Date: August 2004

Life Sciences Identifiers

OMG Available Specification

dtc/04-08-02

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED

HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 250 First Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

The OMG Object Management Group Logo®, CORBA®, CORBA Academy®, The Information Brokerage®, XMI® and IIOP® are registered trademarks of the Object Management Group. OMG™, Object Management Group™, CORBA logos™, OMG Interface Definition Language (IDL)™, The Architecture of Choice for a Changing World™, CORBAservices™, CORBAfacilities™, CORBAmed™, CORBAnet™, Integrate 2002™, Middleware That's Everywhere™, UML™, Unified Modeling Language™, The UML Cube logo™, MOF™, CWM™, The CWM Logo™, Model Driven Architecture™, Model Driven Architecture Logos™, MDA™, OMG Model Driven Architecture™, OMG MDA™ and the XMI Logo™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents & Specifications, Report a Bug/Issue.

Table of Contents

1 Scope

This specification addresses the need for a standardized naming schema for biological entities in the Life Sciences domains, the need for a service assigning unique identifiers complying with such naming schema, and the need for a resolving service that specifies how to retrieve the entities identified by such naming schema from repositories.

2 Conformance

The normative parts of this specification are:

- Platform independent model expressed in the attached XML file created according XMI format rules, v1.0.
- Platform specific model for Web Services using one of the proposed bindings (SOAP/HTTP, HTTP GET, FTP) for those who are implementing Web Services model.
- Platform specific model for Java for those who are implementing Java model.

If there is any inconsistency, or discrepancy between generality and specificity, between the platform independent and platform specific models, the platform specific model has precedence.

There are compliancy rules:

Comment: Issue 7385

- In order to be compliant with this specification an implementation must implement at least one platform specific model.
- An implementation in order to be compliant with this specification must contain LSID Resolution service, or LSID Discovery Resolution Service, or LSID Assigning Service, or any of their combinations.
- ~~In some places the specification allows, after giving a proper reasoning, more freedom. If an implementation follows such hints it still remains compliant.~~

Comment: Issue 7589

- Implementation of the Discovery Resolution Service is optional but if it is implemented, and if it uses ~~DDNS/DNS~~ DDDS/DNS for resolution, then it must follow the rules defined in this specification.

There are other, platform-specific, compliancy rules:

- Web Services based implementation needs to provide at least one binding to be compliant.
- The Web Services based PSM recommends as preferable returning data as SOAP “attachments.” The authors of this specification are aware, at the moment of writing, that not all SOAP toolkits (for various programming languages), fully support “attachments.” In those cases, the implementations may instead choose to use base64 encoding for the returned values, and still remain compliant with this specification.

The normative specifications of the platform specific models are expressed in the accompanied files in a document whose number is given in Annex A. Parts of these files may also appear in the explanatory text of this document. If they do and if there are some differences or discrepancies the contents of the accompanied files has precedence.

3 Normative References

NOTE: Needs to be eliminated.

4 Terms and Definitions

NOTE: Needs to be eliminated.

5 Symbols

List of symbols/abbreviations.

NOTE: Needs to be eliminated.

6 Additional Information

6.1 Relationship to Existing OMG Specifications

The specifications contained in this document require no changes to adopted OMG specifications.

- Gene Expression (formal/2003-02-03)
- OMG: Model Driven Architecture
- OMG: XML Interdata Interchange

6.2 Acknowledgements

The authors of this document wish to express their sincere appreciation to those listed below (in alphabetic order) for their invaluable contributions of ideas and experience. Ultimately, the conclusions expressed in this document are those of the authors and do not necessarily reflect the views or ideas of these individuals, nor does the inclusion of their names imply an endorsement of the final product.

- Jordi Albornoz, jordi@us.ibm.com
- Stefan Atev, satev@us.ibm.com
- Ray Lee, raylee@us.ibm.com
- Alister Lewis-Bowen, alister@us.ibm.com
- Chetan Murthy, chet@us.ibm.com
- Dennis Quandennisq@us.ibm.com
- Ugis Sarkans, ugis@ebi.ac.uk

- Ben Szekely, bhszekel@us.ibm.com
- Alyssa Wolf, alyssaw@us.ibm.com

There were also people involved in the evaluation of our submissions. They did not only evaluate the proposed specification but also contributed with suggestions and ideas. The authors would like to thank specifically:

- Richard Scott, Richard.Scott@denovopharma.com
- Mike Dickson, mike.dickson@earthlink.net
- Loralyn Mears, loralyn.mears@sun.com

The chapter describing LSID Syntax is adapted from the I3C proposal to this same RFP. Here are the contributors as indicated in the I3C document:

- Philip Werner, Avaki Corporation, pwerner@avaki.com
- Ted Liefeld, Millennium Pharmaceuticals, Inc., Ted.Liefeld@mpi.com
- Brian Gilman, MIT/Whitehead Institute, gilmanb@genome.wi.mit.edu
- Stephanos Bacon, Avaki Corporation, sbacon@avaki.com
- Josh Apgar, Avaki Corporation, japgar@avaki.com

7 Introduction

In the life sciences there are many different types of entities that are manipulated analytically. Typically, each type of entity has at least one "identifier" attribute that is used for identification, e.g. a GenBank accession number. The proposed specification fills the gap where there is no universally adopted system for assigning and recognizing identifiers in the life sciences domain. The issue is particularly important for the analysis and processing of data from high-throughput techniques, such as DNA microarrays, protein arrays and mass spectrometry.

An example (use case):

The recently approved Gene Expression specification (formal/2003-02-03) brings standardization to microarray gene expression experiments through the definition of the MAGE-ML XML format. One of the key design principles of MAGE is that an identifier can reference entities, such as Arrays, ArrayDesigns, Sequences, Hybridizations, and Compounds. This allows the pieces of a gene expression experiment to be defined in separate files or data sources, and then cross-referenced in other files or data sources.

To fully utilize this identifier concept, there must be agreement on the syntax of the identifier, so that software can recognize & use the constituent parts. For many identifiable entities in MAGE, there are already well-known ad-hoc naming sources that can be used, e.g. GenBank, ATCC, NCBI Taxonomy. In an effort to address the identifier problem, a request was made for proposals defining a set of interfaces, data structures and services which will enable the unique naming & resolution of life sciences entities in a systematic way.

This specification hopes to address the issues described above.

