVerb</olp_base_url>
        </verbsDescription>
    </verbsInfo>
    <submissionProcedure>
        <description>Submission procedure is not supported by this service.
        </description>
          <olp_base_url> </olp_base_url>
    </submissionProcedure>
    <harvestInformation>
        <description>
              No information is available
            </description>
            <olp_base_url> </olp_base_url>
    </harvestInformation>
    <useRestrictions>Terms and conditions are …...</useRestrictions>
    <contentInfo>
          <indexedFields>
                <description>
                    The set of fields that can be used to search the set of documents.
                </description>
                <olp_base_url> http://…:8230/OLP/Index/1.0/ListIndexedFields</olp_base_url>
          </indexedFields>
          <resultFormats>
                <description>
                    The result formats used by the service are parametric. The pertinent set of result formats is
                    available at the specified address.
                </description>
                <olp_base_url> http://…:8230/OLP/Index/1.0/ListResultFormats</olp_base_url>
          </resultFormats>
          <supportedLanguage>
                <description>
                    Each instance of the IS is language dependent. The language is specified as a local
                    configuration parameter. The language has to be specified because the list of stop words,
                    the stemming rules, and many others rules used by the retrieval engine are specific for each
                    language.
                </description>
                <olp_base_url> http://…:8230/OLP/htdocs/language.html</olp_base_url>
          </supportedLanguage>
    </contentInfo>
</Identify>
```

ListVerbs

```
    Version: 1.0
    Fixed args: none
    Optional Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name of the verbs defined by this service.

Example Request:
/OLP/Index/1.0/ListVerbs

DescribeVerb

```
    Version: 1.0
    Fixed args: verb
    Optional Args: version
    Return MIME type: text/xml
    Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on a version of the specified verb that is supported by this service. The following information may be provided at the verb or version level:

- *description*, description of the verb or a specific version
- *note*, information pertaining to the verb or a specific version

Each element of the list contains the following information:

- *version* number of the verb.
- *arguments*, a list of the names of the fixed and optional arguments, if any, accepted by the verb in that version.

- *example* template of request to this Collection, with fixed arguments indicated in brackets
- *returns*, optional, contains information about response format.

Note that a service may implement more than one version of a verb.

Example Request:
/OLP/Index/1.0/DescribeVerb/ListIndexedFields

ListIndexedFields

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on a field that can be used to search the library contents.
The available fields are identified at Index service start-up time, analysing the configuration parameters harvested from the Meta service (IS configured as networked) or read from local files (IS configured as standalone).
Note that this verb returns information on the IS and that different search methods can use different subsets of the overall set of indexed fields. Detailed information on a search verb can be retrieved using the DescribeVerb method.
Note also that, using the local configuration of the retrieval engine, the IS specifies the operators supported for each field.

Example Request:
/OLP/Index/1.0/ListIndexedFields

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListIndexedFields version="1.0">
  <meta-format name="olms" uri="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/">
    <field name="dc:creator">
      <type>string</type>
      <rel-op name="equal" symbol="=" position="infix"/>
      <rel-op name="like" symbol="=^" position="infix"/>
    </field>
    <field name="dc:date">
      <type>%4d/%2d/%d(%d)</type>
      <rel-op name="equal" symbol="=" position="infix"/>
      <rel-op name="greater than" symbol=">" position="infix"/>
    </field>
    <supported-operators>
      <operator value="and"/>
      <operator value="or"/>
    </supported-operators>
  </meta-format>
</ListIndexedFields>
```

ListResultFormats

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on the format of a search results request on the library contents.
The available formats are identified at service start-up time, analysing the configuration parameters harvested from the Meta service (IS configured as networked) or read from local files (IS configured as standalone). One of these formats must be indicated as default, otherwise the first in the response list is assumed.
The name of these formats can be used as optional arguments in the Search request to select the format of the result.

Example Request:
/OLP/Index/1.0/ListResultFormats

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<ListResultFormats version="1.0">
   <meta-format name=”olms” uri="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/">
      <format name=”short” default=’yes’>
      <field>dc:title</field>
      <field>dc:date</field>
      </format>
      <format name=”long”>
         <field>dc:creator</field>
      <field>dc:title</field>
      <field>dc:date</field>
      <field>dc:subject</field>
      </format>
   </meta-format>
</ListResultFormats>
```

SearchFielded

```
Version: 1.0
Fixed args: none
Fixed_post args: search-condition
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on and a link to a document. This list, called result-set, is uniquely named by the IS that is able to accept query refinements on it. To perform a refinement on the *result*-set, see the SearchFeedback verb: in this case the IS performs the search, specified by means of the search-condition argument, as a query on this subset of the overall digital library content. The result-set remains active for a given amount of time, and when this expires, error 400 is returned.

The search-condition is passed as an HTTP POST request of an XML string. This specification contains two parts. The first part includes metadata that describe the set of authorities on which to perform the query, the result format requested by the user, and the specification of the max number of documents returned in the format specified (called top_documents). For the other documents, that match the filtering condition but overcome the max number specified, only the document header will be returned. To retrieve these documents see GetMetadataDocuments verb. Note that if the top_documents is not specified, the default value is assumed, and if the “-1” value is specified all documents that match the request will be returned in the requested format.
The second part describes the filtering condition. Note that the filtering condition is only performed on the set of documents that belong to the authorities specified.

The search condition is formulated in accordance with the following BNF grammar:

```
Search-Condition ::= Filtering-Condition | Part1 Filtering-Condition
Part1 ::= Format Auths Top | Format Top| Auths Top
Format ::= Format = string
Top ::= top_documents = number
Auths ::= Authories = (list of authorities)
Filtering-Condition ::= Leaf | Condition
Condition ::= Leaf Operator Leaf | Leaf Operator Condition | Condition Operator Leaf |  Condition Operator
Condition
Leaf ::= Attr Rel-Op Value
Rel-Op ::= like | = | …
Operator ::= and | or | ….
Attr ::= indexed_field
```

Note that if the user does not specify a list of authorities, all authorities are assumed as default. Note also that if the user does not specify the response format the default one is assumed.

The structure of the search-specification is reported below:

```
<search-condition>
    <format name=" …"/>
    <top_documents value=" "/>
    <authorities>
        <authority name="…." />
        ……
    </authorities>
    <filtering-condition value=" ….">
         <condition>
              <leaf attribute="" value=" " rel-op="" side="left"/>
            <operator name=" "/>
            <condition side="right">
                <leaf attribute="" value="" rel-op="" side="left"/>
                 <operator name=" "/>
                <leaf attribute="" value="" rel-op="" side="right"/>
            </condition>
        </condition>
    </filtering-condition>
</search-condition>
```

The filtering condition is used to select which documents, published by a set of specific authorities, belong to the result-set.

At present, the search-condition includes information about the authorities that store documents relevant to the request of the client. If the authorities are not specified, the IS performs the query on all authorities.

The name of the formats available can be selected from the list returned by the ListResultsFormat.

Example Request:
/OLP/Index/1.0/SearchFielded/%s

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<SearchFielded version="1.0">
    <result-set name="21908123" count="2">
        <document handle="scholnet.csp/2001-TR-009" version="1" rank="0.434224342">
            <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
                xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
                xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:title>A document test</dc:title>
                <dc:date>2001/06/12</dc:date>
            </olms:ol>
        </document>
        <document handle="scholnet.trs/2001-TR-023" version="1" rank="0.2321431234">
            <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
                xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
                xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:title>A document test2</dc:title>
                <dc:date>2001/08/12</dc:date>
          </olms:ol>
        </document>
    </result-set>
</SearchFielded>
```

SearchSimple

```
    Version: 1.0
    Fixed args: none
    Fixed_post args: search-condition
    Optional Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on and a link to a document. This list, called result-set, is uniquely named by the IS that is able to accept query refinements on it. To perform a refinement on the *result*-set, see the SearchFeedback verb: in this case the IS performs the search, specified by means of the search-condition argument, as a query on this subset of the overall digital library content. The result-set remains active for a given amount of time, and when this expires, error 400 is returned..

The search-condition is passed as an HTTP POST request of an XML string. This specification contains two parts. The first part includes metadata that describe the set of authorities on which to perform the query, the result format requested by the user, and the specification of the max number of documents returned in the format specified (called top_documents). For the other documents, that match the filtering condition but overcome the max number specified, only the document header will be returned. To retrieve these documents see GetMetadataDocuments verb. Note that if the top_documents is not specified, the default value is assumed, and if the "-1" value is specified all documents will be returned in the requested format.

The second part describes the filtering condition. Using the SearchSimple, the request of the client does not specify the fields on which the search has to be performed, and the IS calculates the response as a result of a free text search.
Note that the filtering condition specified is evaluated only on the set of documents that belong to the authorities specified.

The search condition is formulated in accordance with the following BNF grammar:
Search-Condition ::= Filtering-Condition | Part1 Filtering-Condition
Part1 ::= Format Auths Top | Format Top| Auths Top
Format ::= *Format = string*
Top ::= *top_documents = number*
Auths ::= *Authories = (list of authorities)*
Filtering-Condition ::= *string*

Note that if the user does not specify a list of authorities, all authorities are assumed as default. Note also that if the user does not specify the response format the default one is assumed.

The structure of the search-specification is reported below:

```
<search-condition>
    <format name=" …"/>
    <top_documents value=" "/>
    <authorities>
        <authority name="…." />
        ……
    </authorities>
    <filtering-condition value=" …."/>
</search-condition>
```

The filtering condition makes it possible to select which documents, published by a set of specific authorities, belong to the result-set.

At present, the search-condition includes information about the authorities that store documents relevant to request of the client. If the authorities are not specified, the IS performs the query on all authorities. In the future, techniques that allow the IS to select automatically the authorities that publish documents compliant with the query-condition will be considered.

The name of the formats available can be selected from the list returned by the ListResultFormats.

Example Request:
/OLP/Index/1.0/SearchSimple/%s

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<SearchSimple version="1.0">
    <result-set name="21908123" count="2">
        <document handle="scholnet.csp/2001-TR-009" version="1" rank="0.434224342“>
            <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
                xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
                    xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:title>A document test</dc:title>
                <dc:date>2001/06/12</dc:date>
            </olms:ol>
        </document>
        <document handle="scholnet.trs/2001-TR-023" version="2" rank="0.234224342“>
            <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
                xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
                xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:title>A document test2</dc:title>
                <dc:date>2001/08/12</dc:date>
            </olms:ol>
        </document>
    </result-set>
</SearchSimple>
```

SearchFeedback

```
Version: 1.0
Fixed args: none
Fixed_post args: feedback-condition
Optional Args: none
```

```
    Return MIME type: text/xml
    Return Status Codes: 200, 400, 501
```

Allows to perform a refinement on a result set generated by a previous query. This method, using the relevant documents specified, performs an automatic query expansion in order to improve the Index capability to satisfy the user needs. The query expansion algorithm uses terms extracted from the set of fields specified by the Meta Service with the index configuration parameters.
Returns a structured response that contains a list in which each element of the list provides information on and a link to a document. This list, called result-set, is uniquely named by the IS that is able to accept query refinements on it. To perform a refinement on the *result*-set, the SearchFeedback verb can be recalled: in this case the IS performs the search as a query on this subset of the overall digital library content. The result-set remains active for a given amount of time, and when this expires, error 400 is returned.

The feedback-condition is passed as an HTTP POST request of an XML string. This specification contains two parts. The first part includes metadata that report the result set name requested by the user, and the specification of the max number of documents returned in the original format specified (called top_documents). For the other documents, that match the feedback condition but overcome the max number specified, only the document header will be returned. To retrieve these documents see GetMetadataDocuments verb. Note that if the top_documents is not specified, the default value is assumed, and if the "-1" value is specified all documents will be returned in the requested format. The second part of the condition includes the list of relevant documents.

The feedback condition is formulated in accordance with the following BNF grammar:
Feedback-Condition ::= Part1 Rel_Docs
Part1 ::= Result_Set Top | Result_Set
Result_Set ::= *result-set = string*
Top ::= *top_documents = number*
Rel_Docs ::= Doc | Doc Rel_Docs
Doc ::= *document = string*

The structure of the feedback condition is reported below :

```
<feedback-condition>
    <result-set name=""/>
    <top_documents value=""/>
    <rel_documents>
        <document handle="" version=""/>
        <document handle="" version=""/>
    </rel_documents>
</feedback-condition>
```

Example Request:
/OLP/Index/1.0/SearchFeedback/%s
Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<SearchFeedback version="1.0">
    <result-set name="21908123" count="3">
        <document handle="scholnet.csp/2001-TR-009" version="1" rank="0.434224342">
            <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
                xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
                    xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:title>A document test</dc:title>
                <dc:date>2001/06/12</dc:date>
            </olms:ol>
        </document>
        <document handle="scholnet.trs/2001-TR-023" version="2" rank="0.234224342">
```

```
            <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
                xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
                xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:title>A document test2</dc:title>
                <dc:date>2001/08/12</dc:date>
            </olms:ol>
        </document>
        <document handle="scholnet.csp/2001-TR-021" version="2" rank="0.134224342"/>
    </result-set>
</SearchFeedback>
```

GetMetadataDocuments

```
    Version: 1.0
    Fixed args: none
    Optional_post args: documents, meta_format
    Optional Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400, 501
```

Allows to retrieve the metadata of the documents specified in the format specified. If the format is not indicated by the client of the request, the default format will be assumed.

Returns a structured response that contains a list in which each element of the list provides information on and a link to a document.

The documents parameter is passed as an HTTP POST request of an XML string. Its structure is reported below :

```
<documents>
    <document  handle="" version=""/>
    <document  handle="" version=""/>
    …
</documents>
```

Example Request:
/OLP/Index/1.0/GetMetadataDocuments/%s
Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetMetadataDocuments version="1.0">
        <document handle="scholnet.csp/2001-TR-009" version="1">
            <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
                xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
                    xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:title>A document test</dc:title>
                <dc:date>2001/06/12</dc:date>
            </olms:ol>
        </document>
        <document handle="scholnet.trs/2001-TR-023" version="2">
            <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
                xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
                xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:title>A document test2</dc:title>
                <dc:date>2001/08/12</dc:date>
            </olms:ol>
        </document>
</GetMetadataDocuments>
```

IndexUpdate

```
    Version: 1.0
    Fixed args: metadata-format
    Optional Args: authority, force, verbose
    Return MIME type: text/plain
    Return Status Codes: 200, 400, 501
```

Used to update the IS database. This service request harvests the new bibliographic records from remote Repository servers and adds these new document descriptions to the database.

The meaning of the arguments is as follows:

- *metadata-format*. Specifies the metadata format harvested from the Repository. The metadata format managed by the Index service is a parameter determined at service start-up time.
- *authority*. Specifies the authorities to be updated. If no value is specified, all the authorities specified in the configuration parameters are assumed as default.
- *force*. Forces the updating of all authorities without taking into account the time-stamp of each authority. If no value is specified, only the authorities that have a time-stamp within the specific time (fixed as configuration parameter) will be updated.
- *verbose*. Forces the writing of a verbose log.

The following steps are performed:

1. Lock the IS database in order to update it.
2. Select the authorities that have to be updated.
3. Identify the Repositories that store the documents for those authorities.
4. Send a ListContents request to each identified Repository specifying the authority, the starting date, and the metadata-format.
5. Add bibliographic information for a document to the IS database. Note that the same document may be added more than once to account for changes to bibliographic records.
6. Unlock the database.

The method returns 200 if no error occurs, an error code (400 or 501) otherwise.

Example Request:
/OLP/Index/1.0/IndexUpdate/OLMS

15. QUERY MEDIATOR SERVICE

The Query Mediator (QM) receives search requests and returns a structured list of document metadata, each of which contains a link to the document itself. In order to do this, it routes the query to the appropriate Index Servers, collects the results, and transforms them into a format that is interpretable by the User Interface.
The formats of the metadata searchable by the QM, and of the results metadata are parametric and can be easily changed. The QM harvests these configuration parameters from the Meta Service.
In the Scholnet environment, these formats are subsets of the OpenDLib Application Profile, i.e. any metadata format disseminated by the Repository Service is a potential candidate.

The QM Service is a NoInput replicated service that is driven by the Meta Service, i.e. the Meta Service assigns to each QM a set of index servers that can be used to dispatch the queries. The correct IS is identified using the authority-index relation maintained by the Meta Service

Note that the search methods managed by the QM are not the same as those supported by the Index Service. The QM supports querying on collections, monolingual querying, and pseudo-cross-language querying as follows:
- using the collection description, it performs a query on a set of collections into a SearchFielded query on a set of authorities with the appropriate filtering condition;
- using the Index configuration description furnished by the Meta Service, it selects the appropriate monolingual Index servers, configured to support the language specified in the query, and routes the query to them;
- using the relevance feedback capability of the Index Service, it transforms a cross-language query into multiple monolingual querying.

Communication with the Query Mediator Service takes place via the OpenDLib Protocol (OLP).

15.1 *STATE*

This section presents the data structure, its abstract type, and a brief textual description for each object managed by the Query Mediator Service.

Search Methods

The QM Service will export the search methods set available on the specified metadata format. This information is retrieved using the local configuration of the search methods and the indexed fields supported by the IS, determined analysing the configuration parameters at service start-up time.

Result Formats

The QM Service will return a list of documents in which each element of the list contains the fields specified in one of the available result formats. These formats are determined analysing the configuration parameters at service start-up time.

15.2 *QUERY MEDIATOR VERBS*

In this section, we present the set of the Query Mediator verbs. This set is divided into three sub-sets: Service Information Verb, Service Descriptive Verb, and Service Specific Verb, as reported in the following table:

Service Information Verb	Service Descriptive Verb	Service Specific Verb
Identify	ListSearchMethods	SearchFielded

ListVerbs		**SearchSimple**
DescribeVerb		**SearchFeedback**
		SearchAcross
		GetMetadataDocuments

Identify

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name and other information about the service.

Example Request:
/OLP/QM/1.0/Identify

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<Identify version="1.0">
  <serviceName>Experimental OLP QM Server</serviceName >
  <olp_base_url>http://labserv.iei.pi.cnr.it:8230</olp_base_url>
  <metaServiceURL>http://labserv.iei.pi.cnr.it:8221</metaServiceURL>
  <textualDescription></textualDescription>
  <protocolVersion>1.0</protocolVersion>
  <adminEmail>pagano@iei.pi.cnr.it</adminEmail>
  <verbsInfo>
    <verbsSupported>
        <description>   </description>
        <olp_base_url>http://labserv.iei.pi.cnr.it:8230/OLP/QM/1.0/ListVerbs</olp_base_url>
    </verbsSupported>
      <verbsDescription>
        <description>   </description>
        <olp_base_url>http://…:8230/OLP/QM/1.0/DescribeVerb/DescribeVerb</olp_base_url>
    </verbsDescription>
  </verbsInfo>
  <submissionProcedure>
    <description>Submission procedure is not supported by this service.
    </description>
      <olp_base_url> </olp_base_url>
  </submissionProcedure>
  <harvestInformation>
      <description>
     No information is available
    </description>
    <olp_base_url> </olp_base_url>
  </harvestInformation>
  <useRestrictions>Terms and conditions are …...</useRestrictions>
  <contentInfo>
    <searchMethods>
        <description>
        The set of queries syntax usable to search the set of documents.
      </description>
      <olp_base_url> http://…:8230/OLP/QM/1.0/ListSearchMethods</olp_base_url>
      </searchMethods>
    <resultFormats>
        <description>
          The result formats used by the service are parametric. The pertinent set of result formats is
        available at the specified address.
      </description>
```

```
            <olp_base_url> http://…:8230/OLP/QM/1.0/ListResultFormats</olp_base_url>
        </resultFormats>
    </contentInfo>
</Identify>
```

ListVerbs

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name of the verbs defined by this service.

Example Request:
/OLP/QM/1.0/ListVerbs

DescribeVerb

```
Version: 1.0
Fixed args: verb
Optional Args: version
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on a version of the specified verb that is supported by this service. The following information may be provided at the verb or version level:

- *description*, description of the verb or a specific version
- *note*, information pertaining to the verb or a specific version

Each element of the list contains the following information:

- *version* number of the verb.
- *arguments*, a list of the names of the fixed and optional arguments, if any, accepted by the verb in that version.
- *example* template of request to this Collection, with fixed arguments indicated in brackets
- *returns*, optional, contains information about response format.

Note that a service may implement more than one version of a verb.

Example Request:
/OLP/QM/1.0/DescribeVerb/ListSearchMethods

ListSearchMethods

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on a search method available for searching the library contents.
The QM Service combines the information about the available search methods with the information on the available metadata format and its indexed fields. This information is composed using:

- the local configuration of the search methods;
- the indexed fields list supported by any IS. This list is harvested, at service start-up time, from the Meta Service through a request for the Index configuration description.

- the detailed list of indexed fields supported by the Index servers known by the QM. Note that a QM uses a subset of the overall set of Index servers available in the environment.

This verb returns information on the QM service and this information is built after the initialization of the environment. The non-configured information of the search verbs can be retrieved using the DescribeVerb method.
Note that if the metadata format or the filtering contain empty values, it means that all metadata formats or filtering values are supported.

Example Request:
/OLP/QM/1.0/ListSearchMethods

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<ListSearchMethods version="1.0">
    <Verb name="SearchFielded">
        <description>Search this QM server</description>
        <versions>
            <version id="1.0">
                <example>http://labserv.iei.pi.cnr.it:8226/OLP/QM/1.0/SearchBoolean</example>
                <meta-format name="olms" uri="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
default="true">
                    <information-space>
                        <entity value="(collection|authority)"/>
                        <supported-operators>
                            <operator value="or"/>
                        </supported-operators>
                    </information-space>
                    <filtering value="en" name="dc:language" default="true">
                        <indexed-fields>
                            <field name="dc:creator">
                                <type>string</type>
                                <rel-op name="equal" symbol="=" position="infix"/>
                                <rel-op name="like" symbol="=^" position="infix"/>
                            </field>
                            <field name="dc:title">
                                <type>string</type>
                                <rel-op name="equal" symbol="=" position="infix"/>
                                <rel-op name="like" symbol="=^" position="infix"/>
                            </field>
                            <field name="dc:subject">
                                <type>string</type>
                                <rel-op name="equal" symbol="=" position="infix"/>
                                <rel-op name="like" symbol="=^" position="infix"/>
                            </field>
                            <field name="dc:date">
                                <type>%4d/%2d/%d(%d)</type>
                                <rel-op name="equal" symbol="=" position="infix"/>
                                <rel-op name="greater than" symbol=">" position="infix"/>
                            </field>
                            <supported-operators>
                                <operator value="and"/>
                                  <operator value="or"/>
                            </supported-operators>
                        </indexed-fields>
                        <result-formats>
                            <format name="short" default="yes">
                                <field>dc:title</field>
                                <field>dc:date</field>
                            </format>
                            <format name="long">
                                <field>dc:creator</field>
                                <field>dc:title</field>
                                <field>dc:date</field>
                                <field>dc:subject</field>
                            </format>
                        </result-formats>
                    </filtering>
                    <filtering value="it|fr" name="dc:language">
                        <indexed-fields>
                            <field name="dc:creator">
                                <type>string</type>
                                <rel-op name="equal" symbol="=" position="infix"/>
                                <rel-op name="like" symbol="=^" position="infix"/>
                            </field>
                            <field name="dc:date">
                                <type>%4d/%2d/%d(%d)</type>
                                <rel-op name="equal" symbol="=" position="infix"/>
                                <rel-op name="greater than" symbol=">" position="infix"/>
                            </field>
                            <field name="dcq:title.alternative">
                                <type>string</type>
                                <rel-op name="equal" symbol="=" position="infix"/>
                                <rel-op name="like" symbol="=^" position="infix"/>
                            </field>
                            <field name="dcq:abstract.alternative">
                                <type>string</type>
                                <rel-op name="equal" symbol="=" position="infix"/>
                                <rel-op name="like" symbol="=^" position="infix"/>
                            </field>
                            <supported-operators>
                                <operator value="and"/>
                                <operator value="or"/>
                            </supported-operators>
                        </indexed-fields>
                        <result-formats>
                            <format name="short" default="yes">
                                <field>dc:title</field>
                                <field>dc:date</field>
                            </format>
                            <format name="long">
                                <field>dc:creator</field>
                                <field>dc:title</field>
                                <field>dc:date</field>
                                <field>dc:subject</field>
                            </format>
                        </result-formats>
                    </filtering>
                </meta-format>
            </version>
        </versions>
    </Verb>
    <Verb name="SearchSimple">
        <description>Search this QM server</description>
        <versions>
            <version id="1.0">
                <example>http://labserv.iei.pi.cnr.it:8226/OLP/QM/1.0/SearchBoolean</example>
                <meta-format name="olms" uri="http://project.it/OLP/htdocs/olms/NameSpace/" default="true">
                    <information-space>
                        <entity value="(collection|authority)"/>
                        <supported-operators>
```

```
                        <operator value="or"/>
                    </supported-operators>
                </information-space>
                <filtering value="en" name="dc:language" default="true">
                    <indexed-fields>
                        <field name="condition">
                            <type>string</type>
                            <rel-op name="equal" symbol="=" position="infix"/>
                            <rel-op name="like" symbol="=^" position="infix"/>
                        </field>
                    </indexed-fields>
                    <result-formats>
                        <format name="short" default="yes">
                            <field>dc:title</field>
                            <field>dc:date</field>
                        </format>
                        <format name="long">
                            <field>dc:creator</field>
                            <field>dc:title</field>
                            <field>dc:date</field>
                            <field>dc:subject</field>
                        </format>
                    </result-formats>
                </filtering>
            </meta-format>
        </version>
    </versions>
</Verb>
<Verb name="SearchFeedback">
    <description>Allow to use the relevance feedback capability</description>
    <versions>
        <version id="1.0">
            <example>http://labserv.iei.pi.cnr.it:8226/OLP/QM/1.0/SearchFeedback</example>
            <meta-format name="" uri="" default="true">
                <filtering value="" name="" default="true">
                    <result-formats/>
                </filtering>
            </meta-format>
        </version>
    </versions>
</Verb>
<Verb name="SearchAcross">
    <description>Search this QM server</description>
    <versions>
        <version id="1.0">
            <example>http://labserv.iei.pi.cnr.it:8230/OLP/QM/1.0/SearchAcross</example>
            <meta-format name="olms" uri="http://project.it/OLP/htdocs/olms/NameSpace/" default="true">
                <information-space>
                    <entity value="(collection|authority)"/>
                    <supported-operators>
                        <operator value="or"/>
                    </supported-operators>
                </information-space>
                <filtering value="en" name="dc:language" default="true">
                    <indexed-fields>
                        <field name="dc:subject.ccs">
                            <type>string</type>
                            <rel-op name="equal" symbol="=" position="infix"/>
                            <rel-op name="like" symbol="=^" position="infix"/>
                        </field>
                        <field name="dc:description.abstract">
                            <type>string</type>
                            <rel-op name="equal" symbol="=" position="infix"/>
                            <rel-op name="like" symbol="=^" position="infix"/>
                        </field>
                    </indexed-fields>
                    <result-formats>
                        <format name="short" default="yes">
                            <field>dc:title</field>
                            <field>dc:date</field>
                        </format>
                        <format name="long">
                            <field>dc:creator</field>
                            <field>dc:title</field>
                            <field>dc:date</field>
                            <field>dc:subject.ccs</field>
                        </format>
                    </result-formats>
                </filtering>
                <filtering value="it|de" name="dc:language">
                    <indexed-fields>
                        <field name="dc:subject.ccs.alternative">
                            <type>string</type>
                            <rel-op name="equal" symbol="=" position="infix"/>
                            <rel-op name="like" symbol="=^" position="infix"/>
                        </field>
                        <field name="dc:description.abstract.alternative">
                            <type>string</type>
                            <rel-op name="equal" symbol="=" position="infix"/>
                            <rel-op name="like" symbol="=^" position="infix"/>
                        </field>
                    </indexed-fields>
                    <result-formats>
                        <format name="short" default="yes">
                            <field>dc:title</field>
                            <field>dc:date</field>
                        </format>
                        <format name="long">
                            <field>dc:creator</field>
                            <field>dc:title</field>
                            <field>dc:date</field>
                            <field>dc:subject.ccs</field>
                        </format>
                    </result-formats>
                </filtering>
            </meta-format>
        </version>
    </versions>
</Verb>
</ListSearchMethods>
```

SearchFielded

```
    Version: 1.0
    Fixed args: none
    Fixed_post args: file (search_specification)
    Optional Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400, 501
```

The fielded search allows the user to restrict his/her search criteria to specific fields of the selected metadata formats, and to use date and numeric ranges, as far as right types and controlled vocabulary.
The verb returns a structured response that contains a list in which each element of the list provides information on and a link to a document. This list, called result-set, is uniquely named by the QM that can accept query refinements on it. Note that:

1) the QM creates this list as a combination of multiple named result-set lists (each of which contains a different set of documents) generated by different IS;
2) the QM maintains the relation (named rs-rel+) between the QM result-set name and the result-set names generated by the Index servers.