A UML diagram can summarize the whole response:

Comment: Issue 7386

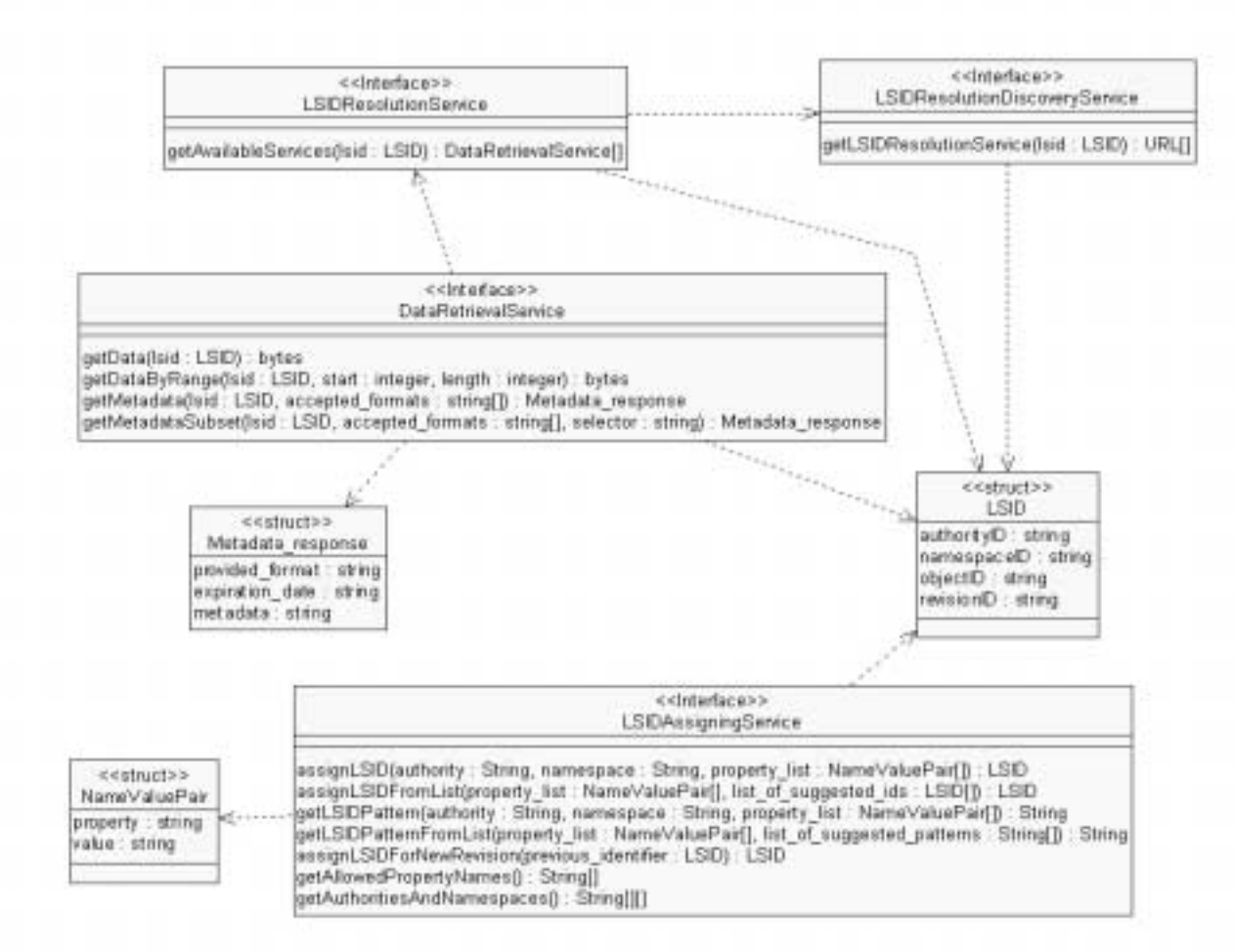

Figure 1 - A UML Diagram

8 LSID Syntax

Life Science Identifiers (LSIDs) are persistent, location-independent, resource identifiers for uniquely naming biologically significant resources including but not limited to individual genes or proteins, or data objects that encode information about them.

LSIDs are intended to be semantically opaque, in that the LSID assigned to a resource should not be counted on to describe the characteristics or attributes of the resource that the LSID refers to. The users of the LSIDs are permitted to use individual components (as specified elsewhere in this document) of LSIDs - although the LSID component parts themselves should be treated as opaque pieces of the identifier.

LSIDs are expressed as a URN namespace [1] and share the following functional capabilities of URNs [2]:

- Global scope: A LSID is a name with global scope that does not imply a location. It has the same meaning everywhere.
- Global uniqueness: The same LSID will never be assigned to two different objects.
- Persistence: It is intended that the lifetime of an LSID be permanent. That is, the LSID will be globally unique forever, and may be used as a reference to an object well beyond the lifetime of the object it identifies or of any naming authority involved in the assignment of its name.
- Scalability: LSIDs can be assigned to any data element that might conceivably be available on the network, for hundreds of years.
- Legacy Support: The LSID naming scheme must permit the support of existing legacy naming systems, insofar as they meet the requirements specified below.
- Extensibility: Any scheme for LSIDs must permit future extensions to the scheme.
- Independence: It is solely the responsibility of a name issuing authority to determine conditions under which it will issue a name.
- Resolution: A URN will not impede resolution (translation to a URL).

This namespace specification is for a formal namespace.

8.1 Specification

Comment: Issue 7565

The LSID declaration consists of the following parts, separated by ~~double~~ single colons:

- "URN"
- "LSID"
- authority identification
- namespace identification
- object identification
- optionally: revision identification. If revision field is omitted then the trailing colon is also omitted.

The authority identification is usually an Internet domain name. In this case it is recommended that it be owned by the organization that assigns an LSID in question. Such organization is responsible for ensuring the uniqueness of the string created from the namespace, object and revision identifications. In the case where the authority identification string is not an Internet domain name, the authority should take care to ensure that it is a unique string and if possible, register that unique string with the organization that is currently the authority for the URN Namespace Identifier (NID) "lsid" (See section ~~8.3~~ 13.3).

The namespace identification is an alphanumeric sequence that constrains the scope in which the subsequent object identification is resolved.

The object identification is an alphanumeric sequence.

The revision identification is an alphanumeric sequence. It is an optional component of the LSID.

Examples:

URN:LSID:ebi.ac.uk:SWISS-PROT.accession:P34355:3

URN:LSID:rcsb.org:PDB:1D4X:22

URN:LSID:ncbi.nlm.nih.gov:GenBank.accession:NT_001063:2

LSIDs must be assigned to at most one resource, and are never reassigned.

It is intended that the lifetime of an LSID be permanent. That is, the LSID will be globally unique forever, and may be used as a reference to an object well beyond the lifetime of the object it identifies.

There is no requirement that a biologically significant object have only one LSID. An existence of such situations is, however, discouraged.

An LSID usually represents a piece of data, but it is allowed to have LSIDs representing an abstract entities or concepts. If an LSID represents real data, the LSID Resolution service (described elsewhere in this document) must resolve always the same set of bytes representing such data. If an LSID represents an abstract entity the LSID resolution service must always resolve an empty result.

8.1.1 Case sensitivity

Life Sciences Identifiers are case-insensitive for the first three portions of the identifier ("URN", "LSID", authority identification). The remainder of the identifier (namespace identification, object identification, revision identification) is case sensitive. This allows LSIDs to be more compatible with RDF URI syntax, and also provides simpler mapping onto existing URLs, which are also case sensitive for their path and query string portions.

8.1.2 Comparing LSIDs

LSIDs are lexically equivalent if the authority, namespace, object, and revision identifications are all identical, the authority in the case insensitive, and the rest in the case-sensitive comparison.

9 LSID Resolution Service

The LSID Resolution service provides access to data entities identified by LSIDs. It can facilitate the retrieval of a data entity, or it can retrieve additional data related to the data entities (described below as metadata). Note that an LSID names one data entity only. While metadata may be associated with the data object, entity or abstract concept named by the LSID, metadata is not the data named by the LSID for the purposes of resolution or naming.

The whole resolution process in a nutshell consists of these steps:

1. A client holds an LSID and would like to retrieve data named by this LSID or metadata associated with the entity represented by this LSID.
2. The client either knows about an LSID Resolution service appropriate to this LSID, or it can use LSID Resolution Discovery service (described in the next chapter). As a result, the client knows how to call methods on a particular LSID Resolution service.
3. The client sends a request to the LSID resolution service using method getAvailableServices supplying the entire LSID as a parameter. The service returns locations, protocols, and, if necessary, additional information of one or more services for retrieving data and metadata.
4. The client picks one or more returned services and sends them requests using their methods getData and/or getMetadata.

Comment: Issue 7388

In general, the Life Sciences Identifier can be deployed upon existing databases without the need to modify the databases directly, especially if an appropriately flexible metadata format is chosen. In such architecture, the Life Sciences Identifier authority is serving as a translator between the legacy database retrieval service and the Life Sciences Identifier resolution service. The ‘namespace identification’ and ‘object identification’ parts of the Life Sciences Identifier can be used to hold the locator to the data within the existing database. Often the unique database key is used in the “object identification” part of the Life Sciences Identifier although another key, perhaps in an extra column or a combination of columns might be used as the “object identification’ part, if that proves more appropriate to the implementer. The Life Sciences Identifier authority can extract the locator from a Life Sciences Identifier presented for resolution and can use the locator to retrieve the data from the existing database.

It should be carefully noted that it is essential that once a Life Sciences Identifier has been issued for any particular set of bytes, those bytes should never change. For this reason implementers are urged to take care when mapping databases that apply versioning or simply updating of an existing record by overwriting and should make arrangements to never violate this important rule and use or update the ‘revision identification’ part where appropriate.

The full interface of the LSID Resolution service consists of the following methods:

services getAvailableServices (LSID lsid)

getAvailableServices (LSID lsid)

lsid is a string formatted as described in the "LSID Syntax chapter" above.

A fault is returned if the LSID Resolution service does not know anything about the given **lsid**.

The method returns a list in which each element represents a data retrieval service. It must contain a location of the service and a protocol how to access the service. The data retrieval services implement the following methods:

bytes getData (LSID lsid)
bytes getDataByRange (LSID lsid, integer start, integer length)
Metadata_response getMetadata (LSID lsid, string[] accepted_formats)
Metadata_response getMetadataSubset (LSID lsid,
string[] accepted_formats, string selector)

The data retrieval services may implement all of the methods, or only methods for retrieving data, or only methods for retrieving associated metadata.

The same LSID named data object must be resolved always to the same set of bytes. Therefore, all of the data retrieval services return the same results for the same LSID. The user has, however, the choice of which one of these to utilize depending on its location, known quality of service and other attributes.

With metadata, the situation is different. Each data retrieval service can provide different metadata for the same LSID.

bytes getData (LSID lsid)

This method is used to return data associated with the given lsid. If a copy of the data represented by an LSID cannot be returned for any reason, an exception should be raised.

If the given **lsid** represents an abstract entity (a concept), this method returns an empty array of bytes.

Note that the semantics of the returned bytes is not defined by this specification. It is either known from an external documentation, or (preferably) it is available by reading the metadata for this particular **lsid**.

bytes getDataByRange (LSID lsid, integer start, integer length)

This method is used to return data associated with the given LSID in data chunks. The caller selects the size of chunks. start indicates a starting position (in bytes, starting with zero), length gives maximal size (in bytes) of the transfer. The end of transfer is indicated when the returned number of bytes is less than was requested.

An exception is raised if the starting position is outside of the size of the data resource. No exception is, however, raised if the call requests data that has been already sent.

Metadata_response getMetadata (LSID lsid, string[] accepted_formats)

This method is used to return document containing the metadata associated with a particular **lsid** at this particular data retrieval service. Note that this means that calling getMetadata on two different data services may yield different metadata since each service may contain different metadata about the same **lsid**.

Metadata can be returned in multiple formats. The **accepted_formats** argument is an array of strings, each of them contains a media type. The media type can include two types of wildcard entries:

- */*, and
- xxx/*,

where xxx is a media content type such as 'application' or 'text' as defined in the IANA media type registry [6]. The **accepted_formats** list shall be evaluated in the order of its elements. The first match of requested media type to available media type must be the format that is returned (see below structure **Metadata_response**). Thus if the first entry is the wildcard */* then the remaining entries will never be considered.

At the time of writing, there are no formally standardized media types for the more popular metadata formats. Specifically, RDF and XMI do not have media types registered with IANA. The W3C has a document [7] of the status "Internet-draft-to-be" of a media type for RDF. XMI has had discussions about a media type but have decided that it is too early to submit to IANA if they will at all. Therefore, this specification suggests using the following types until the formally registered media types appear:

- x-application/rdf+xml (for metadata in RDF format)
- x-application/xmi+xml (for metadata in XMI format)

Having explicitly shown these two media types does not mean that this specification expects metadata only in these two formats.

```
Metadata_response {
    string              provided_format;
    Timestamp           expiration_date;
    Metadata_document   metadata;
}
```

The **Metadata_response** data structure is returned by getMetadata method. If the data retrieval service does not have metadata in one of the formats requested by the **accepted_formats** input argument, then an exception should be raised.

The **provided_format** is given as a media type just as the ones used for the **accepted_formats** input argument. Wildcards are not allowed in the return, however. It is also not a list but a single media type specifying the type of the metadata that is being returned.

Comment: Issue 7386

The **expiration_date** specifies how long the metadata is expected to be valid. The Timestamp is a string encoded as defined in a W3C NOTE "Date and Time Formats" [8] (unless the platform specific model defines otherwise).

This NOTE defines a profile of ISO8601 standard [9]. ISO8601 describes a large number of date/time formats and the NOTE reduces the scope and restricts the supported formats to a small number. The profile offers a number of options from which this specification permits the following one:

YYYY-MM-DD (e.g., 2000-12-31)

The **Metadata_document** is (usually) a string containing the metadata itself. It is considered out of the scope of this specification to restrict the number of formats that the metadata can be returned in. The most popular and expected formats are, however, RDF and XMI.

This is a small example of metadata returned by **getMetadata** in RDF:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description rdf:about="urn:lsid:ensembl.org:homosapiens_gene:ensg00000002016">
        <pred:type xmlns:pred="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
rdf:resource="urn:lsid:ensembl.org:types:gene"/>
    </rdf:Description>
    <rdf:Description rdf:about="urn:lsid:ensembl.org:homosapiens_gene:ensg00000002016">
        <pred:storedas xmlns:pred="urn:lsid:i3c.org:predicates:"
rdf:resource="urn:lsid:ensembl.org:homosapiens_gene:ensg00000002016-fasta"/>
    </rdf:Description>
    <rdf:Description rdf:about="urn:lsid:ensembl.org:homosapiens_gene:ensg00000002016-fasta">
```