To perform a refinement on the *result*-set, see the SearchFeedback verb: in this case the the QM dispatches the query, described by means of the search_specification argument, to every Index server indicating the appropriate result-set name that is identified using the *rs-rel+* relation.
The result-set remains active for a given amount of time, and when it expires, error 400 is returned.

The search-specification is passed as an HTTP POST request of an XML file. This specification contains two parts. The first part includes metadata that describes the set of collections/authorities on which to perfom the query, the meta-format selected, the result set format requested by the user, the filter value that selects the appropriate search method, and the specification of the max number of documents returned (called top_documents). Note that if the top_documents is not specified, the default value is assumed, and if the "-1" value is specified all documents that match the request will be returned in the requested format. To retrieve the other documents of the result-set see GetMetadataDocuments verb.
The second describes the filtering condition. Note that the filtering condition is performed only on the set of documents that belong to the collections/authorities specified, and that if the *result_set_format* option is not specified the default option is assumed.

The search condition is formulated in accordance with the following BNF grammar:
Search-Condition ::= Filter Filtering-Condition | Part1 Filter Filtering-Condition
Part1 ::= Format Auths | Format Colls | Format | Auths | Colls Op | Part1 Top
Format ::= *Meta_format=string* | *Meta_format=string, Result_Set_Format = string*
Top ::= *top_documents = number*
Filter ::= *filter-name = string*
Auths ::= *Authories = (list of authorities)*
Colls ::= *collection-name* | *collection-name* Colls
Filtering-Condition ::= Leaf | Condition
Condition ::= Leaf Operator Leaf | Leaf Operator Condition | Condition Operator Leaf | Condition Operator Condition
Leaf ::= Attr Rel-Op *Value*
Rel-Op ::= *like* | = | …
Op ::= *and* | *or*
Operator ::= #b*and* | #*or* | ….
Attr ::= *indexed_field*

Note that if the user does not specify a list of authorities/collections, all authorities are assumed as default. The "and/or" operator can be used to specify the collections/authorities combination in the information space. If the operator is not specified, the "or" value is assumed as default.
Note also that if the user does not specify the response format the default format is assumed.

The structure of the search-specification is reported below:

```
<search-condition>
   <meta_format name=" …" result_set_format_name=".."/>
   <information_space>
```

```
        <collections>
            <collection name="…." />
            <collection name="…." />
            ……
        </collections>
        <op value=""/>
    </information_space>
    <top_documents value="…."/>
    <filtering value="it" name="dc:language">
        <filtering-condition>
             <condition>
                 <leaf attribute="" value="" rel-op="" side="left"/>
                <operator name=" "/>
                <condition value=" .." side="right">
                     <leaf attribute="" value="" rel-op="" side="left"/>
                     <operator name=""/>
                     <leaf attribute="" value="" rel-op="" side="right"/>
                </condtition>
            </condition>
        </filtering-condition>
    </filtering>
</search-condition>
```

The filtering condition selects the documents published by a set of specific authorities/collections that belong to the result. Note that if *collections* are specified, the *authorities* are ignored.

At present the search-condition includes information about the authorities/collections that store documents relevant to the request of the client. If the authorities/collections are not specified, the QM performs the query on all authorities. In the future, techniques that allow the QM to select automatically the authorities/collections that publish documents compliant with the filtering-condition will be considered.

The formats available can be selected from the list returned by the ListResultFormats.

Example Request:
/OLP/QM/1.0/SearchFielded/%s

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<SearchFielded version="1.0">
    <result-set name="21908123" count="98">
        <document handle="scholnet.csp/2001-TR-009" version="12" rank="0.9084948">
            <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
                xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
                xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:title>A document test</dc:title>
                    <dc:date>2001/06/12</dc:date>
            </olms:ol>
        </document>
        <document handle="scholnet.trs/2001-TR-023" version="2" rank="0.5434534">
            <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
                xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
                xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:title>A document test2</dc:title>
                <dc:date>2001/08/12</dc:date>
            </olms:ol>
        </document>
    </result-set>
</SearchFielded>
```

SearchSimple

```
     Version: 1.0
     Fixed args: none
     Fixed_post args: file (search_specification)
     Optional Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400, 501
```

The simple search allows the user to search the documents in the selected collection(s) that have been described using the specified metadata formats without restricting his/her search criteria to specific fields. This verb performs a full text search and therefore neither numeric ranges nor controlled vocabularies can be applied.
The verb returns a structured response that contains a list in which each element of the list provides information on and a link to a document. This list, called result-set, is uniquely named by the QM that can accept query refinements on it. Note that:

3) the QM creates this list as combination of multiple named result-set lists (each of which contains a different set of documents) generated by different IS;
4) the QM maintains the relation (named rs-rel+) between the QM result-set name and the result-set names generated by the Index servers.

To perform a refinement on the *result*-set, see the SearchFeedback verb: in this case the the QM dispatches the query, described by means of the search_specification argument, to every Index server indicating the appropriate result-set name that is identified using the *rs-rel+* relation.
The result-set remains active for a given amount of time, and when it expires, error 400 is returned.

The search-specification is passed as an HTTP POST request of an XML file. This specification contains two parts. The first part includes metadata that describes the set of collections/authorities on which performs the query, the meta-format selected, the result set format requested by the user, the filter value that selects the appropriate search method, and the specification of the max number of documents returned (called top_documents). Note that if the top_documents is not specified, the default value is assumed, and if the "-1" value is specified all documents that match the request will be returned in the requested format. To retrieve the other documents of the result-set see GetMetadataDocuments verb.
The second one allows to describe the filtering condition. Using the SearchSimple, the request of the client does not specify the fields on which the search has to be performed, and the QM calculates the response as a result of a free text search.
Note that the filtering condition specified is performed only on the set of documents that belong to the collections/authorities specified, and that if the *Result_Set_Format* option is not specified the default one is assumed.
The first part includes metadata that describes the set of collections/authorities on which performs the query, the result format requested by the user, and the filter value that allows to select the appropriate search method. The second one allows to describe the filtering condition.

The search condition is formulated in accordance with the following BNF grammar:
Search-Condition ::= Filter Filtering-Condition | Part1 Filter Filtering-Condition | Filtering-Condition
Part1 ::= Format Auths | Format Colls | Format | Auths | Colls Op | Part1 Top
Top ::= *top_documents = number*
Format ::= *Meta_format=string* | *Meta_format=string, Result_Set_Format = string*
Filter ::= *filter-name = string*
Auths ::= *Authories = (list of authorities)*
Colls ::= *collection-name* | *collection-name* Colls
Op ::= *and* | *or*
Filtering-Condition::= *string*

Note that if the user does not specify a list of authorities/collections, all authorities are assumed as default. The "and/or" operator can be used to specify the collections/authorities combination in the information space. If the operator is not specified, the "or" value is assumed as default.
Note also that if the user does not specify the response format the default one is assumed.

The structure of the search-specification is reported below:

```
<search-condition>
    <meta_format name=" …" result_set_format_name=".."/>
    <information_space>
        <collections>
            <collection name="…." />
            <collection name="…." />
            ……
        </collections>
        <op value=""/>
    </information_space>
    <top_documents value="…."/>
    <filtering value="%lang" name="dc:language">
        <filtering-condition value=" …."/>
    </filtering>
</search-condition>
```

The filtering condition allows to select which documents, published by a set of specific authorities/collections, belong to the result. Note that if *collections* are specified, the *authorities* one are ignored.

At present, the search-condition includes information about the authorities/collections that store documents relevant to the request of the client. If the authorities/collections are not specified, the QM performs the query on all authorities. In the future, techniques that allow the QM to select automatically the authorities/collections that publish documents compliant with the filtering-condition will be taken into account.

The formats available can be selected from the list returned by the ListResultFormats.

```
Example Request:
/OLP/QM/1.0/SearchSimple/%s

Example Response:
<?xml version="1.0" encoding="UTF-8"?>
<SearchSimple version="1.0">
    <result-set name="21908123" count="28">
        <document handle="scholnet.csp/2001-TR-009" version="12" rank="0.9084948">
            <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
                xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
                xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:title>A document test</dc:title>
                <dc:date>2001/06/12</dc:date>
            </olms:ol>
        </document>
        <document handle="scholnet.trs/2001-TR-023" version="1" rank="0.433248">
            <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
                xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
                xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:title>A document test2</dc:title>
                <dc:date>2001/08/12</dc:date>
            </olms:ol>
        </document>
    </result-set>
</SearchSimple>
```

SearchFeedback

```
Version: 1.0
Fixed args: none
Fixed_post args: feedback-condition
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

The feedback search allows the user to access the relevance feedback capability supported by the QM service. The relevance feedback is used as a solution for query modification in order to improve the quality of the results. The user has to specify the relevant retrieved as response to a search. The service use this information to modify the original query. The new query is performed and the results returned to the user.
The verb returns a structured response that contains a list in which each element of the list provides information on and a link to a document. This list, called result-set, is uniquely named by the QM that is able to accept query refinement on it. Note that:

5) the QM creates this list as a combination of multiple named result-set list (each of which contains a different set of documents) generated by different IS;
6) the QM maintains the relation (named rs-rel+) between the QM result-set name and the result-set names generated by the Index servers.

The result-set remains active for a given amount of time, and when it is expires, error 400 is returned.

The feedback-condition is passed as an HTTP POST request of an XML string. This specification contains two parts. The first part includes metadata that report the result set name requested by the user, and the specification of the max number of documents returned in the original format specified (called top_documents). To retrieve the other documents see GetMetadataDocuments verb. Note that if the top_documents is not specified, the default value is assumed, and if the "-1" value is specified all documents will be returned in the requested format.
The second part of the condition includes the list of relevant documents.

The search specification is formulated in accordance with the following BNF grammar:
Feedback-Condition ::= Part1 Rel_Docs
Part1 ::= Result_Set Top | Result_Set
Result_Set ::= *result-set = string*
Top ::= *top_documents = number*
Rel_Docs ::= Doc | Doc Rel_Docs
Doc ::= *document = string*

The structure of the search-specification is reported below:

```
<feedback-condition>
    <result-set name=""/>
    <top_documents value=""/>
    <rel_documents>
        <document handle="" version=""/>
        <document handle="" version=""/>
    </rel_documents>
</feedback-condition>
```

Example Request:

/OLP/QM/1.0/SearchFeedback/%s

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<SearchFeedback version="1.0">
  <result-set name="21908123" count="98">
    <document handle="scholnet.csp/2001-TR-009" version="5" rank="0.434224342">
      <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
        xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
          xmlns:dc="http://purl.org/dc/elements/1.1/">
        <dc:title>A document test</dc:title>
        <dc:date>2001/06/12</dc:date>
      </olms:ol>
    </document>
    <document handle="scholnet.trs/2001-TR-023" version="2" rank="0.234224342">
      <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
        xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
        xmlns:dc="http://purl.org/dc/elements/1.1/">
        <dc:title>A document test2</dc:title>
        <dc:date>2001/08/12</dc:date>
      </olms:ol>
    </document>
  </result-set>
</SearchFeedback>
```

SearchAcross

```
    Version: 1.0
    Fixed args: none
    Fixed_post args: file (search_specification)
    Optional Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400, 501
```

This verb allows the user to access the cross-querying capability of the QM. The user has to specify a value for an attribute that supports the cross-querying capability, and the characteristic of the subset of the results that he/she wants to receive. For example, let us suppose that the subject attribute supports the cross-querying capability with respect to the language. This means that the user can specify a subject in Italian and retrieve all documents containing related subjects, even if these documents are described in a different language.
The verb returns a structured response that contains a list in which each element of the list provides information on and a link to a document. This list, called result-set, is uniquely named by the QM that is able to accept query refinement on it. Note that:

7) the QM creates this list as a combination of multiple named result-set list (each of which contains a different set of documents) generated by different IS;
8) the QM maintains the relation (named rs-rel+) between the QM result-set name and the result-set names generated by the Index servers.

To perform a refinement on the *result*-set, see the SearchFeedback verb: in this case the the QM dispatches the query, described by means of the search_specification argument, to every Index server indicating the appropriate result-set name that is identified using the *rs-rel+* relation.
The result-set remains active for a given amount of time, and when it expires, error 400 is returned.

The search-specification is passed as an HTTP POST request of an XML file. This specification contains two parts. The first part includes metadata that describes the set of collections/authorities on which performs the query, the meta-format selected, the result set format requested by the user, and the specification of the max number of documents returned (called top_documents). Note that if the top_documents is not specified, the default value is assumed, and if the "-1" value is specified

all documents that match the request will be returned in the requested format. To retrieve the other documents of the result-set see GetMetadataDocuments verb.
The second one allows to describe the filtering condition. Using the SearchAcross, the request of the client must specify the fields on which the search is to be performed and the source and target filters; the QM calculates the response as a result of the search.

Note that the filtering condition is performed only on the set of documents that belong to the collections/authorities specified and that if the *Result_Set_Format* option is not specified the default option is assumed.
The first part includes metadata that describes the set of collections/authorities on which to perform the query, the result format requested by the user, and the filter value that selects the appropriate search method. The second part describes the filtering condition.

The search condition is formulated in accordance with the following BNF grammar:

```
Search-Condition ::= Filter Filtering-Condition | Part1 Filter Filtering-Condition
Part1 ::= Format Auths | Format Colls | Format | Auths | Colls Op | Part1 Top
Format ::= Meta_format=string | Meta_format=string, Result_Set_Format = string
Top ::= top_documents = number
Filter ::= filter-name = string
Auths ::= Authories = (list of authorities)
Colls ::= collection-name | collection-name Colls
Filtering-Condition::= Leaf filtering_target
Leaf::= filtering_attribute Rel-Op Value
Rel-Op ::= like | = | contains |…
Op ::= and | or
```

Note that if the user does not specify a list of authorities/collections, all authorities are assumed as default. The “and/or” operator can be used to specify the collections/authorities combination in the information space. If the operator is not specified, the “or” value is assumed as default.
Note also that if the user does not specify the response format the default format is assumed.

An example of the structure of the search-specification is reported below:

```
<search-condition>
   <meta_format name=” …” result_set_format_name=”..”/>
   <information_space>
      <collections>
         <collection name="…." />
         <collection name="…." />
         ……
      </collections>
      <op value=""/>
   </information_space>
   <top_documents value="…."/>
   <filtering value=”it” name="dc:language”>
      <filtering-condition target=”en”>
            <leaf attribute="subject" value="" rel-op=”” />
      </filtering-condition>
   </filtering>
</search-condition>
```

The filtering condition selects which documents published by a set of specific authorities/collection, belong to the result. Note that if *collections* are specified, the *authorities* are ignored.

At present, the search-condition includes information about the authorities/collections that store documents relevant to the request of the client. If the authorities/collections are not specified, the QM performs the query on all authorities. In the future, techniques that allow the QM to select automatically the authorities/collections that publish documents compliant with the filtering-condition will be considered.

The formats available can be selected from the list returned by the ListResultFormats.

Example Request:
/OLP/QM/1.0/SearchAcross/%s

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
< SearchAcross version="1.0">
   <result-set name="21908123" count="98">
      <document handle="scholnet.csp/2001-TR-009" version="12" rank="0.9084948">
         <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
            xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
            xmlns:dc="http://purl.org/dc/elements/1.1/">
            <dc:title>A document test</dc:title>
               <dc:date>2001/06/12</dc:date>
         </olms:ol>
      </document>
      <document handle="scholnet.trs/2001-TR-023" version="2" rank="0.5434534">
         <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
            xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
            xmlns:dc="http://purl.org/dc/elements/1.1/">
            <dc:title>A document test2</dc:title>
            <dc:date>2001/08/12</dc:date>
         </olms:ol>
      </document>
   </result-set>
</ SearchAcross >
```

GetMetadataDocuments

```
Version: 1.0
Fixed args: result-set
Optional args: start, results
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Allows to retrieve the metadata of the documents belonging to the result-set generated by a previous query. The set of returned documents is defined by the range (start, start + results]. If the results is not specified, the default value will be assumed. If the start value is not specified the interval will start with the first element not yet returned to the user.
Returns a structured response that contains a list in which each element of the list provides information on and a link to a document.

Example Request:
/OLP/Index/1.0/GetMetadataDocuments/3213112131?start=10&results=20
Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<GetMetadataDocuments version="1.0">
      <document handle="scholnet.csp/2001-TR-009" version="1">
         <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
            xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
               xmlns:dc="http://purl.org/dc/elements/1.1/">
            <dc:title>A document test</dc:title>
```

```
                <dc:date>2001/06/12</dc:date>
            </olms:ol>
        </document>
        <document handle="scholnet.trs/2001-TR-023" version="2">
            <olms:ol xmlns:olms="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/"
                xmlns:dcq="http://dublincore.org/documents/2000/07/11/dcmes-qualifiers/"
                xmlns:dc="http://purl.org/dc/elements/1.1/">
                <dc:title>A document test2</dc:title>
                <dc:date>2001/08/12</dc:date>
            </olms:ol>
        </document>
</ GetMetadataDocuments >
```

16. PERSONALIZED INFORMATION DISSEMINATION SERVICE

The Personalized Information Dissemination Service (PIDS) is independent of the rest of the Scholnet system. The PIDS and Scholnet communicate via the OpenDLib Protocol. The PIDS automatically notifies a user when a new document, matching the user's interests (called *topics*), is available in the Scholnet digital library. Inside the PIDS, each topic is defined in terms of categories, such as ACM and AMS, plus additional free keywords that further refine the topic description. The user may handle the definitions of his/her own topics, that is, create, list, edit, and delete them.
The PIDS will send out alerts of documents matching the user-specified topics by e-mail. This can be initiated in two different ways: by the PIDS scheduler which at fixed time intervals will retrieve (*pull*) a list of metadata records from the Scholnet library; by any service which may send (*push*) a list of metadata records to the PIDS. The metadata records received by either of these two procedures will be matched against all topics of all users, and the users for which relevant matches have been identified will then be notified.

16.1 *STATE*

The PIDS thus handles two types of objects, PIDS internal objects and the metadata records needed for communication with Scholnet. Internally, the PIDS keeps track of a user database and a topic database.

Metadata records

The PIDS receives a list of XML-encoded metadata records in Dublin Core qualified[3] format from the Scholnet system as a result of an execution of either a push or a pull metadata operation (see the functional description of the pull/push metadata use case). The format of the metadata records is as in the Scholnet system proper.

PIDS User Database

The PIDS internal user database is compatible with the Scholnet one, but contains additional information about each user, i.e. the user's e-mail address and pointers to the topics of interest to that user. Thus each record in the user database consists of:

Attribute		Type	Constraint
user-id	Mandatory	String	alphanumeric characters plus underscore
email-address	Mandatory	String	alphanumeric characters plus [-_@%]
topic-id	Mandatory	List	list of alphanumeric characters plus underscore

PIDS Topic Database

Every topic in the PIDS internal topic database has a unique identifier. Associated with each topic, there is a topic label (the name of the topic), as well as lists of free keywords given by the user and of categories (e.g. ACM categories) which the user has selected as being relevant for the topic. Finally each object in the topic database contains a flag specifying whether to notify the user or not. Thus we have:

[3] See http://www.dublincore.org/documents/dcmes-xml/

Attribute		Type	Constraint
topic-id	Mandatory	String	alphanumeric characters plus underscore
topic-label	Mandatory	String	alphanumeric characters plus underscore
keywords	Optional	List	list of alphanumeric characters plus underscore
categories	Optional	List	list of alphanumeric characters plus underscore
notification-flag	Mandatory	Binary	

16.2 *PERSONALISATION VERBS*

In this section, we present the functionality provided by the PIDS for reactive communication with the Scholnet system. In addition, the PIDS expects the Scholnet system to support the reverse situation, i.e. the retrieval of records inserted into the Scholnet database after a certain point in time. Finally, the PIDS supports a range of internal functions for its own processing and communication between its different parts, i.e., the PIDS scheduler, the profile manager, the topic matching filter, and the deliverer which notifies the users.
This verb set is divided into two sub-sets: Service Information Verb and Service Specific Verb as reported in the following table

Service Information Verb	Service Specific Verb
Identify	**MatchMetadata**
ListVerbs	**CreateTopic**
DescribeVerb	**ListTopics**
	EditTopic
	DeleteTopic

Identify

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name and other information about the service.

Example Request:
/OLP/PIDS/1.0/Identify

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
<Identify version="1.0">
   <service-name>PIDS Server</service-name>
   <base-url>http://labserv.iei.pi.cnr.it:8119</base-url>
   <protocol-version>1.0</protocol-version>
   <admin-email> straccia@iei.pi.cnr.it </admin-email>
   <description>
       <content-description>
```

```
            a human readable description of the content stored in the PIDS
        </content-description>
    </description>
</Identify>
```

ListVerbs

```
     Version: 1.0
     Fixed args: none
     Optional Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

Returns a structured response containing the name of the verbs defined by this service.

Example Request:
/OLP/PIDS/1.0/ListVerbs

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListVerbs version="1.0">
    <verb>Identify</verb>
    <verb>ListVerbs</verb>
    <verb>DescribeVerb</verb>
    <verb>MatchMetadata</verb>
    <verb>CreateTopic</verb>
    <verb>ListTopics</verb>
    <verb>EditTopic</verb>
    <verb>DeleteTopic</verb>
</ListVerbs>
```

DescribeVerb

```
     Version: 1.0
     Fixed args: verb
     Optional Args: version
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

Returns a structured response that contains a list in which each element of the list provides information on a version of the specified verb that is supported by this service. The following information may be provided at the verb or version level.
description, description of the verb or a specific version
note, information pertaining to the verb or a specific version
Each element of the list contains the following information:
version number of the verb.
arguments, a list of the names of the fixed and optional arguments, if any, accepted by the verb in that version.
example template of request to this repository, with fixed arguments indicated in brackets
returns, optional, contains information about response format.
Note that a service may implement more than one version of a verb.

Example Request:
/OLP/PIDS/1.0/DescribeVerb/ListTopics

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <DescribeVerb version="1.0">
    <verb name="ListTopics">
      <description>
```

```
            Returns a structured response that contains a list of all topic-ids corresponding to the given user-
            id.
      </description>
      <versions>
        <version id="1.0">
          <arguments>
           <fixed>
             <arg name="user-id" />
           </fixed>
           <optional>
           </optional>
          </arguments>
          <example>http://../OLP/PIDS/1.0/ListTopics/{example}</example>
        </version>
      </versions>
    </verb>
  </DescribeVerb>
```

MatchMetadata

```
     Version: 1.0
     Fixed args: metadata-record-list
     Optional Args: none
     Return MIME type: none
     Return Status Codes: 200, 400
```

Returns a status code indicating whether the PIDS was able to process the list of metadata records or not, i.e., whether the records were successfully matched against all topics of all users and whether notifications were successfully sent out to the users concerned.

Example Request:
/OLP/PIDS/1.0/MatchMetadata

CreateTopic

```
     Version: 1.0
     Fixed args: user-id, topic-id, topic-label, notification-flag
     Optional Args: keywords, categories
     Return MIME type: none
     Return Status Codes: 200, 400, 503
```

This verb will be used either to create a new topic or to edit an existing one. In the former case, the topic-id will be empty and PIDS will be able to calculate a unique one through the user-id. In the latter case, the topic-id will uniquely identify the topic that the user wants to modify.
The mandatory binary element notification-flag is an On/Off flag specifying whether to notify the user or not (1 is ON, 0 is OFF).

Example Request:
/OLP/PIDS/1.0/CreateTopic

ListTopics

```
     Version: 1.0
     Fixed args: user-id
     Optional Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

Returns a structured response that contains a list of all topic-ids corresponding to the given user-id.