```
        <pred:format xmlns:pred="http://purl.org/dc/elements/1.1/" rdf:resource="urn:lsid:i3c.org:formats:fasta"/>
    </rdf:Description>
</rdf:RDF>
```

The example gives some properties of LSID "urn:lsid:ensembl.org:homosapiens_gene:ensg00000002016". Its "type" is "gene", and it is "stored as" "fasta" format in the (another) LSID "urn:lsid:ensembl.org:homosapiens_gene:ensg00000002016-fasta".

Metadata_response getMetadataSubset (LSID lsid, string[] accepted_formats, string selector)

This method is used to return document containing only a subset of metadata attached to the given lsid.

The method is very similar to the getMetadata method above. The semantics as to the **lsid** and **accepted_formats** input parameters are exactly the same. Also, the response information is returned in the same data structure.

The difference is that the third input argument, the selector, can be used to limit the amount of the metadata that is returned. The contents of this string and how it is used to limit the metadata returned is out of the scope of this specification. The selector string might, for example, include an XPath query string to yield only particular nodes of XML metadata. Or the string might be an SQL query upon the metadata store.

It is important to note that getMetadataSubset is not intended to be a general query of all metadata about an LSID (the same applies for getMetadata method). The returned metadata is only the metadata that the particular data retrieval service knows about. Other data services may know different metadata about the same LSID.

10 LSID Resolution Discovery Service

Depending on implementation and not being specified here, an LSID resolution services can be discovered from the LSID that belongs to this service. But finding the service location and how to access such resolution service from the LSID may be a tedious task for clients. The access may also differ for different LSIDs. Therefore, a special discovery service can be used.

LSIDResolutionService[] getLSIDResolutionService (LSID lsid)

The LSID Resolution Discovery service has only one method that takes any LSID and returns a list of LSID Resolution services that are responsible for the given LSID. The method may raise an exception if it fails to find an appropriate LSID Resolution service for the given LSID. This may occur particularly when the authority identification field of the LSID is less standardized (which may occur for local services known only to a limited number of clients).

It is conceivable (and may frequently prove the case) that the discovery service may be implemented using different middleware technology to that used to implement the LSID Resolution service. In the simplest case, the discovery service may be provided as a library in one or more languages.

Comment: Issue 7563

Section ~~8.3~~ 13.3 details an example of one such implementation using DDDS/DNS.

11 LSID Assigning Service

The LSID Assigning service is responsible for creation of LSIDs for given data entities. For example, it is provided by database storage for assigning LSIDs for submitted data entries.

This service is the only place that can deal with only parts of the LSID components. Otherwise, as stated in the "LSID Syntax" chapter, the LSID components (such as authority, namespace and object identifier) are intended to be semantically opaque.

The service consists of the following methods:

LSID assignLSID (String authority, String namespace, property_list)
LSID assignLSIDFromList (property_list, LSID[] list_of_suggested_ids)
String getLSIDPattern
(String authority, String namespace, property_list)
String getLSIDPatternFromList
(property_list, String[] list_of_suggested_patterns)
LSID assignLSIDForNewRevision (LSID previous_identifier)
String [ ] getAllowedPropertyNames()
String [ ] [ ] getAuthoritiesAndNamespaces()

LSID assignLSID(String authority, String namespace, property_list)

It returns a full LSID for a data entity that has the properties passed in the property_list (like an object type and the attributes belonging to the data entity). This LSID has authority and namespace as requested by caller. Service returns null if it cannot or does not want to name the object. property_list is an array of name/value pairs (both of type string).

An example:

assignLSID ("ebi.ac.uk", "ArrayExpress", [objectType = "Experiment",organization = "Sanger Institute"])

LSID assignLSIDFromList (property_list, LSID[] list_of_suggested_ids)

It is similar to the assignLSID, only the caller is suggesting the identifier itself. "authority" and "namespace" parts for all suggestions should be the same. The service can return one LSID from the list, something different (but with the same authority and namespace) or raise an exception.

An example:

assignLSIDFromList ([objectType = "Experiment",organization = Sanger Institute"],
["URN:LSID:ebi.ac.uk:ArrayExpress:SNGR-Exper-7",
"URN:LSID:ebi.ac.uk:ArrayExpress:Sanger-Exper-7"])

Comment: Issue 7389

~~Both examples above could return:~~

Both the example invocation of assignLSID and the example invocation of assignLSIDFromList above could return:

"URN:LSID:ebi.ac.uk:ArrayExpress:SNGR-Exper-7".

String getLSIDPattern(String authority, String namespace, property_list)

It returns a prefix of an LSID such that the caller can use that as a template for constructing LSIDs and it is still guaranteed that these will be globally unique as far as the caller takes care not to reuse the same LSID twice locally.

An example:

getLSIDPattern ("ebi.ac.uk", "ArrayExpress",
[objectType = "Feature",organization = "Sanger Institute"])

String getLSIDPatternFromList (property_list, list_of_suggested_patterns)

It combines assignLSIDFromList and getLSIDPattern.

An example:

getLSIDPatternFromList ([objectType = "Feature", organization = "Sanger Institute"],
["URN:LSID:ebi.ac.uk:ArrayExpress:SNGR-Feature-",
"URN:LSID:ebi.ac.uk:ArrayExpress:Sanger-F-"])

Comment: Issue 7389

~~Both examples above could return:~~

Both the example invocation of getLSIDPattern and the example invocation of getLSIDPatternFromList above could return:

"URN:LSID:ebi.ac.uk:ArrayExpress:SNGR-Feature-".

LSID assignLSIDForNewRevision (LSID previous_identifier)

The caller sends in an identifier for an existing object and expects a new identifier with a different revision (for naming a new version of some object).

An example:

assignLSIDForNewRevision ("URN:LSID:ebi.ac.uk:ArrayExpress:SNGR-Exper-7")

It could return "URN:LSID:ebi.ac.uk:ArrayExpress:SNGR-Exper-7:2".

String[] getAllowedPropertyNames()

It returns an array of names of properties that the LSID Assigning Service can use when passed in methods assignLSID, assignLSIDFromList, and getLSIDPattern.

For example, this method could return ["objectType","organization"].

String[][] getAuthoritiesAndNamespaces()

It returns an array of pairs of Strings where the first item is the authority part that the service can assign names for, and the second String is valid namespace in that authority.

For example, this method could return

[["ebi.ac.uk", "ArrayExpress"],

["ebi.ac.uk", "anotherDatabase"],

["anotherInstitute", "ArrayExpress"]].

12 Error Codes

Methods of the services described in the previous chapters may raise exceptions in order to stipulate that a fault state occurred. This chapter lists all possible error codes that the implementation should use in such situations. The codes will be used in the platform-specific way, as defined in the platform-specific sections later in this document. The names (in the table below) are only explanatory and may be used in the platform-specific models in the language-specific manner.

Code	Name	Description
Error codes dealing with LSID parameter		
200	MALFORMED_LSID	A syntactically invalid LSID provided.
201	UNKNOWN_LSID	An unknown LSID provided.
202	CANNOT_ASSIGN_LSID	No LSID can be created from the given properties.
Error codes dealing with data retrieval		
300	NO_DATA_AVAILABLE	No data exists for the given LSID. An exception with this code is raised when there could be data attached to the given LSID but they are not available in the time of the request. The exception should not be raised when the LSID identifies an abstract concept in which case there are never any concrete data attached to it.
301	INVALID_RANGE	The requested starting position of data range is not valid.
Error codes dealing with metadata retrieval		
400	NO_METADATA_AVAILABLE	No metadata exists for the given LSID - at the moment. The same data retrieval service may be successful next time. The exception should not be raised if there are no metadata at all, at any time. In such case, the appropriate method simply returns an empty set of metadata.
401	NO_METADATA_AVAILABLE_FOR_FORMATS	No metadata exists (this time, or any time) for the requested format. The exception cannot be raised if the requested format includes wild-chars.
402	UNKNOWN_SELECTOR_FORMAT	The format of the metadata selector is not supported by the service.
General error codes		
500	INTERNAL_PROCESSING_ERROR	A generic catch-all for errors not specifically mentioned elsewhere in this list.

501	METHOD_NOT_IMPLEMENTED	A requested method is not implemented. Note that the method in question must exist (otherwise it may be caught already by the underlying protocol and reported differently) - but it has no implementation.

The implementation may extend the set of error codes in order to include implementation-specific codes. If it does so it should use numbers above 20 in each of the groups, or any number above 700. In other words, the free codes are: 221-299, 321-399, 421-499, 521-599, 701-above).

13 Platform Specific Models

The previous chapters define a platform independent model of services related to the Life Sciences Identifiers. The real implementations, however, are expected to live in a more specific environment. This chapter shows two middleware specific models, one for Java, one for Web Services. Both models were derived from the platform independent model manually.

13.1 Java

The platform specific model for Java is expressed in a set of Java interfaces and as a class defining an exception. All the interfaces, and all the methods and error codes in **LSIDEXception** are normative. The shown implementation of the **LSIDException**, however, is not normative.

In all cases the **LSIDException** shall be raised either as a result of an exceptional condition as described in the platform independent model, or of a Java specific exceptional condition (such as an inability to read an input stream). The implementations are encouraged to define their own, more specific and more expressive exceptions inherited from the **LSIDException**.

The package name **org.omg.lsid** is assumed in all definitions.

LSID Resolution Service

The basic object representing an LSID is defined by an interface **LSID**:

```
package org.omg.lsid;
public interface LSID {
        String getLSID();
        String getAuthority();
        String getNamespace();
        String getObjectId();
        String getRevision();
}
```

The method **getLSID** returns the whole **LSID** as a string, the other methods return individual components of the same LSID.

The interface **LSIDAuthority** defines how to get all data retrieval services:

```
package org.omg.lsid;
public interface LSIDAuthority {
        LSIDResolutionService[] getAvailableServices (LSID lsid)
            throws org.omg.lsid.LSIDException;
}
```

The only method **getAvailableServices** returns a list of all available data (and metadata) retrieval services. Their functionality is expressed by an empty interface **LSIDResolutionService** that is further specialized by interfaces **LSIDDataService** (implemented by those data retrieval services that provide data) and **LSIDMetadataService** (implemented by those providing metadata).

```
package org.omg.lsid;
public interface LSIDResolutionService { }

package org.omg.lsid;
import java.io.InputStream;
public interface LSIDDataService extends LSIDResolutionService {
        InputStream getData (LSID lsid)
            throws org.omg.lsid.LSIDException;
        bytes[] getDataByRange (LSID lsid, int start, int length)
            throws org.omg.lsid.LSIDException;
}

package org.omg.lsid;
public interface LSIDMetadataService extends LSIDResolutionService {
        MetadataResponse getMetadata (LSID lsid, String[] acceptedFormats)
            throws org.omg.lsid.LSIDException;
        MetadataResponse getMetadataSubset (LSID lsid, String[] acceptedFormats,
                                String selector)
            throws org.omg.lsid.LSIDException;
}

package org.omg.lsid;
import java.io.InputStream;import java.util.Date;public interface MetadataResponse {
        String getFormat();
        Date getExpirationDate();
        InputStream getMetadata()
            throws LSIDException;
}
```

LSID Resolution Discovery Service

The only method **getLSIDResolutionService** returns a list of the **LSIDAuthority** instances:

```
package org.omg.lsid;
public interface LSIDResolutionDiscovery {
        LSIDAuthority[] getLSIDResolutionService (LSID lsid)
            throws org.omg.lsid.LSIDException;
}
```

LSID Assigning Service

The LSID Assigning Service follows closely the platform independent model. The exceptions are raised if a method cannot assign an LSID with the given authority and/or namespace, or if the provided properties are not sufficient.

```
package org.omg.lsid;
import java.util.Properties;
public interface LSIDAssigningService {
        LSID assignLSID (String authority, String namespace, Properties property_list)
            throws LSIDException;
        LSID assignLSIDFromList (Properties property_list, LSID[] list_of_suggested_ids)
            throws LSIDException;
```

```
        String getLSIDPattern (String authority, String namespace, Properties property_list)
            throws LSIDException;
        String getLSIDPatternFromList (Properties property_list, String[] list_of_suggested_patterns)
            throws LSIDException;
        LSID assignLSIDForNewRevision (LSID previous_identifier)
            throws LSIDException;
        String[] getAllowedPropertyNames();
        String[][] getAuthoritiesAndNamespaces();
}
```

13.2 Web Services

The platform specific model for Web Services derived its architecture according to the Web Service definition as suggested by the W3C document “Web Services Glossary” [10]:

> *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL).*

Comment: Issue 6957

~~The WSDL definitions adapted by this specification are designed to be compliant with the WS-I Basic Profile 1.0a [11] and the previews of upcoming Basic Profile 1.1 [12] with support for binary attachments descriptions in WSDL.~~

Comment: Issue 7571

The WSDL definitions adapted by this specification are designed to be compliant with the WS-I Basic Profile 1.0a [11] and the WS-I Attachments Profile 1.0 Board Approval Draft [12].

The platform specific model for Web Services is defined as a set of port types, and a set of three bindings. The bindings express the underlying protocol used to access Web Services. The bindings are defined for:

- SOAP over HTTP
- HTTP GET
- FTP

Some bindings are not provided for all port types (but each port type has at least one binding).

The port types and bindings are defined in the following accompanied files (all of them are normative):

File name (with port type definitions)	***(File contents)***	**File name (with binding definitions)**
LSIDPortTypes.wsdl	LSIDAuthorityServicePortType	LSIDAuthorityServiceSOAPBindings.wsdl LSIDAuthorityServiceHTTPBindings.wsdl
	LSIDDataServicePortType LSIDMetadataServicePortType LSIDMetadataSubsetServicePortType	LSIDDataServiceSOAPBindings.wsdl LSIDDataServiceHTTPBindings.wsdl LSIDDataServiceFTPBindings.wsdl
LSIDResolutionDiscoveryPortType.wsdl	LSIDResolutionDiscoveryPortType	LSIDResolutionDiscoverySOAPBinding.wsdl

LSIDAssigningPortType.wsdl	LSIDAssigningPortType	LSIDAssigningSOAPBinding.wsdl

13.2.1 Port types

WSDL has a concept known as a “portType.” A portType is analogous to a CORBA interface or even a Java interface. It is a set of method signatures describing the types of the input and output parameters.

LSID Resolution Service port types

Because this platform specific interface is derived from the general model described earlier all definitions are valid also here with the following exceptions expressing the specificity of the Web Services platform:

Method **getAvailableServices** returns a WSDL document defining all data retrieval services including their locations. Note that some data retrieval services may not support metadata retrieval, some may not support data retrieval, and some may have additional methods. For that reason each data retrieval method has its own port type. That way, a data retrieval service provider can pick and choose which methods it will implement by implementing one or more of the interfaces.

Comment: Issue 7569

When a Life Science Identifier Resolution service provider prepares a WSDL response in which there are multiple bindings for metadata retrieval services the service provider must apply the following policy to these bindings:

- The WSDL must define one uniquely named service for each distinct metadata document.
- Metadata bindings must be grouped into these services such that all bindings within a service return the same metadata document.
- Metadata bindings must not be placed in a service which contains a metadata binding where the two bindings return different metadata documents.
- A client can then ensure that it has a complete set of metadata by accessing exactly one metadata port from each named service.
- In cases where there is ambiguity or uncertainty as to whether or not metadata retrieval services provide documents that are equivalent, the service provider must assume that they are not, and provide multiple named services as described above.

Methods **getMetadata** and **getMetadataSubset** map the input parameter **accepted_formats** (as defined earlier) into a single string **acceptedFormats**. The string contains an ordered, comma-separated list of media types. The usage of such arrangements (instead of an array of strings) is done purposefully to allow the methods to be mapped to other wire formats more easily (specifically to be mapped to HTTP without SOAP).

Note that although the expiration date is expressed as an XML basic data type **xsd:dateTime** its profile still should be restricted as defined earlier.

LSID Resolution Discovery Service port types

Method **getLSIDResolutionService** returns a list of URLs, each of them pointing to a WSDL document defining an LSID Resolution Service, including its location.

LSID Assigning Service port types

The WSDL follow closely the platform-independent model, including the usage of more complex data types.

13.2.2 Bindings

The bindings are separated from the port types (interfaces) described above because it makes it easier for those who build alternate bindings for the same port types using a different transport protocol (which is not part of this specification).

13.2.2.1 Bindings for the SOAP over HTTP

Comment: Issue 6957

The WSDL definitions for this binding are designed to be compliant with the WS-I Basic Profile 1.0a [11] and the WS-I Attachments Profile 1.0 Board Approval Draft [12].

Several WSDL files define the binding for the SOAP over HTTP implementation. This is the only binding that is defined for all port types.

Comment: Issue 7572

The binding uses the "mime" WSDL binding namespace in certain places. This is to signify that the data should be sent as part of a mime multipart encoded message. This means to make use of SOAP with attachments. The reason for this is that it is the optimal way to transmit XML documents and binary blobs of data. This prevents problems with character encodings which can occur when XML documents are embedded within each other. All the methods **getAvailableServices**, **getData**, **getMetadata**, and **getMetadataSubset** return data via SOAP "attachments". This is a preferable way – but the authors of this specification are aware that not all SOAP toolkits (for various programming languages), at the moment of writing, fully support "attachments". In those cases, the implementations may choose to use instead the "~~binary64~~" "base64" encoding for the returned values, and still remain compliant with this specification.

Note that implementations that choose to send base64 encoded data within the SOAP messages rather than using attachments will not be WS-I compliant. This is because the WSDLs explicitly say that attachments should be used and the WS-I profiles state that implementations that send messages that do not conform to the relevant WSDL are not WS-I compliant.

Reporting errors

According to the SOAP specification the errors are carried in a SOAP Fault element.

The **<faultstring>** should minimally contain a human readable error description. Here is an example with a fault response to **getAvailableServices** with an unknown LSID:

<faultstring>201 : Unknown lsid:urn:lsid:a.bad.lsid.i3c.org:bad:object</faultstring>

The **<detail>** should contain two elements[1], an **<errorcode>** carrying the error code, and a **<description>** with the free text. For example:

```
<detail>
   <errorcode>201</errorcode>
   <description>Unknown lsid:urn:lsid:a.bad.lsid.i3c.org:bad:object</description>
 </detail>
```

1. While the SOAP specification asks for an additional and fully qualified element that would include both <errorcode> and <description>, the WS-I specification does not. The WS-I is used here as a main reference in order to remove such ambiguities.

13.2.2.2 Bindings for HTTP GET

Comment: Issue 6957

The HTTP GET binding is not WS-I compliant and can be ignored by WS-I compliant systems.

For port types of the LSID Resolution Service, there is a binding for HTTP using GET method.

LSIDAuthorityServicePortType

This specification defines a single HTTP binding for **LSIDAuthorityServicePortType**. This binding uses **urlEndcoded** input, which specifies that the message part **lsid** is supplied as an **HTTP GET** parameter (callers must append this parameter to the URL). The operation in the WSDL file specifies **location="/authority/"** which indicates that **/authority/** must be appended (cleaning up any extra / at the join) to the location specified in the **port** element. For example:

```
<port name="HTTPPort" binding="ahb:LSIDAuthorityHTTPBinding">
        <http:address location="http://localhost:9080/" />
</port>
```

specifies that a WSDL can be retrieved at **http://localhost:9080/authority/?lsid=<somelsid>**

LSIDDataServicePortType

This specification defines both **urlEncoded** binding and a direct binding for retrieving data. Each of these bindings contains both **getData** and **getDataByRange**. The caller indicates which method it is calling by including or omitting range parameters. For the case of the direct binding, **getDataByRange** cannot be supported for obvious reasons. However, this binding is included in order to remain compliant with the port type.

For example:

```
<port name="HTTPData" binding="dhb:LSIDDataHTTPBinding">
        <http:address location="http://localhost:9080/authority/data"/>
</port>
```

indicates that method **getDataByRange** may be called at **http://localhost:9080/authority/data?lsid=<somelsid>&start=<startind>&length=<length>**

whereas

```
<port name="HTTPData" binding="dhb:LSIDDataHTTPBindingDirect">
  <http:address location="http://localhost:9080/authority/data/protein_234542.fasta"/>
</port>
```

indicates that only method **getData** may be called at **http://localhost:9080/authority/data/protein_234542.fasta**

LSIDMetadataServicePortType

This specification defines two bindings for retrieving metadata. The first **LSIDMetadataHTTPBinding** is similar to **LSIDAuthorityHTTPBinding** in that it uses **urlEncoded** for the parameters, however, the metadata binding does not specify any path information. For example:

```
<port name="HTTPMetadata" binding="dhb:LSIDMetadataHTTPBinding">
```

<http:address location="http://localhost:9080/authority/metadata"/>
</port>

specifies that metadata can be retrieved at **http://localhost:9080/authority/metadata?lsid=<somelsid>&acceptedFormats=application/xml%2Brdf**

(note that %2B is the hex escape code for '+').

The second binding, **LSIDMetadataHTTPBindingDirect**, defines no **urlEncoded** attribute. Callers must retrieve metadata at the exact URL specifed in the **location** in the port.

The first binding that uses GET parameters is designed for services that want to use HTTP as a proxy to a database or another service. The second is designed for services that already have the data for LSIDs available at existing URLs.

In the case of both HTTP bindings, the format of the metadata and the expiration date of the returned metadata is contained in standard HTTP headers **Content-Type** and **Expires**. Note that usage of these headers cannot be currently expressed in WSDL. Thus this is not explicitly stated in the bindings but those headers must be interpreted as such.

Reporting errors

For the HTTP bindings, the errors are reported analogously to usual error responses over HTTP. The error code is carried by a specific HTTP header **LSID-Error-Code**, and the error description is in the response body. For example:

HTTP/1.1 500 Internal Server Error
Server: WebSphere Application Server/5.0
LSID-Error-Code: 201
Content-Language: en-US

LSID Unknown: urn:lsid:a.bad.lsid:bad:object

13.2.2.3 Bindings for FTP

Comment: Issue 6957

The FTP binding is not WS-I compliant and can be ignored by WS-I compliant systems.

The FTP binding is not endorsed by the W3C in the Web Service protocols at this time. It is provided for convenience and backwards compatibility with existing Life Science data resources that are available as files using the File Transfer Protocol. The WSDL FTP extension specified here is not intended to serve as a more general purpose mechanism for describing FTP services beyond this specification. For the purposes of this specification, the LSID resolution client utilizing the FTP binding is expected to extract the server name and file path from a WSDL port that references the standard FTP binding described in this specification and access that file using the standard FTP protocol. In the absence of any other out of band information the file transfer will default to an anonymous login with a binary transfer mode.

Many public Life Sciences data sources already provide their data via File Transfer Protocol. As such, this specification provides standard FTP bindings for data and metadata services. Unfortunately, no standard WSDL port or binding formats exists for FTP so this specification has to invent its own.

Unlike SOAP and HTTP, FTP was not designed as a protocol on which to map arbitrary method calls. Nevertheless, this specification designs port and binding as much as possible in the spirit of existing WSDL constructs for other protocols.

There are two bindings implementing **LSIDDataServicePortType** and **LSIDMetadataPortType** respectively.

The **input** message parts are not mentioned in these bindings because FTP does not provide any constructs on which to bootstrap these parts. The data `output` parts are assigned to the output of the FTP operation (i.e., the bytes returned). The ports that use these bindings specify the server and file path at which to fetch the bytes.

For example:

```
<service name="LSIDService">
        <port name="ftpPort1" binding="dfb:LSIDDataFTPBinding">
            <ftp:location filepath="/pub/lsid/245gs.mmcif" server="lsid.org"/>
        </port>
        <port name="ftpPort2" binding="dfb:LSIDMetadataFTPBinding">
            <ftp:location filepath="/pub/lsid/245gs.rdf" server="lsid.org"/>
        </port>
</service>
```

tells the caller to fetch data at **ftp://lsid.org/pub/lsid/245gs.mmcif** and metadata at **ftp://lsid.org/pub/lsid/245gs.mmcif.rdf**.

There is no specific operation for **GetDataByRange**. However, the motivation for **GetDataByRange** was to enable clients to receive data in a chunked fashion. This goal is sufficiently achieved by any FTP client or FTP API that can handle a URL.

The metadata output format can be deduced by convention from the file name but the exact mapping is out of the scope of this specification. There is no equivalent for the metadata expiration time.

There is also nothing defined regarding error reporting.

13.3 Discovering an LSID Resolution Service using DDDS/DNS

This protocol-specific section consists of a set of rules allowing an interoperable method of discovering a Life Science Identifier Resolution Service from a given Life Science Identifier. It is a platform-specific representation of the Life Science Identifier Discovery Resolution Service described earlier.

Comment: Issue 7589

As the whole Life Science Identifier Discovery Resolution Service is optional, it is not mandatory to implement and use the rules described in this chapter. However, if an implementation chooses to discover an LSID Resolution Service using ~~DDNS/DNS~~ DDDS/DNS, then it must rigorously implement the rules described here in order to remain compliant with this specification.

Before resolution can be performed on any Life Science Identifier, the authority service that has knowledge of where that particular Life Science Identifier can be authoritatively resolved, must first be discovered. The IETF Request For Comments (RFC) archive [13] details a number of standards that were drafted for this purpose. In particular, a mapping of the “Dynamic Delegation Discovery System” (DDDS [14]) to the existing Domain Name System (DNS [15]) is used to provide the discovery service for LSID Resolution Services. DNS Service records also known as SRV [16] records are used to provide the final explicit networked location of the appropriate service.

The advantage of the DDDS scheme is that at least two and potentially more separate directory registries may eventually be used to store the lists of unique authorities for all Life Science Identifiers. In the first instance the authority portion of a Life Science Identifier can be a unique string registered with the holder of the Life Science Identifier URN namespace identifier (NID). As described earlier in this document, the Life Science Identifier NID is the string “lsid.” For example the holding organization for NID “lsid” might be the *lsidauthority.org*. If the authority string is not found in the registry

maintained by the *lsidauthority.org*, or if this registry is not found, the general public DNS registries should instead be consulted to determine if the string is a properly registered domain name. In either case the mechanism will provide an IP address and TCP/IP port for the networked whereabouts of the correct authority service for any Life Science Identifier that can be contacted to continue the resolution process.

This mechanism will allow organizations wishing to establish themselves as an authority for data named by a Life Science Identifier to choose to either add a single line text record to their own DNS server, similar to adding a host entry using their existing domain name, or to externally register their unique authority string with the central owner of the Life Science Identifier namespace. DNS server software is widely available and over the years has proven exceptionally stable and scalable. The DNS is by far the largest; most heavily trafficked, and most widely distributed registry system ever deployed.

Assuming that a NAPTR DNS [17] record for the Life Science Identifier URN NID “lsid” is established at IANA (urn.arpa) by *lsidauthority.org* the following method should be used for determining the TCP/IP address and port of the service capable of further resolving a particular Life Science Identifier.

The client application software should implement the Resolver Discovery Service (RDS) [18] algorithm described in the IETF RFC 3402 [20] such a manner that it can perform the steps described in IETF RFC 3403 Section 6.12 [21] by implementing the capability to handle DNS NAPTR records. The client application software’s *First Well Known Rule* is to extract the characters between the first and second colon of a given Life Science Identifier. For the Life Science Identifier URN, application of this rule results in the string 'lsid'. The client application also specifies that, in order to build a Database-valid Key, the string 'urn.arpa' should be appended to the result of the *First Well Known Rule*. The resulting string is 'lsid.urn.arpa'.

Next, the client queries the DNS for NAPTR records for the domain-name 'lsid.urn.arpa'. The result is a single record:

;;	**Order**	**pref**	**flags**	**service**	**regexp**	**replacement**
IN NAPTR	100	10	“”	“”	“”	lsid.lsidauthority.org

The DNS server that serves domain names for *lsidauthority.org* will carry an entry for a host named lsid.lsidauthority.org. This may either be a CNAME to itself or another DNS server. The DNS server acting as lsid.lsidauthority.org will contain the following NAPTR records:

;;	**order**	**pref**	**flags**	**service**	**regexp**	**replacement**
IN NAPTR	100	10	“s”	“lsid”	"!^urn:lsid:([^:]+):!\1.lsid.lsidauthority.org.!i"	.
IN NAPTR	200	20	“s”	“lsid”	"!^urn:lsid:([^:]+):!\1!i"	.

These records will be widely cached by the DNS system in the normal manner for efficiency. The client application then evaluates these rules in the order and manner defined by the algorithm described in the IETF’s RFC 3402. Since two replacement regular expression substitution rules are likely to be returned and both are terminal or “stop” rules, meaning that after they are applied, the normal DDDS process comes to an end and the remainder of the URN resolution protocol is as defined by the protocol associated with the LSID NID, described earlier in this document. The two rules are likely to be:

```
a."!^urn:lsid:([^:]+):!\1.lsid.lsidauthority.org.!i"

b."!^urn:lsid:([^:]+):!\1!i"
```

Processing these two rules will result in the following steps.

The first rule (a) applied to the entire Life Science Identifier URN string creates a DNS query key of <AuthorityID>.*lsid.lsidauthority.org*. The client application will query DNS for this hostname. In the case where a unique <AuthorityID> string has been registered with the *lsidauthority.org*, this query will find a match, and the DNS server at the *lsidauthority.org* will return the CNAME PTR [22] hostname record to the client for the DNS server where the appropriate SRV record will be found. The returned hostname will be the DNS server name of the issuing authority and the next step in the resolution process.

In the case that there is no match to the first DNS query, the second rule (b) is always applied to the entire Life Science Identifier URN string. The second rule allows the client application to default to using the public DNS directories for discovering the whereabouts of the service for that <AuthorityID>. This regular expression results in the <AuthorityID> portion of the Life Science Identifier URN string being set as next host name to query against DNS. This hostname will be the DNS server name of the Life Science Identifier issuing authority and the next step in the resolution process.

In the fallback case, where the client application is unable to use the DNS to resolve the string "lsid.urn.arpa" or is unable to contact or receives a "not found" error from any DNS server referred to by a rule contained in an NAPTR DNS, the second rule (b) should be applied by default. A copy of this rule should always be included in the client application software to provide for the failure of the RDS entirely. The returned hostname will be the DNS server name of the issuing authority and the next step in the resolution process.

The client software now possess the knowledge of which DNS server it should act against when it queries for the DNS SRV records to determine the hostname and TCP/IP service port for that Life Science Identifier authority. The IETF's RFC 2782 [23] defines DNS SRV records and how they are queried. Using the method described, the "_lsid._tcp" service is discovered. Below is an example of the SRV record in the authority's DNS server which indicates that host *lsidresolver.foo.com* has a resolution service running on TCP/IP port 8080:

_lsid._tcp SRV 1 0 8080 lsidresolver.foo.com.

Example:

In the Life Science Identifier URN *urn:lsid:foo.com:namespaceid:objectid:revision*, *foo.com* is the <AuthorityID>. In the case where no unique string has been registered, the first rule (a) will require querying of DNS for *foo.com.lsid.lsidauthority.org* and this query will fail since no match will be found in the *lsidauthority.org* DNS for this hostname. If the first rule fails, the second rule (b) is applied requiring the client software to use the <AuthorityID>. *foo.com* as its next DNS query. This query will succeed, provided *foo.com* is actually an Life Science Identifier issuing authority, and the client application will be provided with the hostname of the DNS server that has the appropriate SRV record pointing to the resolution service acting for *foo.com*. This would usually be the DNS server for *foo.com*.

In the case where a unique authority string has been registered with the *lsidauthority.org* and where the hostname created by the application of rule (a) on the entire Life Science Identifier string is found, a CNAME PTR hostname record is returned. This hostname will be the DNS server with a record pointing to the correct resolution service.

For example: In the Life Science Identifier URN *urn:lsid:foo.com:namespaceid:objectid:revision*, *foo.com* is the <AuthorityID>. However *foo.com* transferred the part of its business that issued this Life Science Identifier URN to another organization *bar.org*, but continued on to trade and control DNS under the name *foo.com* for email and web access. In this case *bar.org* would apply to the administrator of the *lsidauthority.org* DNS server to have an entry added for hostname *foo.com.lsid.lsidauthority.org* which would return a CNAME to the bar.org DNS server if queried. Of course, <AuthorityID> strings need bear no resemblance to a DNS domain name in order for the RDS to function.

A Accompanied Files

This part of the specification is a set of the accompanied files. You will find these files at this URL:

http://www.omg.org/cgi-bin/doc?dtc/04-08-03

Some of these files are normative and some of them contain examples and convenient images. If there is a discrepancy between the contents of the normative files and this document, then the files take precedence.

This is the list of the accompanied files (the normative files are in bold font):

- LSID.mdl - a Rational Rose file with a diagram of the platform independent model,
- **LSID-XMI-<version>.xml** - an XMI representation of the same model.
- **LSID*.wsdl** - files containing WSDL for the platform specific model for Web services using various bindings (SOAP over HTTP, HTTP, FTP).
- SampleLSID*.wsdl - a set of full WSDL definitions (the same as in the previous set of files) but with a non-existing service locations.
- **org/omg/lsid/*.java** - a set of Java interfaces for the platform specific Java model.
- LSID.[jpg,png] - diagrams exported from LSID.mdl providing better resolution than the same image included in this document.

B References

1. Moats, R., URN Syntax, RFC 2141, May 1997.
2. Sollins, K. and L. Masinter, Functional Requirements for Uniform Resource Names, RFC 1737. December 1994.
3. LSID Resolution (I3C), In preparation, September 2002.
4. Mockapetris, P., Domain names – Implementation and Specification, STD 13, RFC 1035, November 1987.
5. Gulbrandsen, A., Vixie, P., and Esibov, L. A DNS RR for specifying the location of services (DNS SRV), RFC 2782, February 2000.
6. MIME Media Types: http://www.iana.org/assignments/media-types
7. A media type for Resource Description Framework (RDF): http://www.w3.org/2001/03mr/rdf_mt
8. Date and Time Formats (W3C): http://www.w3.org/TR/NOTE-datetime
9. International Standard for representation of dates and times http://www.iso.ch/iso/en/prods-services/popstds/datesandtime.html
10. Web Services Glossary (W3C), W3C Working Draft, August 2003: http://www.w3.org/TR/ws-gloss/

Comment: Issue 7573

11. Basic Profile Version 1.0a (~~W3C~~ WS-I): http://www.ws-i.org/Profiles/Basic/2003-08/BasicProfile-1.0a.htm

Comment: Issue 7571

12. WS-I Attachments Profile 1.0 Board Approval Draft: http://ws-i.org/Profiles/AttachmentsProfile-1.0-2004-06-11.html.
13. The Internet Engineering Taskforce, Request For Comments: http://www.ietf.org/rfc.html
14. Mealling, M (2002) 'Dynamic Delegation Discovery System' (Parts One through Five), RFC3401, 3402, 3403, 3404, 3405 (URLs: http://www.ietf.org/rfc/rfc3401.txt, http://www.ietf.org/rfc/rfc3402.txt, http://www.ietf.org/rfc/rfc3403.txt, http://www.ietf.org/rfc/rfc3404.txt, http://www.ietf.org/rfc/rfc3405.txt
15. Salamon, A (2003) 'DNS Related RFCs' (URL: http://www.dns.net/dnsrd/rfc/)
16. Gulbrandson A, Vixie P, Esobov L (2000) ' A DNS RR for specifying the location of services (DNS SRV)' IETF RFC2782 (URL: http://www.ietf.org/rfc/rfc2782.txt)
17. M. Mealling, The Naming Authority Pointer (NAPTR) DNS Resource Record, http://www.ietf.org/rfc/rfc2915.txt
18. The registration policies for URIs are found in the IETF's RFC 2717 and the URN NID registration policies are found in the IETF's RFC 2611, http://www.ietf.org/rfc.html
19. K. Sollings, Architectural Principles of Uniform Resource Name Resolution, http://www.ietf.org/rfc/rfc2276.txt
20. M. Mealling, Dynamic Delegation Discovery System (DDDS), Part Two: The Algorithm, http://www.ietf.org/rfc/rfc3402.txt
21. M. Mealling, Dynamic Delegation Discovery System (DDDS),Part Three: The Domain Name System (DNS) Database, http://www.ietf.org/rfc/rfc3403.txt
22. R. Elz, Clarifications to the DNS Specification, http://www.ietf.org/rfc/rfc2181.txt
23. A. Gulbrandsen, A DNS RR for specifying the location of services (DNS SRV), http://www.ietf.org/rfc/rfc2782.txt