Example Request:
/OLP/PIDS/1.0/ListTopics

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListTopics version="1.0">
  <topic topic-id="1a"
        topic-label="Personal Assistant"
        keywords="laptops, palmtop, personal digital assistants"
        categories="ACM - C.5.3 - Portable devices"
        notification-flag="1"/>
  <topic
      topic-id="2b"
      topic-label="Digital Libraries"
      categories="ACM - H.3.7 -  Digital Libraries"
      notification-flag="1"/>
  <topic topic-id="3.5"
      topic-label="Digital Network"
      keywords="networks"
      categories="ACM - C.2.1 - ISDN"
      notification-flag="0"/>
  <topic topic-id="21a"
      topic-label="Distributed programming"
      categories="ACM - D.1.3 - Distributed programming"
      notification-flag="1"/>
</ListTopics>
```

EditTopic

```
Version: 1.0
Fixed args: topic-id
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Example Request:

/OLP/PIDS/1.0/EditTopic

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<EditTopic version="1.0">
  <topic topic-id="1a"
        topic-label="Personal Assistant"
        keywords="laptops, palmtop, personal digital assistants"
        categories="ACM - C.5.3 - Portable device"
        notification-flag="1"/>
</EditTopic>
```

DeleteTopic

```
Version: 1.0
Fixed args: topic-id
Optional Args: none
Return MIME type: none
Return Status Codes: 200, 400
```

Example Request:

/OLP/PIDS/1.0/DeleteTopic

17. HYPERMEDIA ANNOTATION SERVICE

The Hypermedia Annotation Service will integrate annotation and reference linking features into the digital library infrastructure. It stores annotations on documents and makes them available to authorised users.
The Hypermedia Annotation Service is an independent service that can be distributed on multiple servers and it will be based on the Semantic Index System (SIS) developed by FORTH [http://www.ics.forth.gr/proj/isst/Systems/sis.html]. The Semantic Index System (SIS) is a tool for describing and documenting large evolving varieties of highly interrelated data, concepts and complex relationships, as opposed to large homogeneous populations in fixed formats (handled by traditional DBMS). The Hypermedia Annotation Service communicates with Scholnet via the Open Library Protocol.

17.1 *STATE*

The Hypermedia Annotation Service will provide the capability to view and/or update the Annotation Server SIS database through XML documents. The Hypermedia Annotation Service handles two types of objects: annotations and annotation metadata. The annotation repository stores only the annotations while the annotation metadata are the result of a mapping between the annotation record and OLAP.

Annotations
Annotations are identified by handles in the same way as documents are. Handles are a kind of URN that includes two parts, a naming authority and a string, separated by a slash. Each annotation keeps references to the handles of the annotated documents or annotations and pointers inside a document if the annotation refers to specific parts of a document. For each referred document, the system also keeps a short description of the document (title, author, etc.) in accordance with the OLAP metadata record of the document in the repository.
The annotation record includes the following fields:

Attribute		Type	Constraint
annotation-id	Mandatory	Handle	unique identifier of the annotation in the system
author	Mandatory	String	loginname of user who created the annotation
project	Optional	List	list of project names whose members can access the annotation
group	Optional	List	list of group names whose members can access the annotation
subject	Optional	List	list of keywords
type	Mandatory	String	annotation type (Comparison, Multidocument Agreement, Multidocument Disagreement, Reference, Similar with, Missing Reference, Rating, Comment, Agreement, Disagreement, Explanation, Question)
text	Optional	String	free text

date		Mandatory	Date	creation date
program		Optional	List	list of keywords providing a working program/project context to the annotation
links		Optional	Tuple	linkType, toValue
	linkType	Mandatory	String	*linkType* is the type of relation between the annotation and the referred object (e.g. rates, describes similar work with, agrees with, disagrees with, explains, inspires, …)
	toValue	Mandatory	String	*ToValue* is the referred object and can be a Scholnet DL Document handle, a pointer inside a Scholnet DL Document handle, an external URL or a Scholnet Annotation handle

Annotation metadata

Each annotation is associated with a metadata record in Dublin Core qualified (DCMeS http://www.dublincore.org/documents/dcmes-qualifiers/) format. The mapping between the Annotation Record (AR) and the Dublin Core metadata record (dc) is the following:

dc:creator = AR.author
dc:subject = "Annotation", AR.subject, AR.type
dc:description = AR.text
dc:date.modified = AR.date
dc:type = "text"
dc:format = Scholnet_annotation.dtd (the XML DTD that will be defined for annotations)
dc:identifier = AR.annotation-id
dc.relation.references = AR.program, AR.links.toValue
dc:rights = AR.group, AR.project

17.2 *HYPERMEDIA ANNOTATION VERBS*

In this section, we present the functionality provided by the Hypermedia Annotation Service for communication with the Scholnet system.
This verb set is divided into two sub-sets: Service Information Verb and Service Specific Verb as reported in the following table[4]:

Service Information Verb	**Service Specific Verb**
Identify	**BeginSession**
ListVerbs	**EndSession**
DescribeVerb	**FetchAnnotation**
	CreateAnnotation
	UpdateAnnotation
	SearchAnnotation

[4] See Appendix A for verb's DTDs

	GetRefDocuments
	GetAnnotationsofDocument
	DeleteAnnotation
	DisplayAnnotations

Identify

```
Version: 1.0
Fixed args: none
Keyword Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name and other information about the service.

Example Request:
/OLP/HAS/1.0/Identify

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<Identify version="1.0">
   <serviceName>Hypermedia Annotation Service</serviceName >
   <baseURL>http://annotationserver.ics.forth.gr:8080</baseURL>
   <protocolVersion>1.0</protocolVersion>
   <adminEmail>maria@ics.forth.gr</adminEmail>
   <description>
       <authorities>
               <description>
                   This Hypermedia Annotation Server handles annotations related to documents that
                   belong to a subset of the overall set of authorities.
               </description>
       </authorities>
       <contentDescription>
           a human readable description of the content stored in the Hypermedia Annotation Service
       </contentDescription>
   </description>
</Identify>
```

ListVerbs

```
Version: 1.0
Fixed args: none
Keyword Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Returns a structured response containing the name of the verbs defined by this service.

Example Request:
/OLP/HAS/1.0/ListVerbs

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListVerbs version="1.0">
   <verb>Identify</verb>
   <verb>ListVerbs</verb>
   <verb>DescribeVerb</verb>
   <verb>BeginSession</verb>
   <verb>EndSession</verb>
   <verb>FetchAnnotation</verb>
   <verb>CreateAnnotation</verb>
   <verb>UpdateAnnotation</verb>
   <verb>SearchAnnotation</verb>
```

```
  <verb>GetRefDocuments</verb>
  <verb>GetAnnotationsofDocument</verb>
  <verb>DeleteAnnotation</verb>
  <verb>DisplayAnnotations</verb>
</ListVerbs>
```

DescribeVerb

```
Version: 1.0
Fixed args: verb
Keyword Args: version
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Returns a structured response that contains a list in which each element of the list provides information on a version of the specified verb that is supported by this service. The following information may be provided at the verb or version level.
description, description of the verb or a specific version
note, information pertaining to the verb or a specific version
Each element of the list contains the following information:
version number of the verb.
arguments, a list of the names of the fixed and keyword arguments, if any, accepted by the verb in that version.
example template of request to this repository, with fixed arguments indicated in brackets
returns, optional, contains information about response format.
Note that a service may implement more than one version of a verb.

Example Request:
/OLP/HAS/1.0/DescribeVerb/FetchAnnotation

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
 <DescribeVerb version="1.0">
  <verb name="FetchAnnotation">
   <description>
         Returns an XML document that contains a description of the annotation corresponding to the
         given AnnotationID.
   </description>
   <versions>
    <version id="1.0">
       <arguments>
        <fixed>
          <arg name="AnnotationID" />
        </fixed>
        <keyword>
        </keyword>
      </arguments>
      <example>http://../OLP/HAS/1.0/FetchAnnotation/{example}</example>
    </version>
   </versions>
  </verb>
 </DescribeVerb>
```

BeginSession

```
Version: 1.0
Fixed args: login
Keyword Args: grouplist, projectlist
Return MIME type: text/xml
Return Status Codes: 200, 400
```

A session with the annotation server for the specified user is started and a unique session id is returned. The list of groups and/or projects to which the user has access is specified. These lists will control access of the user to annotations for the current session (until EndSession is called)

Example Request:
/OLP/HAS/1.0/BeginSession/maria@ics.forth.gr?grouplist=scholnet

Example Response (success):
```
<?xml version="1.0" encoding="UTF-8"?>
<BeginSession version="1.0" login=maria status="OK">
        <sessionid>HAS987</sessionid>
</BeginSession>
```

Example Response (failure):
```
<?xml version="1.0" encoding="UTF-8"?>
<BeginSession version="1.0" login=maria status="Error: A session cannot be started"/>
```

EndSession

```
Version: 1.0
Fixed args: sessionid
Keyword Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

The session with the annotation server with *sessionid* is ended.

Example Request:
/OLP/HAS/1.0/EndSession/HAS987

Example Response (success):
```
<?xml version="1.0" encoding="UTF-8"?>
<EndSession version="1.0" sessionid="HAS987" status="OK"/>
```

Example Response (failure):
```
<?xml version="1.0" encoding="UTF-8"?>
<EndSession version="1.0" sessionid="HAS987" status="Error: The session cannot be ended. Service is still
processing."/>
```

FetchAnnotation

```
Version: 1.0
Fixed args: sessionid, annotation_handle
Keyword Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

The annotation with *annotation_handle* is retrieved from the annotation server and an xml document with all relevant information is returned if the user who started the session has access to this annotation.

Example Request:
/OLP/HAS/1.0/FetchAnnotation/ HAS987/ics.forth/ ann01

Example Response (success):
```
<?xml version="1.0" encoding="UTF-8"?>
  <FetchAnnotation version="1.0" sessionid="HAS987" annotation_handle ="ics.forth/ ann01" status="OK">
      <annotation>
            <handle>ics.forth/ann01</handle>
            <description>
                        <type>Agreement</type>
                        <subject>Testing</subject>
```

```
                    <text>"testing annotation service"</text>
                    <program>Scholnet</program>
          </description>
          <userinfo>
                    <author>Manolis Tzobanakis</author>
                    <project>Scholnet</project>
                    <group>ISL</group>
          </userinfo>
          <links>
                     <linktype>agrees with</linktype>
                     <tovalue>Doc02102001</tovalue>
                     <linktype>disagrees with</linktype>
                     <tovalue>Doc01102001</tovalue>
           </links>
          <date>2001-10-2</date>
          </annotation>
</FetchAnnotation>
```

Example Response (failure):

```
<?xml version="1.0" encoding="UTF-8"?>
<FetchAnnotation version="1.0" sessionid="HAS987" annotation_handle ="ics.forth/ ann01" status="Error:
The annotation_handle does not exist in the annotation server."/>
```

CreateAnnotation

```
Version: 1.0
Fixed args: sessionid, document_handle
Fixed_post args: file
Keyword Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

This verb creates an annotation for the document specified with *document_handle*. The annotation is transmitted as an HTTP POST request where the input stream is a MIME type text/xml. The verb delivers a confirmation message.

Example Request:
/OLP/HAS/1.0/CreateAnnotation/HAS987/ics.forth/TR001

In addition an input stream should be supplied with the POST that includes this request.

Example POST request:

```
<?xml version="1.0" encoding="UTF-8"?>
 <CreateAnnotation version="1.0">
          <annotation>
          <description>
                    <type>Agreement</type>
                    <subject>Testing</subject>
                    <text>"testing annotation service"</text>
                    <program>Scholnet</program>
          </description>
          <userinfo>
                    <author>Manolis Tzobanakis</author>
                    <project>Scholnet</project>
                    <group>ISL</group>
          </userinfo>
          <links>
                     <linktype>agrees with</linktype>
                     <tovalue>Doc02102001</tovalue>
           </links>
          <date>2001-10-2</date>
          </annotation>
```

```
</CreateAnnotation>
```

UpdateAnnotation

```
    Version: 1.0
    Fixed args: sessionid, annotation_handle
    Fixed_post args: file
    Keyword Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

This verb updates the annotation *annotation_handle*. The annotation is transmitted as an HTTP POST request where the input stream is a MIME type text/xml. The verb delivers a confirmation message.

Example Request:
/OLP/HAS/1.0/UpdateAnnotation/HAS987/ics.forth/ann001
In addition an input stream should be supplied with the POST that includes this request

Example Response (failure):

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateAnnotation version="1.0" sessionid="HAS987" annotation_handle ="ics.forth/ ann01" status="Error:
User is not authorised to update the annotation"/>
```

SearchAnnotation

```
    Version: 1.0
    Fixed args: sessionid
    Fixed_post args: search_specification
    Keyword Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

This verb searches the annotation database for annotations that match the query specified in the *search_specification*. The search_specification is passed as an HTTP POST request of an XML file.
The verb returns a structured response that contains a list with basic information regarding the annotations that matched the query.

Example Request:
/OLP/HAS/1.0/SearchAnnotation/HAS987

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <SearchAnnotation version="1.0" sessionid="HAS987" status="OK">
    <description>
            A list of annotation URLs that match a search pattern.
    </description>
    <annotationlist>
            <annotation>
                    <handle>ics.forth/ann003</handle>
                    <description>
                            <subject>Testing</subject>
                            <type> Agreement </type>
                    </description>
                    <userinfo>
                            <author>Manolis Tzobanakis</author>
                    </userinfo>
            </annotation>
            <annotation>
                    <handle>ics.forth/ann045</handle>
                    <description>
```

```
                        <subject>Testing</subject>
                        <type>Rating</type>
                </description>
                <userinfo>
                        <author>Maria Theodoridou</author>
                </userinfo>
        </annotation>
    </annotationlist>
</SearchAnnotation>
```

GetRefDocuments

```
    Version: 1.0
    Fixed args: sessionid, annotation-handle
    Keyword Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

This verb returns the list of document handles referenced by the annotation with *annotation-handle* from the annotation server.

Example Request:
/OLP/HAS/1.0/GetRefDocuments/HAS987/ics.forth/ann001

Example Response (success):

```
<?xml version="1.0" encoding="UTF-8"?>
  <GetRefDocuments version="1.0" sessionid="HAS987" status="OK">
        <handle>ics.forth/ann001</handle>
        <referenced-docs>
                <handle>ics.forth/doc01</handle>
                <handle>ics.forth/doc04</handle>
                <handle>ics.forth/doc25</handle>
        </referenced-docs>
</GetRefDocuments>
```

GetAnnotationsofDocument

```
    Version: 1.0
    Fixed args: sessionid, document-handle
    Keyword Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

This verb returns the list of annotation handles that reference the document with document-*handle*.

Example Request:
/OLP/HAS/1.0/GetAnnotationsofDocument/HAS987/ics.forth/doc001

Example Response (success):

```
<?xml version="1.0" encoding="UTF-8"?>
  <GetAnnotationsofDocument version="1.0" sessionid="HAS987" status="OK">
        <handle>ics.forth/doc001</handle>
        <referenced-by-annt>
                <handle>ics.forth/annt01</handle>
                <handle>ics.forth/annt03</handle>
                <handle>ics.forth/annt10</handle>
        </referenced-by-annt>
</GetAnnotationsofDocument >
```

DeleteAnnotation

```
    Version: 1.0
    Fixed args: sessionid, annotation-handle
```

```
     Keyword Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

This verb deletes the annotation with *annotation-handle* from the annotation server and delivers a confirmation message as a receipt.

Example Request:
/OLP/HAS/1.0/DeleteAnnotation/HAS987/ics.forth/ann001

DisplayAnnotations

```
     Version: 1.0
     Fixed args: sessionid
     Keyword Args: document-handle, annotation-handle, URL, option, format
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

Generates an XML document with basic information about the annotations associated with the specified input. If option is "immediate" only the immediately associated annotations (first level annotations) are included in the returned XML stream. If option is "all", then all associated annotations (all levels of annotations) are included in the returned XML stream. Default value for option is "immediate". The optional keyword format can take the following values:

- *short* (default): only the subject, type, program and text of the annotation is included in the XML stream.
- *long*: information about the author, project, group, date are also included in the XML stream.

Example Request:
/OLP/HAS/1.0/DisplayAnnotation/HAS987?DocumentID=ics.forth/TR001&option=immediate

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
  <DisplayAnnotations version="1.0" sessionid="HAS987" status="OK">
    <annotatedObject>
          <handle>ics.forth/TR001</handle>
                <annotation>
                  <handle>ics.forth/ann001</handle>
                   <description>
                           <type>rating</type>
                           <text>10</text>
                           <program>scholnet</program>
                  </description>
                </annotation>
                <annotation>
                  <handle>ics.forth/ann002</handle>
                   <description>
                           <type>agreement</type>
                           <text>excellent paper</text>
                           <subject>architectural specifications</subject>
                  </description>
                </annotation>
    </annotatedObject>
  </DisplayAnnotations>
```

18. MULTILINGUAL THESAURUS SERVICE

The Multilingual Thesaurus Service is an independent service based on the SIS Thesaurus Management System (SIS-TMS) [http://www.ics.forth.gr/proj/isst/Systems/sis-tms.html]. The SIS-TMS consists of a tool to develop multilingual thesauri and a terminology server for cataloguers and for distributed access to heterogeneous electronic collections. The Multilingual Thesaurus Service communicates with Scholnet via the OpenDLib Protocol.

18.1 *STATE*

The Multilingual Thesaurus Service will provide the capability to view and/or update the SIS-TMS database through XML documents. The basic notion of SIS-TMS is a thesaurus concept which is described by an XML document (see Appendix D). The user will be able to view and/or edit this document.

18.2 *MULTILINGUAL THESAURUS VERBS*

In this section, we present the functionality provided by the Multilingual Thesaurus Service for the communication with the Scholnet system.
This verb set is divided into two sub-sets: Service Information Verb and Service Specific Verb as reported in the following table:

Service Information Verb	Service Specific Verbs	Administration Specific Verbs
Identify	BeginSession	MoveHierarchy
ListVerbs	EndSession	CreateNewHierarchy
DescribeVerb	CreateTerm	CreateHierarchy
	UpdateTerm	DeleteHierarchy
	RenameTerm	CreateFacet
	DeleteTerm	ReleaseThesaurus
	ListTerms	
	ListTerm	
	ListTranslations	
	ListThesauri	
	SetThesaurus	
	SearchTerm	

Identify

```
    Version: 1.0
    Fixed args: none
    Keyword Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400, 501
```

This verb returns a structured response containing the name and other information about the service.

Example Request:
/OLP/MTS/1.0/Identify

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<Identify version="1.0">
    <serviceName>Multilingual Thesaurus Service</serviceName >
    <baseURL>http://thesaurusserver.ics.forth.gr:8080</baseURL>
    <protocolVersion>1.0</protocolVersion>
    <adminEmail>martin@ics.forth.gr</adminEmail>
    <description>
        <contentDescription>
            a human readable description of the content stored in the Multilingual Thesaurus Service
        </contentDescription>
    </description>
</Identify>
```

ListVerbs

```
    Version: 1.0
    Fixed args: none
    Keyword Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

Returns a structured response containing the name of the verbs defined by this service.

Example Request:
/OLP/MTS/1.0/ListVerbs

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListVerbs version="1.0">
    <verb>Identify</verb>
    <verb>ListVerbs</verb>
    <verb>DescribeVerb</verb>
    <verb>BeginSession</verb>
    <verb>EndSession</verb>
    <verb>CreateTerm</verb>
    <verb>UpdateTerm</verb>
    <verb>RenameTerm</verb>
    <verb>DeleteTerm</verb>
    <verb>ListTerms</verb>
    <verb>ListTerm</verb>
    <verb>ListTranslations</verb>
    <verb>ListThesauri</verb>
    <verb>SetThesaurus</verb>
    <verb>SearchTerm</verb>
    <verb>MoveHierarchy</verb>
    <verb>CreateNewHierarchy</verb>
    <verb>CreateHierarchy</verb>
    <verb>DeleteHierarchy</verb>
    <verb>CreateFacet</verb>
    <verb>ReleaseThesaurus</verb>
</ListVerbs>
```

DescribeVerb

```
    Version: 1.0
    Fixed args: verb
    Keyword Args: version
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

Returns a structured response that contains a list in which each element of the list provides information on a version of the specified verb that is supported by this service. The following information may be provided at the verb or version level.
description, description of the verb or a specific version
note, information pertaining to the verb or a specific version
Each element of the list contains the following information:
version number of the verb.

arguments, a list of the names of the fixed and keyword arguments, if any, accepted by the verb in that version.
example template of request to this repository, with fixed arguments indicated in brackets
returns, optional, contains information about response format.
Note that a service may implement more than one version of a verb.

Example Request:
/OLP/MTS/1.0/DescribeVerb/DisplayTerm

Example Response:
```
<?xml version="1.0" encoding="UTF-8"?>
  <DescribeVerb version="1.0">
    <verb name="ListTerm">
      <description>
              Returns an XML document that contains a description of the term corresponding to the given
              target term.
      </description>
      <versions>
        <version id="1.0">
          <arguments>
           <fixed>
             <arg name="targetterm" />
           </fixed>
           <keyword>
           </keyword>
         </arguments>
         <example>http://../OLP/MTS/1.0/ListTerm/{example}</example>
        </version>
      </versions>
    </verb>
  </DescribeVerb>
```

BeginSession

```
Version: 1.0
Fixed args: login, thesaurusname
Keyword Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

A session with the thesaurus server for the specified user is started. If the MTS is able to link the session to the thesaurus with the specified *thesaurusname* a unique sessionid will be returned.

Example Request:
/OLP/MTS/1.0/BeginSession/maria@ics.forth.gr/ACMenglish

Example Response (success):
```
<?xml version="1.0" encoding="UTF-8"?>
  <BeginSession version="1.0" thesaurusname="ACMen" status="OK">
              <sessionid>MTS00234</sessionid>
  </BeginSession>
```

Example Response (failure):
```
<?xml version='1.0' encoding='UTF-8'?>
<BeginSession version='1.0' thesaurusname='ACMen' status='Error: ErrorMessage'/>
```

EndSession

```
Version: 1.0
Fixed args: sessionid
Keyword Args: none
```

```
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

The session with *sessionid* with the thesaurus server is ended.

Example Request:
/OLP/MTS/1.0/EndSession/MTS000001

Example Response (success):

```
<?xml version='1.0' encoding='UTF-8'?>
  <EndSession version='1.0' sessionid='MTS000001' status='OK'/>
```

Example Response (failure):

```
<?xml version='1.0' encoding='UTF-8'?>
    <EndSession version='1.0' sessionid='MTS000001' status='Error:  Session ID does not exist'/>
```

CreateTerm

```
     Version: 1.0
     Fixed args: sessionid, termname, broaderterm
     Keyword Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

This verb creates a new term in the thesaurus. The new term is added to the knowledge base and is associated with the given broader term with a BT relation. It is also classified in its broader term hierarchies

Example Request:
/OLP/MTS/1.0/CreateTerm/MTS000001/Data mining/Database Applications

Example Response (success):

```
<?xml version='1.0' encoding='UTF-8'?>
    <CreateTerm version='1.0' sessionid='MTS000001' status='OK'>
          <new term>Data mining</new term>
    </CreateTerm>
```

Example Response (failure):

```
<?xml version='1.0' encoding='UTF-8'?>
  <CreateTerm version='1.0' sessionid='MTS000001' status='ErrorMessage'/>
```

UpdateTerm

```
     Version: 1.0
     Fixed args: sessionid, targetterm
     Fixed_post args: file
     Keyword Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400, 503
```

Updates an existing term in the thesaurus. This is transmitted as an HTTP POST request where the input stream is a MIME type text/xml.

Example Request:
/OLP/MTS/1.0/UpdateTerm/ MTS000456/Data mining

In addition an input stream should be supplied with the POST that includes this request.

RenameTerm

```
     Version: 1.0
     Fixed args: sessionid, oldName, newName
     Keyword Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

Renames an existing term in the thesaurus. If the *newName* is an existing term in the thesaurus, then the request will fail.

Example Request:
/OLP/MTS/1.0/RenameTerm/MTS000001/databases/databases and GIS.

Example Response (success):

```
<?xml version='1.0' encoding='UTF-8'?>
    <RenameTerm version='1.0' sessionid='MTS000001' status='OK'>
        <old name>databases</old name>
        <new name>databases and GIS</new name>
    </RenameTerm>
```

Example Response (failure):

```
<?xml version='1.0' encoding='UTF-8'?>
  <RenameTerm version='1.0' sessionid='MTS000001' status='Error: Cannot rename term since the new
name already exists'/>
```

DeleteTerm

```
     Version: 1.0
     Fixed args: sessionid, targetterm
     Keyword Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

Example Request:
/OLP/MTS/1.0/DeleteTerm/ MTS000001/Data mining

Example Response (success):

```
<?xml version='1.0' encoding='UTF-8'?>
  <DeleteTerm version='1.0' sessionid='MTS000001' status='OK'>
        <deleted term>Data mining</deleted term>
  </DeletedTerm>
```

Example Response (failure):

```
<?xml version='1.0' encoding='UTF-8'?>
  <DeleteTerm version='1.0' sessionid='MTS000001' status='Error:ErrorMessage'/>
```

ListTerms

```
     Version: 1.0
     Fixed args: sessionid, typeofdisplay, option
     Keyword Args: target, typeofterm
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

A list of terms can be obtained in several ways depending on the level of a term and the type of display. *Typeofdisplay* can take one of two possible values: *alphabetic, hierarchical*. Terms may be listed either alphabetically or hierarchically. In the later case, the display may be either textual in the form of an indented list or graphical in the form of a tree.

In any type of display, the user may select one of the following *options*:

ALLTERMS: Display all terms independently of their level in the Classification System according to their type (obsolete, new, revised, etc.) which is given in the *typeofterm* keyword.
ALLTERMSBYHIER: Display all terms of a specified hierarchy according to their type (obsolete, new, revised etc.) which is given in the *typeofterm* keyword. The name of the hierarchy is given in the *target* keyword.

TOPHIER: Display top level hierarchies, which consists of the 11 first level nodes of the ACM Classification System. The *typeofdisplay* can only be alphabetic with this option.
ALLNT: Display all levels below a term. The name of the term is given in the *target* keyword.
ALLBT: Display all levels above a term. The name of the term is given in the *target* keyword.

Example Request:
/OLP/MTS/1.0/ListTerms/ MTS000456/alphabetic/ALLTERMS

Example Response (success):
```
<?xml version='1.0' encoding='UTF-8'?>
<ListTerms version='1.0' sessionid='MTS000456' status='OK'>
        <term>
                <Name>Abstract data types</Name>
                <Type>ACMenDescriptor</Type>
                <Hierarchy>Software</Hierarchy>
        </term>
   <term>
                <Name>Abstracting methods</Name>
                <Type>ACMenDescriptor</Type>
                <Hierarchy>Information Systems</Hierarchy>
        </term>
        <term>
                <Name>Abuse and crime involving computers</Name>
                <Type>ACMenNewDescriptor</Type>
                <Hierarchy>Computing Milieux</Hierarchy>
        </term>
        ……
</ListTerms>
```

Example Response (failure):
```
<?xml version='1.0' encoding='UTF-8'?>
  <ListTerms version='1.0' sessionid='MTS000456' status='Error:ErrorMessage'/>
```

Example Request:
/OLP/MTS/1.0/ListTerms/ MTS000456/hierarchical/ALLTERMS

Example Response (success):
```
<?xml version='1.0' encoding='UTF-8'?>
<ListTerms version='1.0' sessionid='MTS000456' status='OK'>
   <term>
       <Name>Computing Methodologies</Name>
       <Type>ACMenDescriptor</Type>
       <Hierarchy>Computing Methodologies</Hierarchy>
       <Narrower_Term>
           <Concept_Name>GENERAL</Concept_Name>
           <Concept_Name>SYMBOLIC AND ALGEBRAIC MANIPULATION</Concept_Name>
           <Concept_Name>ARTIFICIAL INTELLIGENCE</Concept_Name>
           ……..
       </Narrower_Term>
   </term>
   <term>
       <Name>GENERAL</Name>
       <Type>ACMenDescriptor</Type>
       <Hierarchy>Computing Methodologies</Hierarchy>
   </term>
   <term>
       <Name>SYMBOLIC AND ALGEBRAIC MANIPULATION</Name>
       <Type>ACMenRevisedDescriptor</Type>
       <Hierarchy>Computing Methodologies</Hierarchy>
       <Narrower_Term>
```

```
                <Concept_Name>General</Concept_Name>
                <Concept_Name>Expressions and Their Representation</Concept_Name>
                <Concept_Name>Algorithms</Concept_Name>
            ……..
            </Narrower_Term>
    </term>
</ListTerms>
```

Example Request:
/OLP/MTS/1.0/ListTerms/ MTS000456/alphabetic/ALLTERMSBYHIER?target=Computing Methodologies

Example Response (success):

```
<?xml version='1.0' encoding='UTF-8'?>
<ListTerms version='1.0' sessionid='MTS000456' status='OK'>
    <term >Algebraic algorithms</term>
    <term >Algorithms</term>
    <term>Analogies</term>
    <term>Antialiasing</term>
    …….
</ListTerms>
```

Example Request:
/OLP/MTS/1.0/ListTerms/ MTS000456/alphabetic/TOPHIER

Example Response (success):

```
<?xml version='1.0' encoding='UTF-8'?>
<ListTerms version='1.0' sessionid='MTS000456' status='OK'>
    <hierarchy>Computer Applications</hierarchy>
    <hierarchy>Computer Systems Organization</hierarchy>
    <hierarchy>Computing Methodologies</hierarchy>
    <hierarchy>Computing Milieux</hierarchy>
    <hierarchy>Data</hierarchy>
    <hierarchy>General Literature</hierarchy>
    <hierarchy>Hardware</hierarchy>
    <hierarchy>Information Systems</hierarchy>
    <hierarchy>Mathematics of Computing</hierarchy>
    <hierarchy>Software</hierarchy>
    <hierarchy>Theory of Computation</hierarchy>
</ListTerms>
```

Example Request:
/OLP/MTS/1.0/ListTerms/MTS000456/hierarchical/ALLNT?target=SYMBOLIC AND ALGEBRAIC MANIPULATION

Example Response (success):

```
<?xml version='1.0' encoding='UTF-8'?>
<ListTerms version='1.0' sessionid='MTS000456' status='OK'>
    <term>
        <name> SYMBOLIC AND ALGEBRAIC MANIPULATION</name>
        <ntterm>General</ntterm>
    </term>
    <term>
          <name> SYMBOLIC AND ALGEBRAIC MANIPULATION </name>
          <ntterm>Expressions and Their Representation</ntterm>
    </term>
    <term>
          <name>Expressions and Their Representation</name>
          <ntterm>Representations (general and polynomial)</ntterm>
    </term>
    <term>
```

```
            <name>Expressions and Their Representation</name>
            <ntterm>Simplification of expressions</ntterm>
    </term>
    <term>
            <name>SYMBOLIC AND ALGEBRAIC MANIPULATION</name>
            <ntterm>Algorithms</ntterm>
    </term>
    <term>
            <name>Algorithms</name>
            <ntterm>Algebraic algorithms</ntterm>
    </term>
    <term>
            <name>Algorithms</name>
            <ntterm>Analysis of algorithms</ntterm>
    </term>
    <term>
            <name>Algorithms</name>
            <ntterm>Nonalgebraic algorithms</ntterm>
    </term>
            ……
</ListTerms>
```

Example Request:
/OLP/MTS/1.0/ListTerms/ MTS000456/alphabetic/ALLBT?target=Algorithms
Example Response (success):

```
<?xml version='1.0' encoding='UTF-8'?>
<ListTerms version='1.0' sessionid='MTS000456' status='OK'>
            <term>Computing Methodologies</term>
    <term>SYMBOLIC AND ALGEBRAIC MANIPULATION</term>
</ListTerms>
```

ListTerm

```
Version: 1.0
Fixed args: sessionid, targetterm
Keyword Args: option
Return MIME type: text/xml
Return Status Codes: 200, 400
```

For *targetterm,* retrieves all the information regarding the term. If option is LONG then all the information regarding the term is returned. If option is SHORT the thesaurus is only checked to see whether the target term is contained in the thesaurus of not. The default value of option is LONG.

Example Request:
/OLP/MTS/1.0/ListTerm/ MTS000001/Algorithms

Example Response (success):

```
<?xml version='1.0' encoding='UTF-8'?>
<ListTerm version='1.0' sessionid='MTS000001' status='OK'>
    <Concept>
        <Definition>
            <Name>Algorithms</Name>
            <Scope_Note> scope note of term</Scope_Note>
            <Thesaurus>ACMen</Thesaurus>
            <Type>ACMenDescriptor</Type>
            <Hierarchy>Computing Methodologies</Hierarchy>
        </Definition>
        <Intra_Thesaurus>
            <Broader_Term>
                <Concept_Name>SYMBOLIC AND ALGEBRAIC MANIPULATION</Concept_Name>
            </Broader_Term>
            <Narrower_Term>
```

```
                <Concept_Name>Algebraic algorithms</Concept_Name>
            </Narrower_Term>
            <Narrower_Term>
                <Concept_Name>Analysis of algorithms</Concept_Name>
            </Narrower_Term>
            <Narrower_Term>
                <Concept_Name>Nonalgebraic algorithms</Concept_Name>
            </Narrower_Term>
            <Related_Term>
                <Concept_Name>Numerical Algorithms and Problems</Concept_Name>
                <Concept_Name>Nonnumerical Algorithms and Problems</Concept_Name>
            </Related_Term>
        </Intra_Thesaurus>
    </Concept>
  </ListTerm>
```

Example Response (failure):

```
<?xml version='1.0' encoding='UTF-8'?>
   <ListTerm version='1.0' sessionid='MTS000001' status='Error:ErrorMessage'/>
```

Example Request:
/OLP/MTS/1.0/ListTerm/ MTS000001/Algorithms?option=SHORT

Example Response (success):

```
<?xml version='1.0' encoding='UTF-8'?>
 <ListTerm version='1.0' sessionid='MTS000001' status='OK'>
    <Concept>
       <Definition>
            <Name>Algorithms</Name>
            <Type>ACMenDescriptor</Type>
            <Hierarchy>Computing Methodologies</Hierarchy>
       </Definition>
    </Concept>
</ListTerm>
```

ListTranslations

```
Version: 1.0
Fixed args: sessionid, thesaurusname
Fixed_post args: list of targetterms
Keyword Args: option
Return MIME type: text/xml
Return Status Codes: 200, 400
```

The list of *targetterms* is passed through an HTTP POST request. For each *targetterm* in the list, all links of the *targetterm* to equivalent terms in the *thesaurusname* are collected. If option is BT then the broader terms of *targetterm* with equivalent terms in *thesaurusname* are collected. If option is NT then the narrower terms of *targetterm* with equivalent terms in *thesaurusname* are collected. If option is BOTH then both broader and narrower terms of *targetterm* with equivalent terms in *thesaurusname* are collected.

Example Request:
/OLP/MTS/1.0/ListTranslations/ MTS000456/ACMde?option=BOTH

An input stream should be supplied with the POST that includes this request:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListTranslations version="1.0">
    <term>Applications and Expert Systems</term>
    <term>Speech recognition and synthesis</term>
```

```
</ ListTranslations >
```

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ListTranslations version="1.0" status="OK">
    <translatedterm>
        <translation>
            <term>Applications and Expert Systems</term>
            <equivalence>
                <Union>
                    <Intersection>
                        <Concept_Name>Anwendung</Concept_Name>
                        <Concept_Name>Kuenstliche Intelligenz</Concept_Name>
                    </Intersection>
                    <Concept_Name>Experten System</Concept_Name>
                </Union>
            </equivalence>
        </translation>
        <btterm>
            <translation>
                <term>ARTIFICIAL INTELLIGENCE</term>
                <equivalence>
                    <Concept_Name>Kuenstliche Intelligenz</Concept_Name>
`               </equivalence>
            </translation>
        </btterm>
        <ntterm>
            <translation>
            .....
            </translation>
        </ntterm>
    </translatedterm>
    <translatedterm>
        <translation>
            <term>Speech recognition and synthesis</term>
            <equivalence>
                <Union>
                    <Concept_Name>Spracherkennung</Concept_Name>
                    <Concept_Name>Spracheingabe</Concept_Name>
                    <Concept_Name>Sprachausgabe</Concept_Name>
                </Union>
            </equivalence>
        </translation>
        <btterm>
            <translation>
                .....
            </translation>
        </btterm>
        <ntterm>
            <translation>
                .....
            </translation>
        </ntterm>
    </translatedterm>
</ListTranslations>
```

ListThesauri

```
Version: 1.0
Fixed args: sessionid
Keyword Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

This verb lists all the available thesauri in the system.

Example Request:
/OLP/MTS/1.0/ListThesauri/ MTS000001

Example Response (success):

```
<?xml version="1.0" encoding="UTF-8"?>
<ListThesauri version="1.0" sessionid='MTS000001' status="OK">
   <thesaurus>ACMen</thesaurus>
   <thesaurus>ACMit</thesaurus>
   <thesaurus>ACMgr</thesaurus>
   <thesaurus>ACMse</thesaurus>
   <thesaurus>ACMde</thesaurus>
   <thesaurus>ACMcz</thesaurus>
   <thesaurus>ACMfr</thesaurus>
</ListThesauri>
```

Example Response (failure):

```
<?xml version='1.0' encoding='UTF-8'?>
  <ListThesauri version='1.0' sessionid='MTS000001' status='Error: Failed to List Thesauri'/>
```

SetThesaurus

```
Version: 1.0
Fixed args: sessionid, thesaurusname
Keyword Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

The user may use this verb in order to change the thesaurus with which he is currently working. He has to provide the name of the new thesaurus as argument.

Example Request:
/OLP/MTS/1.0/SetThesaurus/MTS000001/greek

Example Response (success):

```
<?xml version='1.0' encoding='UTF-8'?>
  <SetThesaurus version='1.0' sessionid='MTS000001' status='OK'>
        <oldthesaurus>ACMfr</oldthesaurus>
        <newthesaurus>ACMgr</newthesaurus>
  </SetThesaurus>
```

Example Response (failure):

```
<?xml version='1.0' encoding='UTF-8'?>
   <SetThesaurus version='1.0' sessionid='MTS000001' status='Error: greek is not  a valid    Thesaurus'/>
```

SearchTerm

```
Version: 1.0
Fixed args: sessionid, string
Keyword Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Selects thesaurus terms whose name matches (completely or partially) the given pattern. Both

precise match and left/right truncation is supported:
***string* :** Matches the terms that end by string
string* : Matches the names that start by string
***string** or **string* :** Matches the names that contain the substring string

Example Request:
/OLP/MTS/1.0/SearchTerm/ MTS000001/*method*

Example Response (success):
```
<?xml version="1.0" encoding="UTF-8"?>
<SearchTerm version="1.0" sessionid=' MTS000001' status="OK">
    <term>Access methods</term >
    <term>Abstracting methods</term >
    <term>Indexing methods</term >
    <term>Evaluation/methodology</term >
    <term>Theory and methods</term >
     …..
</ SearchTerm>
```

Example Response (failure):
```
<?xml version='1.0' encoding='UTF-8'?>
  <SearchTerm version='1.0' sessionid='MTS000001' status='Error: ErrorMessage'/>
```

MoveHierarchy

```
Version: 1.0
Fixed args: sessionid, targetterm, currenthierarchy, newhierarchy,
NewBTterm
Keyword Args: Option
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Depending on the value of Option:
MOVE_NODE_ONLY: The target term is detached from the CurrentHierarchy and classified in the new hierarchy. A broader term relation is established between the target term and the NewBTterm.
MOVE_NODE_AND_SUBTREE: The target term and its subtree of broader term relations are detached from the CurrentHierarchy and are reclassified in the new hierarchy. A broader term relation is established between the target term and the NewBTterm.
CONNECT_NODE_AND_SUBTREE: The targetterm and its subtree of broader term relations are NOT detached from the CurrentHierarchy and are classified in multiple in the new hierarchy. A broader term relation is established between the target term and the NewBTterm.

Example Request:
/OLP/MTS/1.0/MoveHierarchy/MTS000001/User Issues/Information Systems/Mathematics of Computing/MATHEMATICAL SOFTWARE?option=CONNECT_NODE_AND_SUBTREE

Example Response (success):
```
<?xml version='1.0' encoding='UTF-8'?>
  <MoveHierarchy version='1.0' sessionid='MTS000001' status='OK'>
           <targetterm>User Issues</targetterm>
           <currenthierarchy>Information Systems</currenthierarchy>
           <newhierarchy>Mathematics of Computing</newhierarchy>
           <newbtterm>MATHEMATICAL SOFTWARE</newbtterm>
  </MoveHierarchy>
```

Example Response (failure):
```
<?xml version='1.0' encoding='UTF-8'?>
   <MoveHierarchy version='1.0' sessionid='MTS000001' status='ErrorMessage'/>
```

CreateNewHierarchy

```
     Version: 1.0
     Fixed args: sessionid, hierarchyname, facetname
     Keyword Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

Example Request:
/OLP/MTS/1.0/CreateNewHierarchy/MTS0000001/Hardware/ACMFacet

Example Response (success):
```
<?xml version='1.0' encoding='UTF-8'?>
  <CreateNewHierarchy version='1.0' sessionid='MTS000001' status='OK'>
        <newhierarchy>Hardware</newhierarchy>
        <facet>ACMFacet</facet>
  </CreateNewHierarchy>
```

Example Response (failure):
```
<?xml version='1.0' encoding='UTF-8'?>
  <CreateNewHiearchy version='1.0' sessionid='MTS000001' status='Error: ErrorMessage'/>
```

CreateHierarchy

```
     Version: 1.0
     Fixed args: sessionid, hierarchyname
     Keyword Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

Adds a set of terms as instances under an existing hierarchy in the thesaurus. The set of terms is transmitted as an HTTP POST request where the input stream is a MIME type text/xml.

Example Request:
/OLP/MTS/1.0/CreateHierarchy/ MTS000456/Hardware

An input stream should be supplied with the POST that includes this request:

```
<?xml version="1.0" encoding="UTF-8"?>
<CreateHierarchy version="1.0">
    <term>GENERAL</term >
    <term>CONTROL STRUCTURES AND MICROPROGRAMMING</term >
    <term>ARITHMETIC AND LOGIC STRUCTURES</term >
    <term>MEMORY STRUCTURES</term >
     …..
</CreateHierarchy>
```

DeleteHierarchy

```
     Version: 1.0
     Fixed args: sessionid, hierarchyname
     Keyword Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400
```

Deletes the hierarchy with name *hierarchyname* from the thesaurus. All objects under the hierarchy that are not under another hierarchy are also deleted. Any links from the objects that are being deleted and any objects pointed by these links are also deleted – if they can be deleted (checked by the Semantic Checker).

Example Request:

/OLP/MTS/1.0/DeleteHierarchy/ MTS000456/Hardware

CreateFacet

```
Version: 1.0
Fixed args: sessionid, facetname
Keyword Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 404
```

Creates a new facet in the thesaurus with name *facetname.*

Example Request:
/OLP/MTS/1.0/CreateFacet/ MTS000001/ACMFacet

Example Response (success):
```
<?xml version='1.0' encoding='UTF-8'?>
  <CreateFacet version='1.0' sessionid='MTS000001' status='OK'>
        <facet>ACMFacet</facet>
  </CreateFacet>
```

Example Response (failure):
```
<?xml version='1.0' encoding='UTF-8'?>
  <CreateFacet version='1.0' sessionid='MTS000001' status='Error: ErrorMessage'/>
```

ReleaseThesaurus

```
Version: 1.0
Fixed args: sessionid, thesaurusname
Keyword Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Releases a new version of the thesaurus with name *thesaurusname*.

Example Request:
/OLP/MTS/1.0/ReleaseThesaurus/ MTS000456/ACMgr

19. USER INTERFACE

The SCHOLNET User Interface can be considered as a finite state machine. Each state of the machine defines a frame set according to the current navigation position and context. For instance, if users decids to Browse for a text document within a specific collection, their menu selections will define the current state of the machine, which can be seen as a three-dimensional coordinate (Collection, Function, Document Type). Also their identity will be considered. The initial state of the machine is used to visualize the "entry page" of SCHOLNET. It consists of an HTML page divided into 5 frames (see the picture below). The top frame contains the choices available for the dimension "document type", which are "All" for all types of documents, "Text" for only text documents, "Multimedia" for Multimedia documents and "Annotations" for the annotations on documents. The left–hand side frame contains the choices for the dimension "Collection". The number of entries in this menu is dynamic, since the available collections in SCHOLNET can be changed and extended by a collection administrator. The right-hand side frame contains the choices for the type of function a user intends to perform. The functions are: "Search", "Browse", "Edit", "Submit" and "Administrate". The frame in the top left corner only contains a link for invoking the login procedure. Therefore, users can use the system without being logged in, but with only a limited set of functionality available. They will be treated as a guest user. The large frame at the centre of the browser window is called "focus" and will display all the main functionality of the system, according to the choices selected in the other frames. For instance, if a user selects the search function in the function menu, a specific collection in the collection menu, as well as the "text" button in the document type menu, the focus will display a search form for text documents within the chosen collection. Once the search form has been filled and submitted, the focus will then also contain the retrieved results of the search procedure.

Login	All Text Multimedia Annotations	
Collection1 Collection2 Collection3 . . .	FOCUS	Search Browse Edit Submit Administrate

Each state of the underlying finite state machine also implies the availability of a specific functionality. It may happen that a particular combination of menu choices excludes and therefore narrows the possibility to select choices in other menus. For instance, it could be specified that a given collection does not contain multimedia documents. Choosing this Collection implies that the Multimedia button in the top frame will be disabled. The context is also defined by the current user profile. A guest user may not be allowed to submit a document, consequently the "submit" button will not be active for that user. Moreover, if the authenticated user (i.e. a user who succeeds in the login procedure) is not an administrator, the "Administrate" function will not be accessible. The kind of administration tools that have to be shown after pressing the "Administrate" button will depend on the profile of the user logged in. Every kind of selection within the menus takes the current state and context into account. This means, for instance, that if a user is browsing a collection for multimedia documents, pressing the "search" button will imply performing a search only within the sub-part of the collection reached by the browsing process in the previous step.

19.1 *STATE*

In this section we describe the data structure, its abstract type, and give a brief textual description of each object managed by the SCHOLNET User Interface Service.

Frame Set

A Frame Set is the visualization Unit for the SCHOLNET User Interface. Each service request invokes a new Frame Set according to the resulting context. The Frame Set consists of an HTML-document describing the hierarchical arrangement of the frames. Each frame described is an HTML-document.

19.2 *SCHOLNET USER INTERFACE VERBS*

In this section, we present the set of user interface verbs. This set is divided into sub-sets as reported in the following table:

VERBS	**INCLUDING SESSION DATA**	**GENERATE**	- generateFrameSet - generateTopicForm - generateSearchForm - generateAnnotationForm - generateSubmissionForm generateUserProfileEditForm generateGroupMaintenanceForm generateGroupCreationForm generateTermEditForm
		SUBMIT	- submitTopicForm - submitSearchForm - sendWithdrawRequest - submitAnnotationForm - submitSubmissionForm - submitGroupChanges - submitGroupCreationForm - submitThesaurusSearch - performThesaurusChange
		DISPLAY	- displayDocumentStructure - displayGroupInfo - displayThesaurus - displayTermTranslation
	WITHOUT SESSION DATA	**GENERATE**	- generateRegistrationForm - generateKeyFrameMosaic - generateTimeIntervalForm
		SUBMIT	- submitRegistrationForm
		DISPLAY	- displayBrowseResult - displaySearchResult - displayAnnotation - displayMetadata - displayMMDocument - downloadMMDocument - displayTextDocument - downloadTextDocument
	SESSION MANAGEMENT		- login - initialize

Identify

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Returns a structured response containing the name and other information about the service.

Example Request:
/OLP/UserInterface/1.0/Identify

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Identify SYSTEM "P:\Projects\SCHOLNET\XML\identify.dtd">
<Identify version="1.0">
        <service-name>UserInterface</service-name>
        <base-url>http://ipsi.fraunhofer.de:8119</base-url>
        <protocol-version>1.0</protocol-version>
        <admin-email>frommholz@ipsi.fraunhofer.de</admin-email>
        <descriptions>
                <content-description>
The User Interface is responsible for interaction with the user. Most of the underlying functionality is called
                                        and integrated by the UI.
                </content-description>
        </descriptions>
</Identify>
```

ListVerbs

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400
```

Returns a structured response containing the name of the verbs defined by this service.

Example Request:
/OLP/UserInterface/1.0/ListVerbs

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ListVerbs SYSTEM "P:\Projects\SCHOLNET\XML\listverbs.dtd">
<ListVerbs version="1.0">
        <verb>GenerateFrameSet</verb>
        <verb>GenerateTopicForm</verb>
        <verb>GenerateSearchForm</verb>
        <verb>GenerateAnnotationForm</verb>
        <verb>GenerateSubmissionForm</verb>
        <verb>GenerateKeyFrameMosaic</verb>
        <verb>GenerateTimeIntervalForm</verb>
        <verb>SubmitTopicForm</verb>
        <verb>SubmitSearchForm</verb>
        <verb>SendWithdrawRequest</verb>
        <verb>SubmitAnnotationForm</verb>
        <verb>SubmitSubmissionForm</verb>
        <verb>DisplayDocumentStructure</verb>
        <verb>GenerateRegistrationForm</verb>
        <verb>SubmitRegistrationForm</verb>
        <verb>DisplayBrowseResults</verb>
        <verb>DisplaySearchResults</verb>
        <verb>DisplayAnnotation</verb>
        <verb>DisplayMetadata</verb>
        <verb>DisplayMMDocument</verb>
        <verb>DisplayTextDocument</verb>
        <verb>DownloadTextDocument</verb>
        <verb>Login</verb>
        <verb/>
</ListVerbs>
```

DescribeVerb

```
    Version: 1.0
    Fixed args: verb
    Optional Args: version
    Return MIME type: text/xml
    Return Status Codes: 200, 400
```

Returns a structured response that contains a list in which each element of the list provides information on a version of the specified verb that is supported by this service. The following information may be provided at the verb or version level.

- *description*, description of the verb or a specific version
- *note*, information pertaining to the verb or a specific version

Each element of the list contains the following information:

- *version* number of the verb.
- *arguments*, a list of the names of the fixed and keyword arguments, if any, accepted by the verb in that version.
- *example* template of request to this repository, with fixed arguments indicated in brackets
- *returns*, optional, contains information about response format.

Note that a service may implement more than one version of a verb.

Example Request:

/OLP/UserInterface/1.0/DescribeVerb/GenerateFrameSet

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DescribeVerb SYSTEM "P:\Projects\SCHOLNET\XML\describeverb.dtd">
<DescribeVerb version="1.0">
  <verb name="GenerateFrameSet">
    <description>Generates the frame set.</description>
    <versions>
      <version id="1.0">
        <returns>text/html</returns>
        <arguments>
          <keyword>
            <arg name="collectionState"/>
            <arg name="functionState"/>
            <arg name="typeState"/>
            <arg name="inititalize"/>
          </keyword>
        </arguments>
        <example>/OLP/UserInterface/1.0/generateFrameSet?initialize=yes</example>
      </version>
    </versions>
  </verb>
</DescribeVerb>
```

inizialize

```
Version: 1.0
Fixed Args: none
Optional Args: none
Return MIME type: text/html
```

This is the first verb to be called; it initializes the service. Region descriptions, search methods, meta formats, browsable fields etc. are retrieved from a number of services.

Example Request:
/OLP/UserInterface/1.0/initialize

generateFrameSet

```
Version: 1.0
Fixed Args: none
Optional Args: collectionState, functionState, typeState, initialize
Return MIME type: text/html
```

This service generates the frame set. The new state of the three menus (Collection, Function and Type) can be given. Only those states are changed which are given as arguments. If, for example, the new state of the Collection menu is given, only this will be changed; all others will be left unchanged. The state of the three menus is described in terms of a finite state machine.

initialize can have values: yes or no (default: no). If yes, the frame set will be initialized (i.e. the SCHOLNET entry-page will be visualized), regardless of any other given arguments.

Example Request:
/OLP/UserInterface/1.0/generateFrameSet?initialize=yes

generateTopicForm

```
Version: 1.0
Fixed Args: none
Optional Args: none
Return MIME type: text/html
```

Creates a form to choose topics (for the personalized dissemination service). The list of topics that can be chosen choose depends on the user as he/she may already have made selection sessions. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/generateTopicForm/

generateUserProfileEditForm

```
Version: 1.0
Fixed Args: none
Optional Args: none
Return MIME type: text/html
```

Creates a form to edit the user's profile. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/generateUserProfileEditForm/

generateGroupCreationForm

```
Version: 1.0
Fixed Args: none
```

```
    Optional Args: none
    Return MIME type: text/html
```

Generates a form to create a new group. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/generateUserProfileEditForm/

generateSearchForm

```
    Version: 1.0
    Fixed Args: collectionState, typeState
    Optional Args: none
    Return MIME type: text/html
```

Creates the search form. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/generateSearchForm/cID/tID

generateAnnotationForm

```
    Version: 1.0
    Fixed Args: DocumentID
    Optional Args: none
    Return MIME type: text/html
```

Creates the form to insert annotations. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/generateAnnotationForm/d243

generateSubmissionForm

```
    Version: 1.0
    Fixed Args: typeState
    Optional Args: none
    Return MIME type: text/html
```

Creates the form to submit a new document. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/generateSubmissionForm/tID

generateKeyFrameMosaic

```
    Version: 1.0
    Fixed Args: documentID
    Optional Args: searchList, anchorFrameID, typeOfSort
    Return MIME type: text/html
```

This service will be called when the selected MM document is available in MPEG 7 format. A frame containing an arrangement of the key frames of the document, plus some forms for performing searches etc. are displayed. A search list can be given; in this case, only those key frames containing the terms in this search list are displayed. The sequence in which the keyframes appear can be determined by time or by subject. A frame (the *anchor frame*) can be selected by the user; this frame will be displayed first and all other key frames will be ranked according to their similarity to this anchor frame.

Example Request:
/OLP/UserInterface/1.0/generateKeyFrameMosaic/d101?searchList=xml,xslt&typeOfSort=time

Generates a new key frame mosaic. These frames are sorted with respect to time; only those key frames containing the terms "xml" and "xslt" in their metadata are displayed.

generateTimeIntervalForm

```
Version: 1.0
Fixed Args: documentID
Optional Args: none
Return MIME type: text/html
```

For multimedia documents, a form is created in which the user can specify a time interval to display only a certain part of a multimedia document. The document id is needed in order to have the duration of the document.

Example Request:
/OLP/UserInterface/1.0/generateTimeIntervalForm/d101

generateTermEditForm

```
Version: 1.0
Fixed args: none
Fixed_post Args: <term data in XML>
Optional Args: none
Return MIME type: text/html
```

Generates a window within which to edit or create thesaurus terms. The term data is in XML, and an XML stream will be sent to the appropriate verbs of the Thesaurus service.

Example Request:
/OLP/UserInterface/1.0/generateTermEditWindow/<xml-data>

generateGroupMaintenanceForm

```
Version: 1.0
Fixed Args: none
Optional Args: none
Return MIME type: text/html
```

Generates a window in which groups are shown. The user can select from this list and visualize detailed information for a group and edit this group. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/generateGroupMaintenanceForm

submitTopicForm

```
Version: 1.0
Fixed Args: none
Keyword_post Args: <List of POST-Input Fields>
Return MIME type: text/html
```

This service will be called when the topic form is submitted. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/submitTopicForm?field1=value1&field2=value2&……

submitUserProfile

```
Version: 1.0
Fixed Args: none
Keyword_post Args: <List of POST-Input Fields>
Return MIME type: text/html
```

This service will be called when the user profile is submitted. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/submitUserProfileForm?field1=value1&field2=value2&……

submitGroupCreationForm

```
Version: 1.0
Fixed Args: none
Keyword_post Args: <List of POST-Input Fields>
Return MIME type: text/html
```

This service will be called when a new group is created. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/submitGroupChanges?field1=value1&field2=value2&……

submitGroupChanges

```
Version: 1.0
Fixed Args: none
Keyword_post Args: <List of POST-Input Fields>
Return MIME type: text/html
```

This service will be called when group changes are submitted. Users can be added/deleted to a group, group properties can be changed etc. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/submitGroupChanges?field1=value1&field2=value2&……

submitSearchForm

```
Version: 1.0
Fixed Args: collectionState, typeState
Keyword_post Args: <List of POST-Input Fields>
Return MIME type: text/html
```

This service will be called when the search form is submitted. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/submitSearchForm/cID/tID?field1=value1&field2=value2&……

submitThesaurusSearch

```
Version: 1.0
Fixed Args: sessionid, searchString
Optional Args: none
Return MIME type: text/html
```

This service is used to perform a search in the thesaurus. Search results are displayed. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/submitSearchForm/cID/tID?field1=value1&field2=value2&……

sendWithdrawRequest

```
Version: 1.0
Fixed Args: DocumentID
Optional Args: none
Return MIME type: text/html
```

This service will be called when the withdraw request is activated. If the user is not author of the document, the request will be sent to the administrator who is responsible for the document. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/sendWithdrawRequest/d357

submitAnnotationForm

```
Version: 1.0
Fixed Args: DocumentID
Keyword_post Args: <List of POST-Input Fields>
Return MIME type: text/html
```

This service will be called when the annotation form is submitted. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/submitAnnotationForm/d323?field1=value1&field2=value2&……

submitSubmissionForm

```
Version: 1.0
Fixed Args: collectionState, typeState
Keyword_post Args: <List of POST-Input Fields>
Return MIME type: text/html
```

This service will be called when the submission form is submitted. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/submitSubmissionForm/cID/tID?field1=value1&field2=value2&……

displayThesaurus

```
Version: 1.0
Fixed Args: none
Optional Args: thesaurusName, sessionid, typeOfDisplay, option,
               targetTerm, typeOfTerm, targetLanguage
Return MIME type: text/html
```

The thesaurus will be displayed with this verb. If no thesaurus name is given, a menu from which one can be selected is first displayed and a thesaurus session is then started.
The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/displayThesaurus?thesaurusName=acm?sessionid=acb123?typeofdisplay=HIERAR
CHICAL

See Thesaurus Service for a description of keyword arguments.

displayTermTranslation

```
Version: 1.0
Fixed Args: sessionid, targetterm, thesaurusname
Keyword Args: none
Return MIME type: text/html
```

A new window is opened in which the translation(s) of a thesaurus term are presented.

Example Request:
/OLP/UserInterface/1.0/displayTermTranslation/abc123/database/acm

displayTermInfo

```
Version: 1.0
Fixed Args: none
Optional Args: sessionid, term
Return MIME type: text/html
```

Displays detailed information for a given term. The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/displayTerminfo/abc123/computers

displayDocumentStructure

```
Version: 1.0
Fixed Args: none
Optional Args: expanded
Keyword_post Args: XMLStream
Return MIME type: text/html
```

The document structure is a tree; each node in this tree represents e.g. a chapter, section, paragraph, or Language, Subpart, Scene (for an MM document). If a node is expanded (i.e. the node is selected and its sons are displayed), the `expanded` optional Argument is updated with the id of the expanded node and the service is called again. The displayDocumentStructure service is called either by a OLP service or by itself. In every case, the whole document structure (including the metadata) will be provided as an XML-Stream, which will be transformed into an HTML-representation with respected to the expanded tree nodes.
The current session information (UserID, State, …) will be stored and accessed globally.

Example Request:
/OLP/UserInterface/1.0/displayDocumentStructure/d1?expanded=id1,id2?XMLStream=<?XML><tree>……

shows the structure of the document with the id d1; the nodes with the ids id1 and id2 are expanded.

generateRegistrationForm

```
Version: 1.0
Fixed Args: none
Optional Args: none
Return MIME type: text/html
```

Generates the registration form.

Example Request:
/OLP/UserInterface/1.0/generateRegistrationForm

submitRegistrationForm

```
Version: 1.0
Fixed Args: none
Optional Args: <List of POST-Input Fields>
```

```
      Return MIME type: text/html
```

This service will be called when the registration form is submitted.

Example Request:
/OLP/UserInterface/1.0/generateRegistrationForm/?field1=value1&field2=value2&……

displayGroupInfo

```
      Version: 1.0
      Fixed Args: groupname
      Optional Args: none
      Return MIME type: text/html
```

Displays information on a specific group. If the user is owner of the group, the group settings can be edited.

Example Request:
/OLP/UserInterface/1.0/displayGroupInfo/myGroup

displayBrowseResults

```
      Version: 1.0
      Fixed Args: none
      Keyword_post Args: XMLStream
      Return MIME type: text/html
```

In each Browsing Step, OLP calls the displayBrowseResults service and provides an XML-Stream, which will be parsed and transformed to HTML

Example Request:
/OLP/UserInterface/1.0/displayBrowseResults?XMLStream=<?XML><result>……

displaySearchResults

```
      Version: 1.0
      Fixed Args: none
      Keyword_post Args: XMLStream
      Return MIME type: text/html
```

OLP calls the displaySearchResults service and provides an XML-Stream, which will be parsed and transformed to HTML.

Example Request:
/OLP/UserInterface/1.0/displaySearchResults?XMLStream=<?XML><result>……

displayAnnotation

```
      Version: 1.0
      Fixed Args: none
      Keyword_post Args: XMLStream
      Return MIME type: text/html
```

A service of the annotation module calls the displayAnnotation service and provides an XML-Stream, which will be parsed and transformed to HTML

Example Request:
/OLP/UserInterface/1.0/displayAnnotation?XMLStream=<?XML><annotation>……

displayMetadata

```
      Version: 1.0
      Fixed Args: none
      Keyword_post Args: XMLStream
      Return MIME type: text/html
```

A service of OLP calls the displayMetadata service and provides an XML-Stream, which will be parsed and transformed to HTML

Example Request:
/OLP/UserInterface/1.0/displayMetadata?XMLStream=<?XML><metadata>……

displayMMDocument

```
Version: 1.1
Fixed Args: none
Optional Args: MMLink, Start, End
Return MIME type: depends on the document type
```

This service displays a multimedia document (by automatically invoking an appropriate plug-in). The start and end point of a time interval can be given (in the form *hhmmss*).

Example Request:
/OLP/UserInterface/1.0/displayMMDocument?MMLink=http://www.video.org?Start=000434?End=010934

requests the time interval 00:04:34 to 01:09:34 of the MM document to be displayed.

downloadMMDocument

```
Version: 1.0
Fixed Args: none
Optional Args: MMLink, Start, End
Return MIME type: depends on the document type
```

This service downloads a multimedia document (by automatically invoking a document-save-request). The start and end point of a time interval can be given (in the form *hhmmss*).

Example Request:
/OLP/UserInterface/1.0/downloadMMDocument?MMLink=http://www.video.org?Start=000434?End=010934

requests the time interval 00:04:34 to 01:09:34 of the MM document to be downloaded.

displayTextDocument

```
Version: 1.1
Fixed Args: none
Optional Args: DocLink
Return MIME type: depends on the document type (e.g. text/html,
text/pdf,…)
```

This service displays a text document (by automatically invoking an appropriate plug-in).

Example Request:
/OLP/UserInterface/1.0/displayTextDocument?MMLink=http://www.document.org

downloadTextDocument

```
Version: 1.1
Fixed Args: none
Optional Args: DocLink
Return MIME type: depends on the document type (e.g. text/html,
text/pdf,…)
```

This service downloads a text document (by automatically invoking a document-save-request).

Example Request:
/OLP/UserInterface/1.0/downloadTextDocument?MMLink=http://www.document.org

performThesaurusChange

```
Version: 1.0
Fixed Args: thesaurusname
Keyword Args:
Return MIME type: none
```

Performs all operations necessary to change the thesaurus. DisplayThesaurus is then invoked.

login

```
Version: 1.0
Fixed Args: none
Fixed_post Args: username, password
Optional Args: none
Return MIME type: text/html
```

After the login procedure, the system authenticates the user by calling this service. The current session information is updated.

Example Request:
/OLP/UserInterface/1.0/login

20. META SERVICE

The Meta Service manages the meta information that permits the distribution and replication of the services on multiple servers. It gathers information from all the servers, organises the information service space in a map, and disseminates this map to all other servers on demand.

The Meta Service thus organises the overall information service space into different regions. Each region contains complete information on a set of servers that provide the Scholnet services.
The replicated servers of a region are organised according to a priority value. At system start-up, the Meta Service administrator assigns this priority with respect to a logical optimised organisation of the servers. For each service request, servers will request information from the Meta Service in order to retrieve the URL of the servers that can satisfy the given request, selecting the server with the highest priority.

The Meta Service dynamically updates the priority values in order to maintain the best connectivity between servers. In other words, when a network or server failure occurs, rather than continuing to use the failing primary server that hosts a service before invoking the alternative one, the Meta Service assigns a temporary highest priority to another server. All other servers of the region will use the alternative server, immediately adapting the request routing. In this way, the best connectivity of the region is always maintained.

To implement this behaviour, each server sends a notification to the Meta Service when a server is not available. The Meta Service keeps track of the success or failure history of each server and using a simple adaptive algorithm manages the priority values. If a specific server repeatedly fails within a specified period and a secondary server exists that hosts the service, a low priority is assigned to the unreliable server and a higher value is assigned to the appropriate backup server. This change in the priority values remains for a fixed period, after which the original values are re-assigned if the server becomes stable again.

The distributed servers of the overall infrastructure are organised taking into account the publishing authorities served by each server. Each Repository and Library Management server specifies the publishing authority that they manage. The Meta Service administrator determines how the services around these authorities will be organised. He/she has to specify, for example, the primary Index server that indexes the documents published by an authority, plus a backup Index server for that authority.

The Meta Service also maintains service descriptions that make it possibleto provide the distributed or replicated services with a common configuration. The Index and Browse Services, for example, can manage generic bibliographic metadata but the format of this metadata must be the same on each server that hosts one of these services. For this reason, the Meta Service manages the Index/Browse configuration description, which contains the bibliographic metadata, the set of indexed fields and the result formats.

The administrators are those users authorised to use the Meta Service to manage the meta information. Administrators are currently defined at system set-up time.

Communication with the Meta Service takes place via the OpenDLib Protocol (OLP).

20.1 *STATE*

The Meta Service maintains a descriptive record for each of the digital library services. The format of these records is service type dependent. The Meta Service stores also descriptive records for each of the existing Publishing Authorities and Regions.
The digital library administrator fills some of the information stored in these records at set-up time. This information is briefly listed below organised by service type.

Publishing Authority

A naming authority is an entity that is authorised to create new handles. Naming authorities are hierarchically organised, with periods as the separators. For example CNR, CNR.IEI, CNR.IEI.MultimediaDepartment.
The list of authorities managed by a Meta Service is set up at time of configuration of the service, and can only be changed manually by the administrator of the service.

Collection

For each Collection server, the Meta Service administrator has to specify its address, and has to identify one of them as the master Collection Service. All other information, such as the version of the OLP protocol verbs supported by the specific Collection server, is gathered by the Meta Service directly querying the server.

Repository

For each Repository server, the Meta Service administrator only has to specify its address. All other information, such as the version of the OLP protocol verbs supported by the specific Repository server or the list of authorities hosted in it, is gathered by the Meta Service directly querying the server.

Multimedia Storage

For each Multimedia Storage server, the Meta Service administrator only has to specify its address. All other information, such as the version of the OLP protocol verbs supported by the specific Multimedia Storage server, is gathered by the Meta Service directly querying the server.

Library Management

For each Library Management server, the administrator of the Meta Service only has to specify its address. All other information, such as the version of the OLP protocol verbs supported by the specific Repository server or the list of authorities managed by it, is gathered by the Meta Service directly querying the server.

Index

For each Index server, the administrator has to specify the list of authorities indexed, and its address. He/she also has to specify the service configuration parameters, such as the bibliographic metadata that the service will manage, the set of fields to index, and the possible result formats. All other information, such as the version of the OLP protocol verbs supported by the specific Index server or the indexed language supported, is gathered by the Meta Service directly querying the server.

Query Mediator

For each Query Mediator server, the Meta Service administrator only has to specify its address. All other information, such as the version of the OLP protocol verbs supported by the specific Query Mediator server, is gathered by the Meta Service directly querying the server.

Browse

For each Browse server, the Meta Service administrator has to specify its address, and the configuration parameters, such as the bibliographic metadata that the service will manage, the set of browsable fields, and the possible results format. All other information, such as the version of the OLP protocol verbs supported by the specific Browse server, is gathered by the Meta Service directly querying the server.

Registry

The administrator of the Meta Service only has to specify the address of the Registry. All other information, such as the version of the OLP protocol verbs supported by the Registry server, is gathered by the Meta Service directly querying the server.

Personalized Dissemination

The Meta Service administrator only has to specify the address of the Personalized Dissemination Service. All other information is gathered by the Meta Service directly querying the server.

Annotation

For each Annotation server, the Meta Service administrator has to specify its address and the list of the authorities for which the server has to maintain the annotations. All other information is gathered by the Meta Service directly querying the server.

Multilingual Resources

The Meta Service administrator only has to specify the address of the Multilingual Resources Service. All other information is gathered by the Meta Service directly querying the server.

Regional Meta

The Meta Service administrator does not have to specify any information. Each server harvests the information pertaining to its region from the Meta Service.

Region

The Region concept is used to organise the overall information service space in an optimised map that automatically changes when the connectivity of a single server is compromised by network or server failure.
Here below, we show a map that describes a region.

Service	Server and Relation		
QMS			
	qm host="" port="" priority=""		
		indexer host="" port="" priority=""	
		indexer host="" port="" priority=""	
	qm host="" port="" priority=""		
		indexer host="" port="" priority=""	
			authority name="" priority=""
BS			
	browser host="" port="" priority=""		
	browser host="" port="" priority=""		
CS			
	collection host="" port="" priority=""		
	collection host="" port="" priority=""		

LMS		
	libmgt host="" port="" priority=""	
		authority name="" priority=""
		authority name="" priority=""
	libmgt host="" port="" priority=""	
		authority name="" priority=""
		authority name="" priority=""
RMSS		
	rms host="" port="" priority=""	
	rms host="" port="" priority=""	
RES	registry host="" port=""	
PIDS	pids name="" priority=""	
MTS	mts name="" priority=""	

Let us suppose, for example, that we have the authorities A1, A2, A3, A4, and the indexes I1, I2, I3. The administrator in a region may assign the following relationships:

Index	Authority	Priority
I1	A1	1
	A2	2
I2	A3	2
	A4	1
	A2	3
I3	A1	2
	A2	1
	A3	1
	A4	1

Using this information, the Query Mediator Service of the region knows that authority A1 is indexed, at a given time, by I1 with priority 1 and by I3 with priority 2. Therefore, the Query Mediator will first send a search request on authority A1 to index I1 and if it does not receive any result will then send the same request to index I3.
In general, services use the map to retrieve the primary address of the server to which a request is to be sent. A UI, for example, uses the map to retrieve the address of the QM to which a query is to be sent, the address of the Browse to which browse requests are to be sent, the address of the Registry where the user is to be authenticated, etc.

20.2 *META VERBS*

In this section, we present the set of the Meta verbs. This set is divided into three sub-sets: Service Information Verb, Architecture Information Verb, and Service Specific Verb as reported in the following table:

Service Information Verb	Architecture Information Verb	Service Specific Verb
DescribeVerb	ListAnnotations	RegionDescription
Identify	ListAuthorities	ServerBugReport
ListVerbs	ListBrowsers	ServiceConfigDescription
	ListCollections	

	ListIndices	
	ListLibMgts	
	ListMultimediaStorages	
	ListQueryMediators	
	ListRegionalMetaServers	
	ListRegions	
	ListRepositories	
	ListServices	
	MDFDetails	

Identify

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name and other information about the service.

Example Request:
/OLP/Meta/1.0/Identify

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<Identify version="1.0">
    <serviceName>Experimental OLP Meta Server</serviceName>
    <olp_base_url>http://labserv.iei.pi.cnr.it:8221</olp_base_url>
    <metaServiceURL>http://labserv.iei.pi.cnr.it:8221</metaServiceURL>
    <textualDescription/>
    <protocolVersion>1.0</protocolVersion>
    <adminEmail>pagano@iei.pi.cnr.it</adminEmail>
    <verbsInfo>
        <verbsSupported>
            <description/>
            <olp_base_url>http://labserv.iei.pi.cnr.it:8221/OLP/Meta/1.0/ListVerbs</olp_base_url>
        </verbsSupported>
        <verbsDescription>
            <description/>
            <olp_base_url>http://…:8221/OLP/Meta/1.0/DescribeVerb/DescribeVerb</olp_base_url>
        </verbsDescription>
    </verbsInfo>
    <submissionProcedure>
        <description>Submission procedure is not supported by this service.
        </description>
        <olp_base_url/>
    </submissionProcedure>
    <harvestInformation>
        <description>
                No information is available
            </description>
        <olp_base_url/>
    </harvestInformation>
    <useRestrictions>Terms and conditions are …...</useRestrictions>
    <contentInfo>
        <publishingAuthority>
            <description>
              A naming authority is an entity that is authorised to create new handles. Naming authorities are
              hierarchically organised, with periods as the separators. For example CNR, CNR.IEI,
              CNR.IEI.MultimediaDepartment.
        The list of authorities managed by a Meta Service is set up at time of configuration of the
        service, and can only be changed manually by the administrator of the service.
    </description>
    <olp_base_url>http://labserv.iei.pi.cnr.it:8221/OLP/Meta/1.0/ListAuthorities</olp_base_url>
</publishingAuthority>
<collection>
    <description>
        Collection Service provides a virtual organisation of the documents stored in the repositories. It
        supplies the information necessary to manage these virtual document aggregations. This
        information is used by the other services  to handle the collection objects enabling, for example,
        the Query Mediator to perform a query on a specified collection, or the Browse to perform a
        browse on a collection, and so on. It can be replicated on multiple servers.
        </description>
    <olp_base_url>http://labserv.iei.pi.cnr.it:8221/OLP/Meta/1.0/ListCollections</olp_base_url>
</collection>
<repository>
    <description>
        Repository stores documents that conform to the DoMDL document model. It can be distributed
        on multiple servers because a Repository server can store documents published by different
        authorities and different authorities can be hosted by different Repository servers.
    </description>
    <olp_base_url>http://labserv.iei.pi.cnr.it:8221/OLP/Meta/1.0/ListRepositories</olp_base_url>
</repository>
<multimediaStorage>
    <description>
        Multimedia Storage stores video documents (according to the DoMDL document model), and
        supports their dissemination either as whole documents or as aggregations of scenes, shots
        and frames. It can be distributed on multiple servers because a Multimedia Storage server can
        store documents published by different authorities and different authorities can be hosted by
        different Multimedia Storage servers.
    </description>
    <olp_base_url>http://..:8221/OLP/Meta/1.0/ListMultimediaStorages</olp_base_url>
</multimediaStorage>
<libraryManagement>
    <description>
        Library Management supports the submission, withdrawal, and replacement of documents. It
        can be distributed on multiple servers because a Library Management server can manage
        documents of different authorities published in different Repositories and different authorities
        can be managed by different Library Management servers.
    </description>
    <olp_base_url>http://labserv.iei.pi.cnr.it:8221/OLP/Meta/1.0/ListLibMgts</olp_base_url>
</libraryManagement>
<index>
    <description>
        Index accepts queries and returns lists of document identifiers matching those queries. It can be
        distributed and replicated on multiple servers. It is distributed because an Index server can
        index documents published by different authorities that are stored in different Repositories. It is
        replicated because the document published by different authorities can be indexed by different
        Index servers.
    </description>
    <olp_base_url>http://labserv.iei.pi.cnr.it:8221/OLP/Meta/1.0/ListIndices</olp_base_url>
</index>
<queryMediator>
    <description>
        Query Mediator dispatches queries to appropriate index servers. It can be replicated on multiple
        servers.
    </description>
    <olp_base_url>http://labserv.iei.pi.cnr.it:8221/OLP/Meta/1.0/ListQueryMediators</olp_base_url>
</queryMediator>
<browse>
    <description>
```

```
            Browse supports the construction of browsing indexes and the browsing of those indexes on
            library contents. It can be replicated on multiple servers.
        </description>
        <olp_base_url>http://labserv.iei.pi.cnr.it:8221/OLP/Meta/1.0/ListBrowsers</olp_base_url>
    </browse>
    <registry>
        <description>
            Registry supports the storage and access of information about authors, individual users, and
            user groups. Hosted by one server.
        </description>
        <olp_base_url/>
    </registry>
    <personalisedDissemination>
        <description>
            Personalised Dissemination  supports the storage and execution of persistent queries, on a
            collection. Hosted by one server.
        </description>
        <olp_base_url/>
    </personalisedDissemination>
    <annotation>
        <description>
            Annotation stores annotations on documents and makes them available to authorised users. It
            can be distributed on multiple servers because an Annotation server can store annotations on
            documents published by different authorities that are stored in different repositories.
        </description>
        <olp_base_url>http://labserv.iei.pi.cnr.it:8221/OLP/Meta/1.0/ListAnnotations</olp_base_url>
    </annotation>
    <multilingualThesaurus>
        <description>
            Multilingual Thesaurus maintains language resources, such as multilingual thesauri, which are
            required to support cross-language functionality. Hosted by one server.
        </description>
        <olp_base_url/>
    </multilingualThesaurus>
    <regionalMeta>
        <description>
            Regional Meta Service maintains the information pertaining to the regions (see below). It is
            replicated on multiple servers, one for each region.
        </description>
        <olp_base_url>http://labserv.iei.pi.cnr.it:8221/OLP/Meta/1.0/ListRegions</olp_base_url>
    </regionalMeta>
    <region>
        <description>
            The Region concept is used to organise the overall information service space in an optimised
            map that automatically changes when the connectivity of a single server is compromised by
            network or server failure.
        </description>
        <olp_base_url>http://labserv.iei.pi.cnr.it:8221/OLP/Meta/1.0/Regiondescription</olp_base_url>
    </region>
  </contentInfo>
</Identify>
```

ListVerbs

```
    Version: 1.0
    Fixed args: none
    Optional Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400, 501
```

Returns a structured response containing the name of the verbs defined by this service.

Example Request:
/OLP/Meta/1.0/ListVerbs

DescribeVerb

```
Version: 1.0
Fixed args: verb
Optional Args: version
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response that contains a list in which each element of the list provides information on a version of the verb that is supported by this service. The following information may be provided at the verb or version level.
description, description of the verb or a specific version
note, information pertaining to the verb or a specific version
Each element of the list contains the following information:
version number of the verb.
arguments, a list of the names of the fixed and optional arguments, if any, accepted by the verb in that version.
example template of request to this LibMgt, with fixed arguments indicated in brackets
returns, optional, contains information about the response format.
Note that a service may implement more than one version of a verb.

Example Request:
/OLP/Meta/1.0/DescribeVerb/ListServices

ListServices

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the list of all available services. Each element of the list reports the name of the services, a brief description, and a short name used as key in the RegionDescription verb.

Example Request:
/OLP/Meta/1.0/ListServices

Example Response
```
<?xml version="1.0" encoding="UTF-8" ?>
<ListServices version="1.0">
   <service name="Repository" key="repository" description="Repository Service"/>
   <service name="LibMgt" key="libmgt" description="Library Management Service"/>
   <service name="MDS" key="mds" description="Multimedia Document Storage and Delivery Service"/>
   <service name="Collection" key="collection" description="Collection Service"/>
   <service name="Index" key="indexer" description="Index Service"/>
   <service name="QM" key="qm" description="Query Mediator Service"/>
   <service name="Browse" key="browser" description="Browse Service"/>
   <service name="Registry" key="registry" description="Registry Service"/>
   <service name="PIDS" key="pids" description="Personalised Dissemination Service"/>
   <service name="HAS" key="annotation" description="Annotation Service"/>
   <service name="MTS" key="mts" description="Multilingual Thesaurus Service"/>
   <service name="Meta" key="rms" description="Regional Meta"/>
</ListServices>
```

ListRepositories

```
Version: 1.0
Fixed args: none
Optional Args: verbs, auths
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the addresses of the servers hosting the Repository Service.
The meaning of the optional arguments is as follows:
verbs: specifies the verbose mode of the response. For each server, the complete list of verbs supported is returned. It assumes a Boolean value.
auths: the list of authorities hosted by each server is requested. It assumes a Boolean value.

Example Request:
/OLP/Meta/1.0/ListRepositories?auths=true&verbs=ture

Example Response:
```
<?xml version="1.0" encoding="UTF-8" ?>
<ListRepositories version="1.0">
    <repository port="8230" host="labserv.iei.pi.cnr.it">
        <verbs>
            <verb>BuildFormats</verb>
            <verb>DescribeVerb</verb>
            <verb>Disseminate</verb>
            <verb>Identify</verb>
            <verb>ListAuthorities</verb>
            <verb>ListBinders</verb>
            ……………
        </verbs>
        <authorities>
            <authority name="ercim.cnr.imc" display="CNR-IMC">
                <allowed-sets>
                    <set>csp</set>
                    <set>csp;trs</set>
                </allowed-sets>
            </authority>
            <authority name="ercim.cnr.iei" display="CNR-IEI">
                <allowed-sets>
                    <set>csp</set>
                    <set>csp;trs</set>
                    <set>test</set>
                </allowed-sets>
            </authority>
        </authorities>
    </repository>
</ListRepositories>
```

ListMultimediaStorages

```
Version: 1.0
Fixed args: none
Optional Args: verbs, auths
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the addresses of the servers hosting the Multimedia Storage Service.
The meaning of the optional arguments is as follows:

verbs: specifies the verbose mode of the response. For each server, the complete list of verbs supported is returned. It assumes a Boolean value.
auths: the list of authorities hosted by each server is requested. It assumes a Boolean value.

Example Request:
/OLP/Meta/1.0/ListMultimediaStorages?auths=true

Example Response:
```
<?xml version="1.0" encoding="UTF-8" ?>
<ListMultimediaStorages version="1.0">
    <mds port="8230" host="labserv.iei.pi.cnr.it">
        <authorities>
            <authority name="ercim.cnr.imc" display="CNR-IMC"/>
            <authority name="ercim.cnr.iei" display="CNR-IEI"/>
        </authorities>
    </mds>
</ListMultimediaStorages>
```

ListLibMgts

```
Version: 1.0
Fixed args: none
Optional Args: region, verbs, auths
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the addresses of the servers hosting the LibMgt Service.
The meaning of the optional arguments is as follows:
region: specifies a region name. Only the list of the servers of the region is returned. If omitted, it provides information about the overall set of LibMgt servers. If specified, the *verbs* and *auths* parameters are ignored.
verbs: specifies the verbose mode of the response. For each server, the complete list of verbs supported is returned. It assumes a Boolean value.
auths: the list of authorities hosted by each server is requested. It assumes a Boolean value.

Example Request:
/OLP/Meta/1.0/ListLibMgts?auths=true

Example Response:
```
<?xml version="1.0" encoding="UTF-8" ?>
<ListLibMgts version="1.0">
    <libmgt port="8230" host="labserv.iei.pi.cnr.it">
        <authorities>
            <authority display="cornell university, cs two-bad" name="ncstrl.cornell.two.bad" />
            <authority display="cornell university, cs one" name="ncstrl.cornell.one" />
            <authority display="cornell university, cs one-trs" name="ncstrl.cornell.one.trs" />
            <authority display="cornell university, cs two" name="ncstrl.cornell.two" />
            <authority display="cornell university, cs one-trs-good" name="ncstrl.cornell.one.trs.good" />
        </authorities>
    </libmgt>
</ListLibMgts>
```

ListIndices

```
Version: 1.0
Fixed args: none
Optional Args: region, verbs, auths
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the addresses of the servers hosting the Index Service.
The meaning of the optional arguments is as follows:
region: specifies a region name. Only the list of the servers of the region is returned. If omitted, it provides information about the overall set of Index servers. If specified, the *verbs* and *auths* parameters are ignored.
verbs: specifies the verbose mode of the response. For each server, the complete list of verbs supported is returned. It assumes a Boolean value.
auths: requests the list of authorities hosted by each server. It assumes a Boolean value.

Example Request:
/OLP/Meta/1.0/ListIndices?auths=true

Example Response:
```
<?xml version="1.0" encoding="UTF-8" ?>
<ListIndices version="1.0">
   <indexer port="8231" host="labserv.iei.pi.cnr.it" filtering_attribute=”dc:language” filtering_value=”en”
            meta-format=”olms”>
      <authorities>
         <authority display="Cornell University, CS two-bad" name="ncstrl.cornell.two.bad" />
         <authority display="Cornell University, CS one" name="ncstrl.cornell.one" />
      </authorities>
   </indexer>
   <indexer port="8230" host="labserv.iei.pi.cnr.it" filtering_attribute=”dc:language” filtering_value=”it” meta-
            format=”dc”>
      <authorities>
         <authority display="Cornell University, CS two-bad" name="ncstrl.cornell.two.bad" />
         <authority display="Cornell University, CS one" name="ncstrl.cornell.one" />
      </authorities>
   </indexer>
</ListIndices>
```

ListQueryMediators

```
Version: 1.0
Fixed args: none
Optional Args: region, verbs
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the addresses of the servers hosting the Query Mediator Service.
The meaning of the optional arguments is as follows:
region: specifies a region name. Only the list of the Query Mediator servers of the region is returned. If omitted, it provides information about the overall set of Query Mediator servers. If specified, the *verbs* parameter is ignored.
verbs: specifies the verbose mode of the response. For each server, the complete list of verbs supported by the each server is returned. It assumes a Boolean value.

Example Request:
/OLP/Meta/1.0/ListQueryMediators?region=TEST

Example Response:
```
<?xml version="1.0" encoding="UTF-8" ?>
<ListQueryMediators version="1.0">
   <qm host="labserv.iei.pi.cnr.it" port="8119"/>
   <qm host="labserv.iei.pi.cnr.it" port="8129"/>
</ListQueryMediators>
```

ListBrowsers

```
Version: 1.0
Fixed args: none
Optional Args: region, verbs
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the addresses of the servers hosting the Browser Service.
The meaning of the optional arguments is as follows:
region: specifies a region name. Only the list of the Browser servers of the region is returned. If omitted, it provides information about the overall set of Browser servers. If specified, the *verbs* parameter is ignored.
verbs: specifies the verbose mode of the response. For each server, the complete list of verbs supported is returned. It assumes a Boolean value.

Example Request:
/OLP/Meta/1.0/ListBrowsers?region=TEST

Example Response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ListBrowsers version="1.0">
   <browser host="labserv.iei.pi.cnr.it" port="8111"/>
   <browser host="labserv.iei.pi.cnr.it" port="8112"/>
</ListBrowsers>
```

ListAnnotations

```
Version: 1.0
Fixed args: none
Optional Args: verbs, auths
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the addresses of the servers hosting the Annotation Service.
The meaning of the optional arguments is as follows:
verbs: specifies the verbose mode of the response. For each server, the complete list of verbs supported is returned. It assumes a Boolean value.
auths: requests the list of authorities hosted by each server. It assumes a Boolean value.

Example Request:
/OLP/Meta/1.0/ListAnnotations?auths=1

Example Response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ListAnnotations version="1.0">
   <annotation host="labserv.iei.pi.cnr.it" port="8111">
      <authorities>
         <authority display="cornell university, cs two-bad" name="ncstrl.cornell.two.bad" />
         <authority display="cornell university, cs one" name="ncstrl.cornell.one" />
         <authority display="cornell university, cs one-trs" name="ncstrl.cornell.one.trs" />
      </authorities>
   </annotation>
   <annotation host="labserv.iei.pi.cnr.it" port="8112">
      <authorities>
         <authority display="cornell university, cs two" name="ncstrl.cornell.two" />
            <authority display="cornell university, cs one-trs-good" name="ncstrl.cornell.one.trs.good" />
```

```
        </authorities>
    </annotation>
</ListAnnotations>
```

ListCollections

```
Version: 1.0
Fixed args: none
Optional Args: region, verbs
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the addresses of the servers hosting the Collection Service.
The meaning of the optional arguments is as follows:
region: specifies a region name. Only the list of the Collection servers of the region is returned. If omitted, it provides information about the overall set of Collection servers. If specified, the *verbs* parameter is ignored.
verbs: specifies the verbose mode of the response. For each server, the complete list of verbs supported is returned. It assumes a Boolean value.

Example Request:
/OLP/Meta/1.0/ListCollections?region=TEST

Example Response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ListCollections version="1.0">
    <collection host="labserv.iei.pi.cnr.it" port="8111" master="yes"/>
    <collection host="labserv.iei.pi.cnr.it" port="8112" master="no"/>
</ListCollections>
```

ListRegionalMetaServers

```
Version: 1.0
Fixed args: none
Optional Args: region
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the addresses of the servers hosting the Regional Meta Service.
The meaning of the optional arguments is as follows:
region: specifies a region name. Only the list of the Browser servers of the region is returned. If omitted, it provides information about the overall set of RMS servers.

Example Request:
/OLP/Meta/1.0/ ListRegionalMetaServers?region=TEST

Example Response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ListRegionalMetaServers version="1.0">
    <rms host="labserv.iei.pi.cnr.it" port="8111" priority="1"/>
    <rms host="labserv.iei.pi.cnr.it" port="8112" priority="2"/>
</ListRegionalMetaServers>
```

ListRegions

```
Version: 1.0
Fixed args: none
Optional Args: region
```

```
     Return MIME type: text/xml
     Return Status Codes: 200, 400, 501
```

Returns a structured response containing the addresses of the servers hosting the Regional Meta Service.
The meaning of the optional arguments is as follows:
region: specifies a region name. Only the list of the Regional Meta server of the region is returned. If omitted, it provides information about the overall set of Regional Meta servers.

Example Request:
/OLP/Meta/1.0/ListRegions?region=TEST

Example Response:
```
<?xml version="1.0" encoding="UTF-8" ?>
<ListRegions version="1.0">
   <region host="labserv.iei.pi.cnr.it" port="8000" name="TEST"/>
</ListRegions>
```

ListAuthorities

```
     Version: 1.0
     Fixed args: none
     Optional Args: none
     Return MIME type: text/xml
     Return Status Codes: 200, 400, 501
```

Returns a structured response containing the overall set of authorities that publish documents in the environment.
For each element of the list the following information is returned:
name, an identifier used in any service request that includes the authority argument
display, a short description that can be displayed to the user

Example Request:
/OLP/Meta/1.0/ListAuthorities

Example Response:
```
<?xml version="1.0" encoding="UTF-8" ?>
<ListAuthorities version="1.0">
   <publisher display="CNR - Istituto CNUCE (Pisa)" authority="ercim.cnr.cnuce" />
   <publisher display="CNR - Istituto per le Applicazioni Telematiche (Pisa)" authority="ercim.cnr.iat" />
   <publisher display="CNR - Istituto di Elaborazione della Informazione (Pisa)" authority="ercim.cnr.iei" />
</ListAuthorities>
```

-- --

```
     Version: 1.1
     Fixed args: none
     Optional Args: authorities
     Optional_Post Args: authorities
     Return MIME type: text/xml
     Return Status Codes: 200, 400, 501
```

Returns a quite complex structured response containing a set of authorities that publish documents in the environment. The set of authorities reported in the result take into account the set of authorities specified as “authorities” argument. If the argument is not speicifed the overall set of authorities is assument by default. If a list is apecified the “%20” separator has to be used, as reported below:
authorities=scholnet.test%20ercim.cnr.iei%20ercim.cnr.gmd
For each authority of the list the following information is returned:
1) the metadata formats available for the documents published by the specified authority;

2) the repository server that stores the documents;
3) the libmgt server/s that can manage the submission/withdrawal phase;
4) the indexer server/s that index the documents;
5) the annotation server/s that maintain the annotation related to the documents published by the specified authority.
When a registered user wants, for example, to submit a document, he/she selects the authority that has to publish the document for him. The UI, therefore, has to select the LibMgt server to send the request. Using the new response format of the ListAuthorities verb this selection is very simple.

Example Request:
/OLP/Meta/1.1/ListAuthorities

Example Response:

```
<?xml version="1.0" encoding="UTF-8" ?>
<ListAuthorities authorities_without_any_indexer="0" authorities_without_any_libmgt="0"
  authorities_without_any_repository="0" version="1.1" authorities_processed="124">
  <publisher pretty="test authority" publisher="scholnet.test" authority="scholnet.test">
    <metadataformats>
      <metadataformat value="DC" source_format="no" />
      <metadataformat value="olms" source_format="yes" />
    </metadataformats>
    <services>
      <class value="repository">
        <repository port="8230" host="project.iei.pi.cnr.it" />
      </class>
      <class value="libmgt">
        <libmgt port="8229" host="project.iei.pi.cnr.it" />
      </class>
      <class value="indexer">
        <indexer filtering_attribute="dc:language" meta-format="olms" port="8228"
            filtering_value="en" host="project.iei.pi.cnr.it" />
        <indexer filtering_attribute="dc:language" meta-format="olms" port="8229"
            filtering_value="it" host="project.iei.pi.cnr.it" />
        <indexer filtering_attribute="dc:language" meta-format="olms" port="8230"
            filtering_value="en" host="project.iei.pi.cnr.it" />
      </class>
      <class value="annotation">
        <annotation host="….." port="…." />
      </class>
    </services>
  </publisher>
  <publisher ……
  …….
</ListAuthorities>
```

MDFDetails

```
Version: 1.0
Fixed args: none
Optional Args: none
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the overall set of metadata format specifications that are used in the environment.
For each element of the list the following information is returned:
value, an identifier used in any service request that includes the metadata argument
dtd, the reference to the DTD of the metadata

contenttype, the specification of the MIME type of the metadata
displayname, a short description that can be displayed to the user
pattern, the pattern that is used to store metadata manifestation
variable, is a list of names of variables to be used as input to the pattern. Valid variables are: publisher, number, page, name plus any others defined. This information is used by the Repository Service.
internal, flag used by the Repository Service
namespace, a list of the namespaces used by the metadata formats
doc, a long description that can be displayed to the user
starttag, the first tag of a metadata manifestation. Used by the Index Service.
memofield, a list of fields that may contain multiple lines text or blob data. Used by the UI.

Example Request:
/OLP/Meta/1.1/MDFDetails

Example Response:
```
<?xml version="1.0" encoding="UTF-8" ?>
<metadataformats>
  <metadataformat value="bib" contenttype="text/plain" displayname="cataloging information"
internal="1" pattern="%s.bib" variable="name">
    <namespace name="bibns" uri="ftp://nic.merit.edu/document/rfc/rfc1807.txt" />
    <doc />
  </metadataformat>
  <metadataformat value="dc" dtd="http://project.iei.pi.cnr.it:8221/OLP/htdocs/DTD/dc.dtd"
contenttype="text/xml" displayname="DC XML encoded cataloguing information"
pattern="%s_dc.xml" variable="name" internal="1">
    <namespace name="dc" uri="http://purl.org/dc/elements/1.1/" />
    <doc />
    <memofields>
      <memofield value="dc:description" />
    </memofields>
  </metadataformat>
  <metadataformat value="dcq" dtd="http://project.iei.pi.cnr.it:8221/OLP/htdocs/DTD/dcq.dtd"
contenttype="text/xml" displayname="DC qualified XML encoded cataloguing information"
pattern="%s_dcq.xml" variable="name" internal="1">
    <namespace name="dcq" uri="http://purl.org/dc/terms/" />
    <namespace name="dc" uri="http://purl.org/dc/elements/1.1/" />
    <doc />
    <memofields>
      <memofield value="dcq:description.abstract" />
      <memofield value="dc:description" />
      <memofield value="dcq:description.tableofcontents" />
    </memofields>
  </metadataformat>
<metadataformat value="rfc1807" dtd="http://proj.iei.pi.cnr.it:8221/OLP/htdocs/DTD/rfc1807.dtd"
contenttype="text/xml" displayname="RFC-1807 XML encoded cataloguing information"
pattern="%s_rfc1807.xml" variable="name" internal="1">
  <namespace name="rfc1807" uri="ftp://nic.merit.edu/document/rfc/rfc1807.txt" />
  <doc />
  <memofields>
    <memofield value="rfc1807:description" />
  </memofields>
</metadataformat>
<metadataformat value="olms" dtd="http://project.iei.pi.cnr.it:8221/OLP/htdocs/DTD/olap.dtd"
contenttype="text/xml" internal="1" displayname="Open Library XML encoded cataloguing
information" pattern="%s_olms.xml" variable="name">
  <namespace name="dcq" uri="http://purl.org/dc/terms/" />
  <namespace name="dc" uri="http://purl.org/dc/elements/1.1/" />
  <namespace name="olms" uri="http://p.iei.pi.cnr.it:8221/OLP/htdocs/NameSpace/olms.xml" />
  <doc />
  <memofields>
    <memofield value="dc:description" />
    <memofield value="dcq:description.abstract" />
    <memofield value="olms:description.abstract.alternative" />
    <memofield value="dcq:description.toc" />
    <memofield value="olms:description.toc.alternative" />
    <memofield value="olms:note" />
    <memofield value="olms:note.retireval" />
    <memofield value="olms:note.withdraw" />
    <memofield value="olms:note.contact" />
    <memofield value="olms:note.collection" />
    <memofield value="olms:note.series" />
    <memofield value="olms:note.revision" />
  </memofields>
 </metadataformat>
 </metadataformats>
```

RegionDescription

```
Version: 1.0
Fixed args: none
Optional Args: region
Return MIME type: text/xml
Return Status Codes: 200, 400, 501
```

Returns a structured response containing the addresses of all servers grouped by region. The only server that is not represented in this response is the Repository.
The meaning of the optional arguments is as follows:
region: specifies a region name. Only the list of the servers of the region is returned. If omitted, it provides information about the overall set of servers.

Example Request:
/OLP/Meta/1.0/RegionDescription?region=TEST

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<RegionDescription version="1.0">
    <Regions>
    <region name="" symbol="">
        <qms>
            <qm host="" port="" priority="">
                <indexer host="" port="" priority=""/>
                <indexer host="" port="" priority="" />
            </qm>
            <qm host="" port="" priority="">
                <indexer host="" port="" priority="" >
                    <authority name="" priority=""/>
                </indexer>
            </qm>
        </qms>
        <browsers>
            <browser host="" port="" priority=""/>
            <browser host="" port="" priority=""/>
        </browsers>
        <libmgts>
            <libmgt host="" port="" priority=""/>
            <libmgt host="" port="" priority=""/>
        </libmgts>
        <collections>
            <collection host="" port="" priority="" />
            <collection host="" port="" priority="" />
        </collections>
        <rmss>
            <rms host="" port="" priority=""/>
            <rms host="" port="" priority=""/>
        </rmss>
        <registry host="" port=""/>
        <mts host="" port=""/>
        <pids host="" port=""/>
    </region>
    </Regions>
</RegionDescription>
```

ServerBugReport

```
    Version: 1.0
    Fixed args: service, host, port
    Optional Args: request, error_message
    Return MIME type: text/plain
    Return Status Codes: 200, 400, 501
```

Signals that an error occurred when sending a request to a server.
The mandatory arguments specify that the server hosting a given service is unreliable.
The meaning of the optional arguments is as follows:
request: specifies the request sent to the server .
error_*message*: specifies the error message returned by the server. If no value is specified the default "time out" is assumed.
Returns 200 if the request is accepted, the appropriate error code otherwise.

Example Request:
/OLP/Meta/1.0/ServerBugReport/Repository/labserv.iei.pi.cnr.it/80?request="OLP/Repository/1.0/ListVerbs"

ServiceConfigDescription

```
    Version: 1.0
    Fixed args: service
    Optional Args: none
    Return MIME type: text/xml
    Return Status Codes: 200, 400, 501
```

Returns a structured response containing the configuration parameters of the service specified.
The accepted value for the service argument is listed by the ListServices as key value of each service.

Example Request:
/OLP/Meta/1.0/ServiceConfigDescription/browser

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ServiceConfigDescription version="1.0">
    <service name="browser">
        <meta-format name="olms" uri="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/">
            <browsable-fields>
                <field name="dc:creator">
                    <type>string</type>
                </field>
                <field name="dc:date">
                <type>%4d/%2d/%d(%d)</type>
                </field>
```

```
                        ………
            </browsable-fields>.
            <result-formats>
                <format name="short" default="yes">
                <field>dc:title</field>
                <field>dc:date</field>
                </format>
                <format name="long">
                    <field>dc:creator</field>
                <field>dc:title</field>
                <field>dc:date</field>
                <field>dc:subject</field>
                </format>
            </result-formats>
        </meta-format>
        <meta-format name="" uri="">
            …..
        </meta-format>
    </service>
</ServiceConfigDescription
```

Example Request:
/OLP/Meta/1.0/ServiceConfigDescription/indexer

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<ServiceConfigDescription version="1.0">
    <service name="indexer">
        <meta-format name="olms" uri="http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/">
        <filtering value="en" name="dc:language" default="true">
            <indexed-fields>
                    <field name="dc:creator">
                        <type>string</type>
                    </field>
                    <field name="dc:date">
                            <type>%4d/%2d/%d(%d)</type>
                    </field>
                    <field name="dc:description" support_feedback="yes">
                            <type>%4d/%2d/%d(%d)</type>
                    </field>
                    <field name="dcq:description.abstract" support_feedback="yes">
                            <type>%4d/%2d/%d(%d)</type>
                    </field>
                        ………
            </indexed-fields>
                <result-formats>
                    <format name="short" default='yes'>
                            <field>dc:title</field>
                            <field>dc:date</field>
                    </format>
                    <format name="long">
                        <field>dc:creator</field>
                            <field>dc:title</field>
                            <field>dc:date</field>
                            <field>dc:subject</field>
                    </format>
                </result-formats>
        </filtering>
        <filtering value="%lang" name="dc:language">
            <indexed-fields>
                    <field name="dcq:title.alternative">
```

```
                        <type>string</type>
                    </field>
                    <field name=”dcq:abstract.alternative”>
                            <type>string</type>
                </field>
                        ………
            </indexed-fields>
                <result-formats>
                    <format name=”short” default=’yes’>
                            <field>dc:title</field>
                            <field>dc:date</field>
                    </format>
                    <format name=”long”>
                        <field>dc:creator</field>
                            <field>dc:title.alternative</field>
                            <field>dc:date</field>
                            <field>dc:subject.alternative</field>
                </result-formats>
        </filtering>
        </meta-format>
        <meta-format name=”” uri=””>
            …..
        </meta-format>
    </service>
</ServiceConfigDescription
```

Example Request:
/OLP/Meta/1.0/ServiceConfigDescription/qm

Example Response:

```
<?xml version="1.0" encoding="UTF-8"?>
<service name="qm">
    <verb name="SearchAcross">
    <meta-format name="olms" uri=" http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/NameSpace/">
        <filtering value="en" name="dc:language" default="true">
            <indexed-fields>
                <field name="olms:subject.ccs" type="direct" ord="1">
                    <type>string</type>
                </field>
                <field name="dcq:description.abstract" type="calculated" ord="2">
                    <type>string</type>
                </field>
            </indexed-fields>
        </filtering>
        <filtering value="%lang" name="dc:language">
            <indexed-fields>
                <field name="olms:subject.ccs.alternative" type="direct" ord="1">
                    <type>string</type>
                </field>
                <field name="dcq:description.abstract.alternative" type="calculated" ord="2">
                    <type>string</type>
                </field>
            </indexed-fields>
        </filtering>
    </meta-format>
   </verb>
</service>
```

21. A METADATA EDITOR FOR MULTIMEDIA DOCUMENTS

This section introduces a new tool that will be implemented in the Scholnet project to support the user in the production of the metadata descriptions of multimedia documents.

21.1 *REQUIREMENTS*

Editing metadata for multimedia structured documents such as, for example, videos of lectures, tutorials, etc., is not always a straightforward task. According the DoMDL model there are different levels of description: versions views and manifestations. Each of these has its own specific metadata descriptive fields. Moreover, views may be composed of several parts, and, again, each of these parts may have its own description. Unlike Scholnet textual documents, the parts of a Scholnet audio/video will rarely be derivable automatically. Although there are tools for automatic scene detection of video material, these are not applicable for most of the Scholnet documents. A change of scene is seldom an indication of a meaningful breakdown in the video of a lecture. For this type of document the structuring must mainly be performed by a human that applies some criteria based on the content and the semantics of the lecture itself.
In some cases, for example, when the documents are in MPEG7 or SMILE formats, some of the descriptive metadata are stored with the content and can be extracted automatically. This metadata must be profitably exploited to reduce the effort of the cataloguer.

A semi-automatic metadata editor to support the partitioning of video documents and the creation and editing of the metadata descriptions associated with them will be implemented as part of the Scholnet project to meet the above specific requirements. The next sections specifies this tool.

21.2 *METADATA EDITOR USE CASE*

1. **Preconditions**:

The typical cataloguer workflow scenario is the following:

1. A new audiovisual document is digitazed and/or transformed from one digital format into another.
2. The document is processed for automatic metadata extraction**.**
3. When the metadata extraction has been completed, the manual editing of the metadata guided by the editor can start.

Main flow

The user typically edits the textual description for typos or factual content, reviews or sets values for the metadata fields, selects and adjusts the bounds of the document parts, removes unwanted segments and merges multiple documents. This phase is usually performed starting from the top level of the model (the DoMDL document), and is continued by modifying/editing the lower-level objects connected to the document (i.e., Version, View and Manifestation).

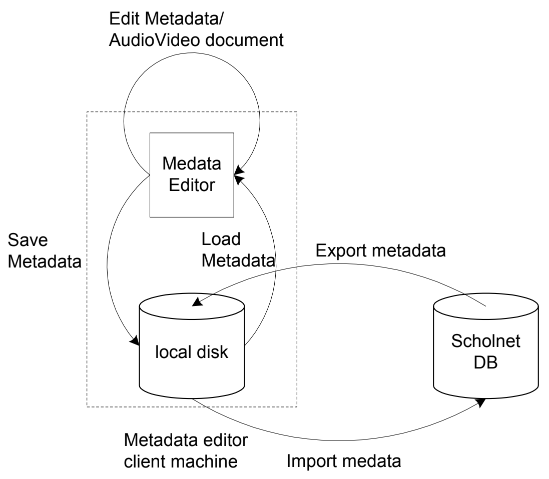


Figure 1: The workflow of the main metadata editor functionalities.

The following functions will be provided in order to support this behaviour (see Figure 1):

Load metadata description
Allows the user to load metadata descriptions that have been previously exported from the Scholnet database and locally stored in the XML format (see Figure 2).

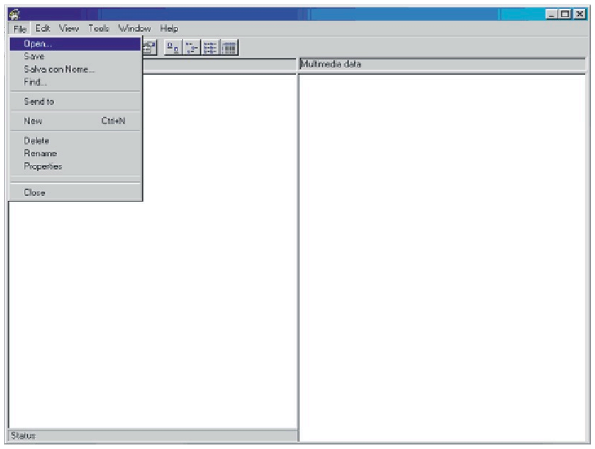


Figure 2: Loading the metadata description

Display metadata description
Used to browse the model (represented by a tree structure) and to display all the metadata descriptions associated with the objects of the model (see Figure 3).

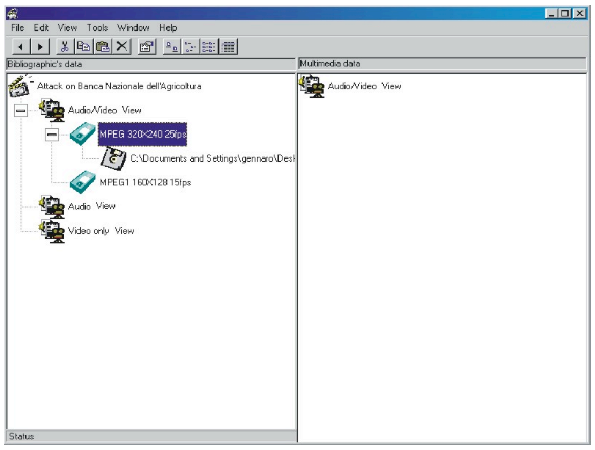


Figure 3: The structure of the loaded document

Modify metadata field values

By clicking on the icon representing an entity of the loaded document a popup menu can be accessed (see Figure 4) that allows the user to open a window (see Figure 5) in order to modify the metadata descriptions.

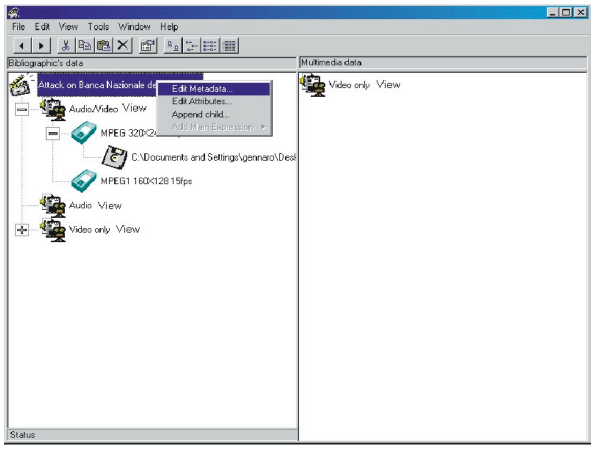
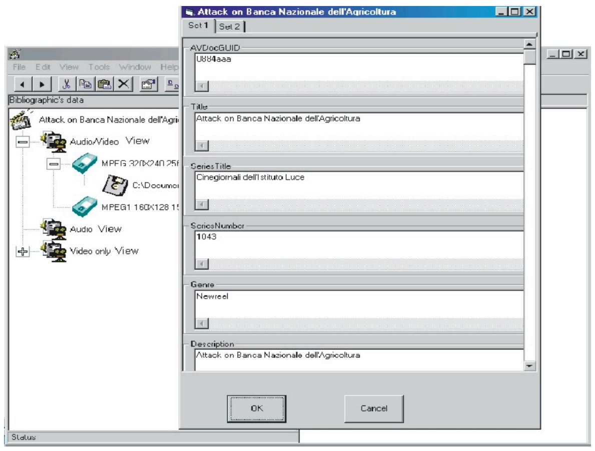


Figure 4: Modifying the Metadata descriptions



AVDocGUID: U884aaa
Title: Attack on Banca Nazionale dell'Agricoltura
Series Title: Cinegiornali dell'Istituto Luce
SeriesNumber: 1043
Genre: Newreel
Description: Attack on Banca Nazionale dell'Agricoltura

Figure 5: Editing the metadata descriptions

Navigate through the audio/video document
The audio/video documents are stored as View of the DoMDL Document. The View can represent the hierarchy of the video segmentation (For instance, a Video can be divided in scenes, shots, etc). Through the metadata editor it is possible to visualize this segmentation (see Figure 6).

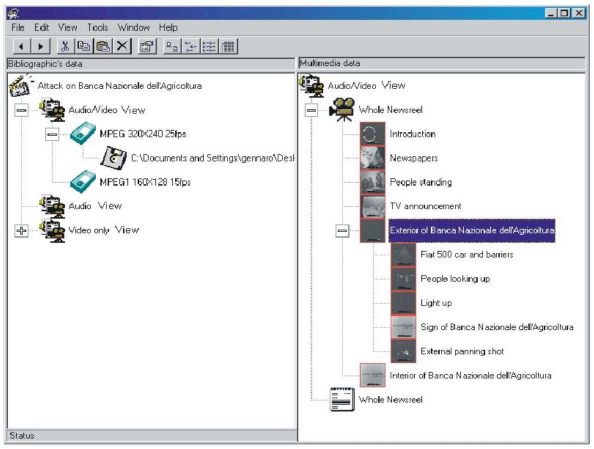

File Edit View Tools Window Help
Bibliographic's data
Attack on Banca Nazionale dell'Agricoltura
Audio/Video View
MPEG 320X240 25fps
C:\Documents and Settings\gennaro\Des
MPEG1 160X128 15fps
Audio View
Video only View
Multimedia data
Audio/Video View
Whole Newsreel
Introduction
Newspapers
People standing
TV announcement
Exterior of Banca Nazionale dell'Agricoltura
Fiat 500 car and barriers
People looking up
Light up
Sign of Banca Nazionale dell'Agricoltura
External panning shot
Interior of Banca Nazionale dell'Agricoltura
Whole Newsreel
Status

Figure 6: Navigating the Document Video View.

Video access
The video can be played by means of the video document interface provided by the editor. The editor offers user an interface similar to a VCR (see Figure 7).

MMEditor - Scholnet gateway

MMEditor Core

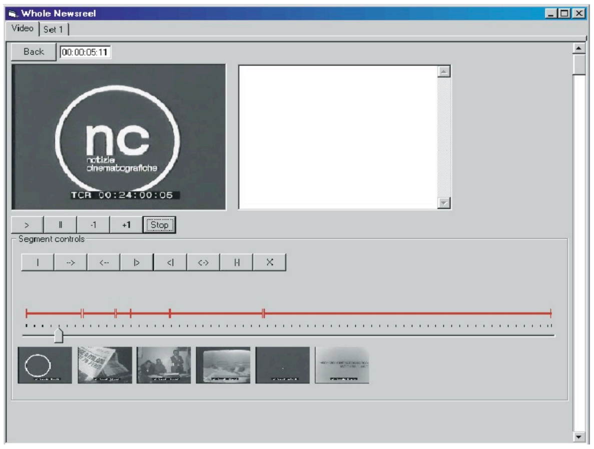


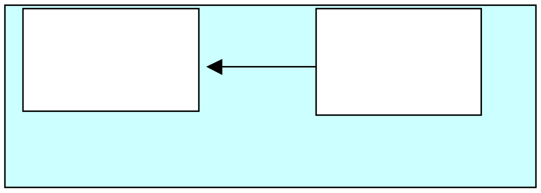
Figure 7: Playing the video and editing the View.

Editing of the audio/video document
The user can delete/add new segments in the Video Views using the same window as that used to play the video. (see Figure 7).

Save metadata description
The modified metadata can be saved locally for importing in the Scholnet database.

21.3 *ARCHITECTURAL CHOICES*

As the metadata editing phase will usually take some time, the editor will work off-line, i.e. the user will work on a local copy of the metadata. This simplifies the interface for communication between the editor and the Library Management Service and has the advantage of allowing the user to work on a connectionless machine.
The metadata editor will be composed of two different components (see the figure below).



The MMEditor Core will implement the functionality described in the previous section. It will not be hard-wired with a particular metadata attribute set; the metadata schema will be defined in the W3C XML Schema Definition (XSD) and will be used by the editor as the configuration file for the metadata model. The advantage of this choice is that it will be possible to adapt it to work with metadata formats that differ in their set of fields. This will be achieved by allowing the editor to recognise a subset of the types available for the XSD schemas: xsd:string, xsd:bool, xsd:date, xsd:float, xsd:integer and a set of new types to support the multimedia data.

The format chosen to exchange document metadata to/from the editor will be XML.

The MMEditor Core will be instantiated to work with the Scholnet metadata formats. Communication with the rest of the Scholnet architecture will take place via the MMEditor-Scholnet gateway. This gateway will transform the output produced by the MMEditor core instantiated for Scholnet into the format required by the submit request of the Library Management service.

22. POSSIBLE FUTURE EXTENSIONS

The design an extensible DL system is one of the main objectives of Scholnet. This has motivated our decision to define an open architecture and protocol in such great detail
In its basic version Scholnet offers a set of generic services to support scholarly communities. However, the current level of openness allows extensions to meet the more specific requirements dictated by a particular scholarly community. As an example of a possible extension, we discuss here how a service for supporting citation and linking might be added to the current version of the Scholnet system.

Reference linking is the ability to access cited documents immediately from the citing paper. Citation is the reverse facility, i.e. the possibility of knowing the list of documents citing a given paper. The most well-known system for supporting both services is ResearchIndex [ResearchIndex]. This service seeks to build a large on-line database of citation information in the field of computer science. An emerging application in the area of reference and citation is SFX [SFX]. This is an advanced reference linking system that takes the user's context into account. Another new application in this field has recently been implemented at Cornell University [Bergmark]. The aim of this service is to create a reference linking layer on the Web that provides sufficient data for a variety of value-added reference linking applications. This service partially exploits the ResearchIndex software to build valued-added surrogates to enhance the reference linking behaviour of Web documents. A surrogate is a digital library object that encapsulates reference linking information relating to one single item on the Web. In this service, reference linking is distributed across the collection of surrogate objects, and all the data relating to one item is grouped together within a single surrogate. The Reference Linking service functionality is accessible through a set of well-defined APIs, which enable access to the on-line document metadata and its references. These APIs are:

- getLinkedText – returns the content of the paper augmented with reference linking information
- getReferenceList – returns the references contained in this paper
- getMyData –returns the paper's own metadata
- getCurrentCitationList – returns the list of work citing this paper to the best of the surrogate's knowledge

A Reference Linking service might be implemented in Scholnet by integrating Cornell's Reference Linking service (CRLS) in the Scholnet architecture. The first step for this integration would be the construction of a software layer that transforms each APIs into a Scholnet protocol verb. At this point, several integration strategies are possible. The simplest is:

1. Create an instance of the CRLS service for each Repository service and associate the two kinds of services
2. Add a new view "surrogate" to the structure of the documents. This step is trivial since the whole Scholnet system is parametric with respect to the type of document views.
3. Modify the Library Management service so that, whenever it receives a submission, it asks to the CRLS service to create a surrogate manifestation. This manifestation is then stored as an external manifestation of the surrogate view in the Repository service.
4. Transform the CRLS service into a service that supports a DistributedInput replication typology. This is required if a global view of the citations of a document is required.

23. REFERENCES

[Bergmark]???manca nomi An Architecture for an Automatic Reference Linking, *Proc. of the 5th European Conference on Digital Library*, Lecture Notes in Computer Science 2163, Darmstadt, Germany, Sept 2001.
[DESIRE] *DESIRE metadata registry: a prototype registry developed as part of the EC funded DESIRE project*, http://www.ariadne.ac.uk/issue25/app-profiles/1
[Dienst] *Dienst Protocol*. http://www.cs.cornell.edu/cdlrg/dienst/protocols/DienstProtocol.htm
[ETRDL] ERCIM Technical Reference Digital Library. http://dienst.iei.pi.cnr.it/
[Heery00] Rachel Heery and Manjula Patel. *Application profiles: mixing and matching metadata schemas, October 2000,* http://www.ariadne.ac.uk/issue25/app-profiles/
[OAI] *Open Archives Initiatives*. http://www.openarchives.org/
[Research Index] S. Lowrence, C.L. Giles and K. Bollacker, Digital Libraries and autonomous citation indexing, *IEEE Computer 32 (6)* ,1999, http://www.researchindex.com.
[SFX] H. Van de Sompel and P. Hochstembach, Reference linking in an hybrid library environment, part 2:SFX, a generic linking solution, *D-Lib Magazine 5 (4),*1999.
[ScholnetD2.1.1] *Functionality and Efficiency Tests Report*: Scholnet Project IST-199-20664, Report D2.1.1. June 2001.
[ScholnetTA] *A Digital Library Testbed to Support Networked Scholarly Communities*. ScholnetProject IST-199-20664 Technical Annex, 2000.
[UML] Hans-Erik Eriksson and Magnus Penker. UML Toolkit. John Wiley & Sons, Inc.

24. APPENDIX A: LIST OF ACRONYMS

DoMDL	Document Model for Digital Library
DC	Dublin Core
DCQ	Dublin Core Qualified
ETRDL	ERCIM Technical Reference Digital Library
HAS	Hypermedia Annotation Service
GUI	Graphical User Interface
LibMgt	Library Management Service
MDS	Multimedia Document Storage and Delivery Service
MIME	Multipurpose Internet Mail Extensions
MTS	Multilingual Thesaurus Service
OAI	Open Archives Initiative
OLAP	OpenDLib Application Profile
OLCM	OpenDLib Collection Metadata profile
OLGM	OpenDLib Group Metadata profile
OLMS	OpenDLib Metadata Set
OLUM	OpenDLib User Metadata profile
OLP	OpenDLib protocol
PIDS	Personalised Information Dissemination Service
QM	Query Mediator Service
SIS-TM	SIS Thesaurus Management System
SMS	Structure Metadata Set
URI	Unique Resource Identifier
URN	Uniform Resource Name

25. APPENDIX B: THE OPENDLIB APPLICATION PROFILE

This Appendix lists the fields of the OpenDLib Architecture Profile (OLAP), their schemas (when their values are taken from existing controlled vocabularies) and their description. It also provides an XML DTD for OLAP.

The OLAP fields and their definition

Element Name	Scheme	Element definition
dc:title		A name given to the resource. In the case of OLA documents we expect that this is written in English. When the document is in another language, this field contains a translation of the title in English.
dc:creator		The entity primarily responsible for the content of the resource.
dc:subject		The topic of the content of the resource
dc:description		A description of the content of the resource
dc.contributor		An entity making contributions to the content of the resource.
dc:publisher		An entity responsible for making the resource available
dc:date		A date associated with an event in the life cycle of the resource.
dc:type		The nature or genre of the content of the resource
dc:format		The physical or digital manifestation of the resource
dc:identifier		An unambiguous reference to the resource within a given context
dc:source		A reference to a resource from which the present resource is derived.
dc:language		A language of the intellectual content of the resource
dc.relation		A reference to a related resource
dc:coverage		The extent or scope of the content of the resource
dc:rights		Information about rights held in and over the resource
dcq:title.alternative		Any form of the title used as a substitute or alternative to the formal title of the resource. In the specific case of OLA documents this field contains the title if this is in a language different from English.
dcq:description.abstract		A summary of the content of the resource. In the case of OLA documents we expect that this is written in English
dcq:description.tableofcontents		A list of subunits of the content of the resource
dcq:date.issued		Date of formal issuance (e.g.,

		publication) of the resource
dcq:date.available		Date (often a range) when the resource will become or did become available
dcq:date.created		Date of creation of the resource
dcq:date.valid		Date (often a range) of validity of a resource
dcq:date.modified		Date on which the resource was changed
dcq:coverage.temporal		Temporal characteristics of the intellectual content of the resource
dcq:coverage.spatial		Spatial characteristics of the intellectual content of the resource
dcq:subject.keyword		
dcq:subject	CCS	
	MSC	
olms:subject.keyword.alternative		
olms:subject.alternative	CCS	
	MSC	
olms:description.abstract.alternative		This field has the same meaning as dcq.description.abstract except that it contains the abstract written in a language different from English.
olms:description.tableofcontents.alternative		
olms:note		Textual notes about the document
olms:note.retrieval		
olms:note.withdraw		
olms:note.contact		
olms:note.collection		
olms:note.series		

The OLAP DTD

```
<!ENTITY dcns 'http://purl.org/dc/elements/1.1/'>
<!ENTITY dcqns 'http://purl.org/dc/terms/'>
<!ENTITY olmsns 'http://project.iei.pi.cnr.it:8221/OLP/htdocs/NameSpace/olms.xml'>
<!-- Declare convenience entities for XML namespace declarations -->
<!ENTITY % dcnsdecl 'xmlns:dc CDATA #FIXED "&dcns;"'>
<!ENTITY % dcqnsdecl 'xmlns:dcq CDATA #FIXED "&dcqns;"'>
<!ENTITY % olmsnsdecl 'xmlns:olms CDATA #FIXED "&olmsns;"'>
<!ELEMENT olms:ol (dc:title+, olms:title.alternative*, dc:creator+, dc:subject+, dcq:subject.keyword*,
dcq:subject.ccs*, dcq:subject.msc*, olms:subject.alternative*, olms:subject.keyword.alternative*,
olms:subject.ccs.alternative*, olms:subject.msc.alternative*, dc:description*, dcq:description.abstract+,
olms:description.abstract.alternative*, dcq:description.toc*, olms:description.toc.alternative*, dc:contributor*,
dc:publisher+, dc:date*, dcq:date.issued+, dcq:date.available+, dcq:date.created*, dcq:date.valid*,
dcq:date.modified*, dc:type+, dc:format*, dc:identifier*, dc:source*, dc:language+, dc:relation*, dc:coverage*,
dcq:coverage.temporal*, dcq:coverage.spatial*, dc:rights*, olms:note*, olms:note.retrieval*,
olms:note.withdraw*, olms:note.contact*, olms:note.collection+, olms:note.series*, olms:note.revision*)>
<!ATTLIST olms:ol
    xmlns:dc CDATA #FIXED "http://purl.org/dc/elements/1.1/"
    xmlns:dcq CDATA #FIXED "http:// purl.org/dc/elements/1.1/"
    xmlns:olms CDATA #FIXED "http:// project.iei.pi.cnr.it:8221/OLP/htdocs/NameSpace/olms.xml"'
>
<!ELEMENT dc:title (#PCDATA)>
<!ELEMENT dc:creator (#PCDATA)>
<!ELEMENT dc:subject (#PCDATA)>
<!ELEMENT dc:description (#PCDATA)>
```

```
<!ELEMENT dc:contributor (#PCDATA)>
<!ELEMENT dc:publisher (#PCDATA)>
<!ELEMENT dc:date (#PCDATA)>
<!ELEMENT dc:type (#PCDATA)>
<!ELEMENT dc:format (#PCDATA)>
<!ELEMENT dc:identifier (#PCDATA)>
<!ELEMENT dc:source (#PCDATA)>
<!ELEMENT dc:language (#PCDATA)>
<!ELEMENT dc:relation (#PCDATA)>
<!ELEMENT dc:coverage (#PCDATA)>
<!ELEMENT dc:rights (#PCDATA)>
<!ELEMENT dcq:subject.keyword (#PCDATA)>
<!ELEMENT dcq:subject.ccs (#PCDATA)>
<!ELEMENT dcq:subject.msc (#PCDATA)>
<!ELEMENT dcq:description.abstract (#PCDATA)>
<!ELEMENT dcq:description.toc (#PCDATA)>
<!ELEMENT dcq:date.issued (#PCDATA)>
<!ELEMENT dcq:date.available (#PCDATA)>
<!ELEMENT dcq:date.created (#PCDATA)>
<!ELEMENT dcq:date.valid (#PCDATA)>
<!ELEMENT dcq:date.modified (#PCDATA)>
<!ELEMENT dcq:coverage.temporal (#PCDATA)>
<!ELEMENT dcq:coverage.spatial (#PCDATA)>
<!ELEMENT olms:title.alternative (#PCDATA)>
<!ELEMENT olms:subject.alternative (#PCDATA)>
<!ELEMENT olms:subject.keyword.alternative (#PCDATA)>
<!ELEMENT olms:subject.ccs.alternative (#PCDATA)>
<!ELEMENT olms:subject.msc.alternative (#PCDATA)>
<!ELEMENT olms:description.abstract.alternative (#PCDATA)>
<!ELEMENT olms:description.toc.alternative (#PCDATA)>
<!ELEMENT olms:note (#PCDATA)>
<!ELEMENT olms:note.retrieval (#PCDATA)>
<!ELEMENT olms:note.withdraw (#PCDATA)>
<!ELEMENT olms:note.contact (#PCDATA)>
<!ELEMENT olms:note.collection (#PCDATA)>
<!ELEMENT olms:note.series (#PCDATA)>
<!ELEMENT olms:note.revision (#PCDATA)>
```

26. APPENDIX C: MAPPING THE ETRDL METADATA FORMAT INTO OLMS

This Appendix reports the mapping between the ETRDL supported metadata format and OLAP. This mapping is used in order to transform the metadata descriptions of the documents handled by the ETRDL digital library into OLAP valid metadata descriptions.

ETRDL Formats		OLAP	
Element name	Type	Element name	Type
Id	URN	dc:identifier	URI
Entry	Month_alphabetic, Day, Year	dcq:date.available	ISO8601
Organization	String	dc:publisher	string
Title	String	dc:title	string
Type	String	dc:type	string
Revision	String	olms:note.revision	string
Withdraw	String	olms:note.withdraw	string
Author	String	dc:creator	string
Corp-author	String	dc:contributor	string
Contact	String	olms:note.contact	string
Date	Month_alphabetic, Day, Year	dcq:date.issued	ISO8601
Other_Access	URL	dc:identifier	URI
Retrieval	String	olms:note.retrieval	string
Keyword	String	dcq:subject.keyword	string
ACM	String	dcq:subject.ccs	string
MSC	String	dcq:subject.msc	string
CR-Category	String	dc:subject	string
Period	String	dcq:coverage.temporal	W3C-DTF
Series	String	olms:note.series	string
Language	String	dc:language	RFC 1766
Notes	String	olms:note.text	string
Abstract	String	dcq:description.abstract	string
LocAbstract	String	olms:description.abstract.alternative	string

27. APPENDIX D: THE STRUCTURE METADATA SET

This Appendix reports the DTDs that specify the Structure Metadata Set. The SMS describes the structure of a DoMDL document.

SMS DTD

```
<!ELEMENT Structure (document+)>
<!ELEMENT document (view+)>
<!ATTLIST Structure
    version CDATA #REQUIRED
>
<!ATTLIST document
    handle CDATA #REQUIRED
    version CDATA #REQUIRED
>
<!ENTITY % view.dtd SYSTEM "http://labserv.iei.pi.cnr.it/OLP/htdocs/sms/DTD/view.dtd">
%view.dtd;
```

VIEW DTD

```
<!--ATTRIBUTES min, max, ord e ref_version: integer; ATTRIBUTE multiple: boolean-->
<!ELEMENT view ((manifestation+ | view*)*, (manifestation+ | view*)*)*>
<!ATTLIST view
    name CDATA #REQUIRED
    type (metadata | body | reference | choice) #REQUIRED
    display CDATA #IMPLIED
    min CDATA #IMPLIED
    max CDATA #IMPLIED
    ord CDATA #IMPLIED
    ref-handle CDATA #IMPLIED
    ref-version CDATA #IMPLIED
    ref-view CDATA #IMPLIED
    transcoding (no | yes) #REQUIRED
    downloading (no | yes) #REQUIRED
    delivering (no | yes) #REQUIRED
>
<!ENTITY % manifestation.dtd SYSTEM "http://labserv.iei.pi.cnr.it/OLP/htdocs/sms/DTD\manifestation.dtd">
%manifestation.dtd;
```

MANIFESTATION DTD

```
<!ELEMENT manifestation EMPTY>
<!ATTLIST manifestation
    name (bib | olms | rfc1807 | html | postscript | xml | gif | bib | pdf | postscript-part | inline | inline-part |
composite-part | ) #REQUIRED
    content-type CDATA #REQUIRED
    display CDATA #IMPLIED
    URI CDATA #IMPLIED
    type (inside | outside) #IMPLIED
    min CDATA #IMPLIED
    max CDATA #IMPLIED
    multiple (1 | 0) #IMPLIED
    size CDATA #IMPLIED
>
```

28. APPENDIX E: THE OPENDLIB USER METADATA PROFILE

This Appendix lists the fields of the OpenDLib User Metadata (OLUM) profile. It also provides it with an XML DTD.

The OLUM fields and their definition

	Attribute	Obligation	Type	Constraint/Comment
SETTINGS	Login	Mandatory	String	alphanumeric characters plus underscore, unique in the system
	Password	Mandatory	String	user's password to the system
	FullName	Mandatory	String	user's first name(s) and surname
	Email	Mandatory	String	valid e-mail contact for the user
	Institution	Optional	String	user's institution
	Address	Optional	String	contact address for the user
	Phone	Optional	String	contact phone for the user
	Comment	Optional	String	any text
DATA	HandleList	Optional	Set	set of document handles associated with the user (the user is the owner of these documents)
RIGHTS	Authorities	Optional	Set	set of authorities in which the user can submit documents. If the set is empty the user cannot submit documents.
	Collection-Adm	Optional	Boolean	specifies whether the user can administer the Collection Service
	LibMgt-Adm	Optional	Set	specifies the set of authorities that can be administered with the Library Management Service
	Repository-Adm	Optional	Set	specifies the set of authorities that can be administered
	Registry-Adm	Optional	Boolean	specifies whether the user can administer the Registry Service
	Thesaurus-Adm	Optional	Boolean	specifies whether the user can administer the Thesaurus Service

The OLUM DTD

```
<!ELEMENT login (#PCDATA)>
<!ELEMENT fullname (#PCDATA)>
<!ELEMENT institution (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT authorities (authority+)>
<!ELEMENT repository-adm (authorities+)>
<!ELEMENT libmgt-adm (authorities+)>
<!ELEMENT authority (#PCDATA)>
<!ELEMENT documents (handle+)>
```

```
<!ELEMENT handle (#PCDATA)>
<!ELEMENT collection-adm EMPTY>
<!ATTLIST collection-adm
    value (0 | 1) #REQUIRED
>
<!ELEMENT registry-adm EMPTY>
<!ATTLIST registry-adm
    value (0 | 1) #REQUIRED
>
<!ELEMENT thesaurus-adm EMPTY>
<!ATTLIST thesaurus -adm
    value (0 | 1) #REQUIRED
>
<!ELEMENT userprofile (login, fullname, email, institution, address, phone, comment, documents,
authorities, repository-adm, libmgt-adm, collection-adm, registry-adm, thesaurus-adm)>
```

29. APPENDIX F: THE OPENDLIB GROUP METADATA PROFILE

This Appendix lists the fields of the OpenDLib Group Metadata (OLGM) profile. It also provides it with an XML DTD.

The OLGM fields and their definition

	Attribute	Obligation	Type	Constraint/Comment
SETTINGS	GroupName	Mandatory	String	alphanumeric characters plus underscore, unique in the system
	Description	Mandatory	String	short description of the group
	Owner	Mandatory	String	login of the group owner, must already be registered
	UserList	Optional	Set	login of each registered user belonging to the group
	Comment	Optional	String	any text
	Public	Mandatory	Boolean	default value is true.

The OLGM DTD

```
<!ELEMENT groupname (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT owner (#PCDATA)>
<!ELEMENT comment EMPTY>
<!ELEMENT public EMPTY>
<!ATTLIST public
    value (0 | 1) #REQUIRED
>
<!ELEMENT userlist (login+)>
<!ELEMENT login (#PCDATA)>
<!ELEMENT groupprofile (groupname, description, owner, comment, public, userlist)>
```

30. APPENDIX G: THE OPENDLIB COLLECTION METADATA PROFILE

This Appendix lists the fields of the OpenDLib Collection Metadata (OLCM) profile. It also provides it with an XML DTD.

The OLCM fields and their definition

Element Name	Scheme	Element definition
collectionId		The unique identifier of the collection
name		The name of the collection
description		The textual description of the collection
subject		A list of free text keywords that describe the subject of the collection
owner		The login of the creator of the collection
membershipCondition		A set of pairs (Authority, Condition) with the following meaning: Authority: authority that publishes documents in the collection Condition: filter for selecting the documents of the collection among those published by the authority
services		A set of service descriptions. Each service description reports the name of the service and its verbs (with the specification of the version) that can be used on the collection
parentCollection		The parent collection CollectionId. This information is used to build a hierarchy between collections in order to present them in a more usable and meaningful way to the end-user

The OLCM DTD

```
<!ELEMENT collection (owner, services, parent-collection, membership-condition)>
<!ATTLIST collection
    name CDATA #REQUIRED
    description CDATA #REQUIRED
    subject CDATA #IMPLIED
    id CDATA #REQUIRED
>
<!ELEMENT owner (login, passwd)>
<!ELEMENT login (#PCDATA)>
<!ELEMENT passwd (#PCDATA)>
<!ELEMENT services (service+)>
<!ATTLIST service
    name CDATA #REQUIRED
>
<!ELEMENT service (verb+)>
<!ELEMENT verb EMPTY>
<!ATTLIST verb
    name CDATA #REQUIRED
    version CDATA #IMPLIED
```

```
>
<!ELEMENT parent-collection EMPTY>
<!ATTLIST parent-collection
    id CDATA #REQUIRED
>
<!ELEMENT membership-condition (pair+)>
<!ELEMENT pair (authorities, condition)>
<!ELEMENT authorities (authority+)>
<!ELEMENT authority EMPTY>
<!ATTLIST authority
    name CDATA #REQUIRED
>
<!ELEMENT condition (leaf, op?, condition?)>
<!ATTLIST condition
    value CDATA #REQUIRED
>
<!ELEMENT leaf (#PCDATA)>
<!ELEMENT op (#PCDATA)>
```

31. APPENDIX H: THE THESAURUS CONCEPT DTD

This Appendix describes the SIS Thesaurus Management System (SIS-TMS). SIS-TMS consists of a tool to develop multilingual thesauri. The basic notion of SIS-TMS is a thesaurus concept which is described by an XML document

```
<!ELEMENT Concept (Definition , Intra_Thesaurus , Inter_Thesauri , Administration_Description)>
<!ELEMENT Definition (Name , Scope_Note , Thesaurus , Type , Hierarchy*)>
<!ELEMENT Intra_Thesaurus (Broader_Term , Narrower_Term , Related_Term , Alternative_Term ,
Used_For_Term)>
<!ELEMENT Inter_Thesauri (Exact_Equivalence* , Broader_Equivalence* , Narrower_Equivalence* ,
Inexact_Equivalence*)>
<!ELEMENT Administration_Description (Created_By , Last_Modified , Found_In , Not_Found_In)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Scope_Note (#PCDATA)>
<!ELEMENT Thesaurus (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT Hierarchy (#PCDATA)>
<!ELEMENT Broader_Term (Concept_Name*)>
<!ELEMENT Narrower_Term (Concept_Name *)>
<!ELEMENT Related_Term (Concept_Name *)>
<!ELEMENT Alternative_Term (English | French | Italian)>
<!ELEMENT Used_For_Term (English , French , Italian)>
<!ELEMENT English (term*)>
<!ELEMENT French (term*)>
<!ELEMENT Italian (term*)>
<!ELEMENT Concept_Name (#PCDATA)>
<!ELEMENT Union (Intersection* , Concept_Name*)>
<!ELEMENT Intersection (Concept_Name*)>
<!ELEMENT Exact_Equivalence (Thesaurus , (Concept_Name | Union | Intersection))>
<!ELEMENT Broader_Equivalence (Thesaurus , (Concept_Name | Union | Intersection))>
<!ELEMENT Narrower_Equivalence (Thesaurus , (Concept_Name | Union | Intersection))>
<!ELEMENT Inexact_Equivalence (Thesaurus , Concept_Name)*>
<!ELEMENT Created_By (Date , Time , Creator)>
<!ELEMENT Last_Modified (Date , Time , Creator)>
<!ELEMENT Found_In (Source)>
<!ELEMENT Not_Found_In (Source)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Time (#PCDATA)>
<!ELEMENT Creator (#PCDATA)>
<!ELEMENT Source (Author , Title , Publisher , ISBN)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Belongs_To (#PCDATA)>
<!ELEMENT term (#PCDATA)>
```

32. APPENDIX I: DTD

This Appendix contains the DTDs that specify the format of the response of each verb.

REPOSITORY

Identify

```
<!ELEMENT Identify (serviceName, olp_base_url, metaServiceURL, textualDescription, protocolVersion, adminEmail, verbsInfo,
submissionProcedure, harvestInformation, useRestrictions, contentInfo)>
<!ATTLIST Identify
      version CDATA #REQUIRED
>
<!ELEMENT adminEmail (#PCDATA)>
<!ELEMENT authorities (description, olp_base_url)>
<!ELEMENT contentDescription (#PCDATA)>
<!ELEMENT contentInfo (contentDescription, authorities, sets, metadataFormats, documentStructure)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT documentStructure (description, olp_base_url, olp_dtd_url)>
<!ELEMENT harvestInformation (description, olp_base_url)>
<!ELEMENT metaServiceURL (#PCDATA)>
<!ELEMENT metadataFormats (description, olp_base_url, olp_dtd_url)>
<!ELEMENT olp_base_url (#PCDATA)>
<!ELEMENT olp_dtd_url (#PCDATA)>
<!ELEMENT protocolVersion (#PCDATA)>
<!ELEMENT serviceName (#PCDATA)>
<!ELEMENT sets (description, olp_base_url)>
<!ELEMENT submissionProcedure (description, olp_base_url)>
<!ELEMENT textualDescription EMPTY>
<!ELEMENT useRestrictions (#PCDATA)>
<!ELEMENT verbsDescription (description, olp_base_url)>
<!ELEMENT verbsInfo (verbsSupported, verbsDescription)>
<!ELEMENT verbsSupported (description, olp_base_url)>
```

ListVerbs

```
<!ELEMENT ListVerbs (verb+)>
<!ATTLIST ListVerbs
     version CDATA #REQUIRED
>
<!ELEMENT verb (#PCDATA)>
```

DescribeVerb

```
<!ELEMENT DescribeVerb (verb)>
<!ELEMENT verb (description, versions+)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT versions (version+)>
<!ELEMENT version ((arguments | returns+ | example* | note*), (arguments | returns+ | example* | note*))*>
<!ELEMENT arguments (fixed?, keyword?)>
<!ELEMENT returns (#PCDATA)>
<!ELEMENT example (#PCDATA)>
<!ELEMENT note (#PCDATA)>
<!ELEMENT fixed (arg*)>
<!ELEMENT keyword (arg*)>
<!ELEMENT arg EMPTY>
<!ATTLIST DescribeVerb
     version CDATA #REQUIRED
>
<!ATTLIST verb
     name (Identify | ListVerbs | DescribeVerb | ListAuthorities | ListSets | ListBinders | ListEncodings |
ListMetaFormats | ListSubmissionFormats | ListContents | ListVersions | Terms | Structure | Manifestations |
Disseminate | Submit | NewVersion | Withdraw) #REQUIRED
>
<!ATTLIST version
     id CDATA #REQUIRED
>
```

```
<!ATTLIST arg
    name CDATA #REQUIRED
>
```

ListAuthorities

```
<!ELEMENT ListAuthorities (authority+)>
<!ELEMENT authority (allowed-sets)>
<!ELEMENT allowed-sets (set+)>
<!ELEMENT set EMPTY>
<!ATTLIST ListAuthorities
    version CDATA #REQUIRED
>
<!ATTLIST authority
    name CDATA #REQUIRED
    display CDATA #REQUIRED
>
<!ATTLIST set
    name CDATA #REQUIRED
>
```

ListSets

```
<!ELEMENT ListSets (set+)>
<!ELEMENT set ((display | allowed-authorities | set*), (display | allowed-authorities | set*))*>
<!ELEMENT allowed-authorities (authority+)>
<!ELEMENT display (#PCDATA)>
<!ELEMENT authority EMPTY>
<!ATTLIST ListSets
    version CDATA #REQUIRED
>
<!ATTLIST authority
    name CDATA #REQUIRED
>
<!ATTLIST set
    name CDATA #REQUIRED
>
```

ListBinders

```
<!ELEMENT ListBinders (binder+)>
<!ELEMENT binder (#PCDATA)>
<!ATTLIST ListBinders
    version CDATA #REQUIRED
>
```

ListEncodings

```
<!ELEMENT encoding (#PCDATA)>
<!ELEMENT ListEncodings (encoding+)>
<!ATTLIST ListEncodings
    version CDATA #REQUIRED
>
```

ListMetaFormats

```
<!ELEMENT ListMetaFormats (meta_format+)>
<!ATTLIST ListMetaFormats
    version CDATA #REQUIRED
>
<!ELEMENT meta_format (namespace+)>
<!ATTLIST meta_format
    name (bib | olms | rfc1807) #REQUIRED
    dtd CDATA #IMPLIED
```

```
>
<!ELEMENT namespace EMPTY>
<!ATTLIST namespace
    name (bibns | dc | dcq | olms | rfc1807) #REQUIRED
    uri CDATA #REQUIRED
>
```

ListSubmissionFormats

```
<!ELEMENT ListSubmissionFormats (manifestation+)>
<!ATTLIST ListSubmissionFormats
    version CDATA #REQUIRED
>
<!ENTITY % manifestation.dtd SYSTEM "http://labserv.iei.pi.cnr.it/OLP/htdocs/sms/DTD/manifestation.dtd">
%manifestation.dtd;

<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT manifestation EMPTY>
<!ATTLIST manifestation
    name (doc | html | postscript | xml | gif | bib | pdf | postscript-pageimage | inline) #REQUIRED
    content-type CDATA #REQUIRED
    display CDATA #IMPLIED
    filename CDATA #IMPLIED
    min CDATA #IMPLIED
    max CDATA #IMPLIED
    multiple (1 | 0) #IMPLIED
    size CDATA #IMPLIED
>
```

ListContents

```
<!ELEMENT ListContents (document+)>
<!ELEMENT document (#PCDATA)>
<!ATTLIST ListContents
    count CDATA #IMPLIED
    version CDATA #REQUIRED
>
<!ATTLIST document
    handle CDATA #REQUIRED
>
```

ListVersions

```
<!ELEMENT ListVersions (version+)>
<!ELEMENT version ((date | comment), (date | comment))*>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ATTLIST ListVersions
    version CDATA #REQUIRED
>
<!ATTLIST version
    id CDATA #REQUIRED
>
```

Terms

```
<!ELEMENT Terms (text)>
<!ATTLIST Terms
    version CDATA #REQUIRED
>
<!ELEMENT text (#PCDATA)>
```

Structure

```
<!ELEMENT Structure (document+)>
<!ELEMENT document (view+)>
<!ATTLIST Structure
    version CDATA #REQUIRED
>
<!ATTLIST document
    handle CDATA #REQUIRED
    version CDATA #REQUIRED
>
<!ENTITY % view.dtd SYSTEM "http://labserv.iei.pi.cnr.it/OLP/htdocs/sms/DTD/view.dtd">
%view.dtd;

<!--ATTRIBUTES min, max, ord e ref_version: integer; ATTRIBUTE multiple: boolean-->
<!ELEMENT view ((manifestation+ | view*)*, (manifestation+ | view*)*)*>
<!ATTLIST view
    name CDATA #REQUIRED
    type (metadata | body | reference | choice) #REQUIRED
    display CDATA #IMPLIED
    min CDATA #IMPLIED
    max CDATA #IMPLIED
    ord CDATA #IMPLIED
    ref-handle CDATA #IMPLIED
    ref-version CDATA #IMPLIED
    ref-view CDATA #IMPLIED
>
<!ENTITY % manifestation.dtd SYSTEM "http://labserv.iei.pi.cnr.it/OLP/htdocs/sms/DTD\manifestation.dtd">
%manifestation.dtd;

<!ELEMENT manifestation EMPTY>
<!ATTLIST manifestation
    name (bib | olms | rfc1807 | html | postscript | xml | gif | bib | pdf | postscript-part | inline | inline-part |
composite-part | ) #REQUIRED
    content-type CDATA #REQUIRED
    display CDATA #IMPLIED
    filename CDATA #IMPLIED
    min CDATA #IMPLIED
    max CDATA #IMPLIED
    multiple (1 | 0) #IMPLIED
    size CDATA #IMPLIED
>
```

Manifestations

```
<!ELEMENT Manifestations (document)>
<!ELEMENT document (manifestation+)>
<!ATTLIST Manifestations
    version CDATA #REQUIRED
>
<!ATTLIST document
    handle CDATA #REQUIRED
    version CDATA #REQUIRED
>
<!ENTITY % manifestation.dtd SYSTEM "http://labserv.iei.pi.cnr.it/OLP/htdocs/sms/DTD\manifestation.dtd">
%manifestation.dtd;
```

Submit

```
<!ELEMENT Submit ((handle | set+), (handle | set+))*>
<!ELEMENT handle (#PCDATA)>
<!ELEMENT set ((display | set?), (display | set?))*>
<!ELEMENT display (#PCDATA)>
<!ATTLIST Submit
    version CDATA #REQUIRED
```

```
>
<!ATTLIST set
    name CDATA #REQUIRED
>
```

NewVersion

```
<!ELEMENT NewVersion (version)>
<!ELEMENT version ((date | comment*), (date | comment*))*>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ATTLIST NewVersion
    version CDATA #REQUIRED
>
<!ATTLIST version
    id CDATA #REQUIRED
>
```

Withdraw

```
<!ELEMENT Withdraw (handle)>
<!ELEMENT handle (#PCDATA)>
<!ATTLIST Withdraw
    version CDATA #REQUIRED
>
```

Library Management

Identify

```
<!ELEMENT Identify ((service-name | base-url | protocol-version | admin-email | descriptions*), (service-
name | base-url | protocol-version | admin-email | descriptions?))*>
<!ATTLIST Identify
    version CDATA #REQUIRED
>
<!ELEMENT admin-email (#PCDATA)>
<!ELEMENT base-url (#PCDATA)>
<!ELEMENT descriptions EMPTY>
<!ELEMENT protocol-version (#PCDATA)>
<!ELEMENT service-name (#PCDATA)>
```

ListVerbs

```
<!ELEMENT ListVerbs (verb+)>
<!ATTLIST ListVerbs
    version CDATA #REQUIRED
>
<!ELEMENT verb (#PCDATA)>
```

DescribeVerb

```
<!ELEMENT DescribeVerb (verb)>
<!ELEMENT verb (description, versions+)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT versions (version+)>
<!ELEMENT version ((arguments | returns+ | example* | note*), (arguments | returns+ | example* | note*))*>
<!ELEMENT arguments (fixed?, keyword?)>
<!ELEMENT returns (#PCDATA)>
<!ELEMENT example (#PCDATA)>
<!ELEMENT note (#PCDATA)>
<!ELEMENT fixed (arg*)>
<!ELEMENT keyword (arg*)>
<!ELEMENT arg EMPTY>
<!ATTLIST DescribeVerb
```

```
    version CDATA #REQUIRED
>
<!ATTLIST verb
    name (Identify | ListVerbs | DescribeVerb | AdminLoginForm | AdminLogin | IncomingSubmitForm |
ApproveSubmit | IncomingWithdrawForm | ApproveWithdraw | RejectForm | Reject | ShowBibRecord |
ShowStructureRecord | UserSubmitForm | UserSubmit | UserEditForm | UserSubmitNewVersion |
UserWithdrawForm | UserWithdraw) #REQUIRED
>
<!ATTLIST version
    id CDATA #REQUIRED
>
<!ATTLIST arg
    name CDATA #REQUIRED
>
```

ApproveSubmit

```
<!ELEMENT ApproveSubmit ((handle | set), (handle | set))*>
<!ATTLIST ApproveSubmit
    version CDATA #REQUIRED
>
<!ELEMENT display (#PCDATA)>
<!ELEMENT handle (#PCDATA)>
<!ELEMENT set (display, set*)>
<!ATTLIST set
    name CDATA #REQUIRED
>
```

ApproveWithdraw

```
<!ELEMENT ApproveWithdraw (handle)>
<!ATTLIST ApproveWithdraw
    version CDATA #REQUIRED
>
<!ELEMENT handle (#PCDATA)>
```

Reject

```
<!ELEMENT Reject ((title | type), (title | type))*>
<!ATTLIST Reject
    version CDATA #REQUIRED
>
<!ELEMENT title (#PCDATA)>
<!ELEMENT type (#PCDATA)>
```

UserWithdraw

```
<!ELEMENT UserWithdraw ((handle | type), (handle | type))*>
<!ATTLIST UserWithdraw
    version CDATA #REQUIRED
>
<!ELEMENT handle (#PCDATA)>
<!ELEMENT type (#PCDATA)>
```

UserSubmit

```
<!ELEMENT UserSubmit (doc-id)>
<!ATTLIST UserSubmit
    version CDATA #REQUIRED
>
<!ELEMENT doc-id (#PCDATA)>
```

UserSubmitNewVersion

```
<!ELEMENT UserSubmitNewVersion (doc-id)>
```

```
<!ATTLIST UserSubmitNewVersion
    version CDATA #REQUIRED
>
<!ELEMENT doc-id (#PCDATA)>
```

Personalisation Handler

Identify

```
<!ELEMENT Identify ((serviceName | baseURL | protocolVersion | adminEmail | description?), (serviceName
| baseURL | protocolVersion | adminEmail | description?))*>
<!ELEMENT serviceName (#PCDATA)>
<!ELEMENT baseURL (#PCDATA)>
<!ELEMENT protocolVersion (#PCDATA)>
<!ELEMENT adminEmail (#PCDATA)>
<!ELEMENT description (contentDescription)>
<!ELEMENT contentDescription (#PCDATA)>
<!ATTLIST Identify
    version CDATA #REQUIRED
>
```

ListVerbs

```
<!ELEMENT ListVerbs (verb+)>
<!ATTLIST ListVerbs
    version CDATA #REQUIRED>
<!ELEMENT verb (#PCDATA)>
```

DescribVerb

```
<!ELEMENT DescribeVerb (verb)>
<!ELEMENT verb (description, versions+)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT versions (version+)>
<!ELEMENT version ((arguments | returns+ | example* | note*),(arguments | returns+ | example* | note*))*>
<!ELEMENT arguments (fixed?, keyword?)>
<!ELEMENT returns (#PCDATA)>
<!ELEMENT example (#PCDATA)>
<!ELEMENT note (#PCDATA)>
<!ELEMENT fixed (arg*)>
<!ELEMENT keyword (arg*)>
<!ELEMENT arg EMPTY>
<!ATTLIST DescribeVerb
    version CDATA #REQUIRED
>
<!ATTLIST verb
    name (Identify | ListVerbs | DescribeVerb | MatchMetadata | CreateTopic | ListTopics | EditTopic |
    DeleteTopic) #REQUIRED
>
<!ATTLIST version
    id CDATA #REQUIRED
>
<!ATTLIST arg
    name CDATA #REQUIRED
>
```

Topic

```
<!ELEMENT topic EMPTY>
<!ATTLIST topic
    topic_id CDATA #REQUIRED
    topic_label CDATA #REQUIRED
    keywords CDATA #IMPLIED
    acm_categories CDATA #IMPLIED
    notification_flag (1 | 0) #REQUIRED
>
```

ListTopics

```
<!ELEMENT ListTopics (topic*)>
<!ATTLIST ListTopics
    version CDATA #REQUIRED
>
<!ENTITY % topic.dtd SYSTEM "http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/Dtd/\topic.dtd">
%topic.dtd;
```

EditTopic

```
<!ELEMENT EditTopic (topic)>
<!ATTLIST EditTopic
    version CDATA #REQUIRED
>
<!ENTITY % topic.dtd SYSTEM "http://labserv.iei.pi.cnr.it/OLP/htdocs/olms/Dtd/\topic.dtd">
%topic.dtd;
```

ANNOTATION

DisplayAnnotations

```
<!ELEMENT annotatedObject (document | annotation | docURL)>
<!ELEMENT document (handle)>
<!ELEMENT annotation (handle, description , userinfo , links , date)>
<!ELEMENT docURL (handle)>
<!ELEMENT handle (#PCDATA)>
<!ELEMENT description (type , subject* , text , program*)>
<!ELEMENT userinfo (author , project , group)>
<!ELEMENT links (linktype+, tovalue+)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT text (#PCDATA)>
<!ELEMENT program (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT project (#PCDATA)>
<!ELEMENT group (#PCDATA)>
<!ELEMENT linktype (#PCDATA)>
<!ELEMENT tovalue (#PCDATA)>
<!ELEMENT date (#PCDATA)>
```