Date: July 2011

Retrieve, Locate, and Update Service (RLUS) Specification

Version 1.0.1

OMG Document Number:	formal/2011-07-02
Standard Document URL:	http://www.omg.org/spec/RLUS/1.0.1/
Associated File(s)*:	http://www.omg.org/spec/RLUS/20101201
	http://www.omg.org/spec/RLUS/20101201/RLUSGenericService.wsdl
	http://www.omg.org/spec/RLUS/20101201/RLUSOrderService.wsdl
	http://www.omg.org/spec/RLUS/20101201/RLUSPatientService.wsdl
	http://www.omg.org/spec/RLUS/20101201/CCD.xsd
	http://www.omg.org/spec/RLUS/20101201/CDA.xsd
	http://www.omg.org/spec/RLUS/20101201/HW_POCD_MT000040.xsd
	http://www.omg.org/spec/RLUS/20101201/RLUSExpression.xsd
	http://www.omg.org/spec/RLUS/20101201/RLUSTypes.xsd

* original file(s): dtc/2010-12-02 (machine-consumable files - updated)

Minor revision to version 1.0 due to updated machine consumable files. The specification has been updated as well.

OMG’s Issue Reporting Procedure

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, Report a Bug/Issue (http://www.omg.org/technology/agreement.htm).

Table of Contents

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG’s specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at *http://www.omg.org/*.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. A catalog of all OMG Specifications is available from the OMG website at:

http://www.omg.org/technology/documents/spec_catalog.htm

Specifications within the Catalog are organized by the following categories:

OMG Modeling Specifications

- UML
- MOF
- XMI
- CWM
- Profile specifications

OMG Middleware Specifications

- CORBA/IIOP
- IDL/Language Mappings
- Specialized CORBA specifications
- CORBA Component Model (CCM)

Platform Specific Model and Interface Specifications

- CORBAservices
- CORBAfacilities
- OMG Domain specifications
- OMG Embedded Intelligence specifications
- OMG Security specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult *http://www.iso.org*

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to *http://www.omg.org/technology/agreement.htm*.

1 RLUS Overview

The Retrieve, Locate, and Update Service (RLUS) provides a set of interfaces through which information systems can access and manage information. RLUS allows health data to be located, accessed and updated regardless of underlying data structures, security concerns, or delivery mechanisms. RLUS explicitly occupies the service space within an information processing environment. It is independent of but compatible with underlying structures, including local security implementations, data models, or delivery mechanisms. By separating and exposing those aspects of resources that facilitate inter-organization work flows in a service layer, this specification abstracts the problem of interoperability away from underlying systems. It is this abstraction and reconfiguration that allows interoperability and system durability independent of burdensome technology integration.

1.1 Scope

The Retrieve, Locate, and Update Service (RLUS) web service specification seeks to define, at a service level, appropriate interfaces to locate, retrieve, and update resources among and between healthcare organizations. It is not intended to replace existing systems or implementations, but to create an interface standard for a service-oriented layer to expose those healthcare assets and resources within an organization that are needed to meet business or medical needs.

1.2 Purpose

Services in general make individual systems and components less brittle and more durable (e.g., less subject to change) because they guarantee a minimum level of functionality internally and externally to an organization. By providing for functionality inherent in a location, retrieval, and update service, an essential component of interoperability is thus defined. Further, by explicitly creating this component as a service, the need to adhere to the system architecture being used by the organization is reduced to aspects implied in the service interface definition, allowing narrowly defined roles and functionality to be supported between organizations.

RLUS supports extended business cases that are common to healthcare, but which have either gained little support in local implementations or are implemented in ad hoc ways. These business cases generally involve multiple partners in the health team sharing data over long periods of time. RLUS provides an appropriate space to define and implement these business cases, because the overall effects on internal systems are minimized.

1.3 Relationship to HL7 RLUS SFM

The RLUS specification is the product of a collaborative effort between Health Level 7 (HL7) and the Object Management Group (OMG). Within HL7, members of the SOA Special Interest Group produced a functional specification document (known as the Service Functional Model – SFM). This was used as the basis for the requirements expressed in the RFP. Many of the RFP requirements explicitly referred to sections within the SFM, especially the definition of the capabilities required in Section 5 of the SFM.

Although this specification is directly in response to the issued RFP, readers and viewers would be advised to become familiar with the SFM as well.

The SFM is an HL7 document that can be found at the following location:

http://www.hl7.org/documentcenter/ballots/2006SEP/support/SUPPORT_POOL_V3_RLUS_R1_D1_2006SEP_2006122 01 12743.pdf

1.4 Relation to Existing OMG Specifications

Reference	Description	Relationship to RLUS
COAS OMG document: formal/ 01-04-06 OMG URL: http:// www.omg.org/ spec/COAS/1.0/	The OMG Clinical Observation Access Service (COAS) provides a mechanism that allows for the retrieval of [clinical] content via a set of established interfaces. Effectively, COAS provides a generic retrieval capability and is independent of information model.	RLUS is to supersede the COAS specification, among other issues, the absence of an information model and accepted industry constructs inhibited marketplace adoption of COAS. RLUS is explicitly requiring support for HL7 semantic content, while leaving open the possibility to support other information models. The computational aspects of COAS were analyzed and have been considered during the development of this specification.
PIDS OMG document: formal/ 01-04-04 OMG URL: http:// www.omg.org/ spec/PIDS/1.1	The OMG Person Identification Service (PIDS) provides the functional capability to create and manage patient identities, and is a CORBA IDL specification.	The RLUS specification does not address identity management as part of its functionality, and is intended to defer to identity services (such as would be provided by PIDS-compliant software) for these functions.Worth noting is that the PIDS specification is being revised to address shortcomings, with the "Identity Cross-Reference Service (IXS)" specification.

The HILS submission (OMG doc # health/01-02-01) although not an adopted specification was used to guide much of the thinking regarding the location capabilities of RLUS.

2 Conformance (Compliance)

RLUS Level L0: Compliance to this level requires an implementation of the RLUS Management And Query Interface, providing navigation, access, and update capabilities for healthcare-related data. This includes a full implementation the required operations on the RLUS Management And Query Interface, which meets the platform specific models described in Clause 7 as well as follows the guidelines for mapping a semantic signifier to an RLUS interface definition as described in sub clause 7.2. Implementations compliant to RLUS L0 are considered "Minimum RLUS Implementations."

RLUS Level L1: Compliance to this level requires full compliance to RLUS L0 (see above) and a full implementation of the RLUS Meta Data Interface, which provides dynamic meta data management capabilities related to the RLUS Managament And QueryInterface. Only RLUS L1 compliant implementations can claim to be an "RLUS Implementation" or "Full RLUS Implementation."

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl

Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000, http://www.w3.org/TR/SOAP

XML Schema Part 2: Datatypes, W3C Recommendation 02 May 2001, http://www.w3.org/TR/xmlschema-2

4 Terms and Definitions

There are no terms or definitions defined in this specification.

5 Symbols

There are no symbols/abbreviations.

6 Additional Information

6.1 Initial Implementations

An initial implementation of this interface has been built by Intel and is part of the Healthcare Development Kit offered by Intel SOA Expressway for Healthcare. More information on the solution can be found at http://www.intel.com/healthcare/ps/soa/.

This implementation provided the baseline for the machine readable files and provides a working implementation of RLUS for Patient Histories, Clinical Orders and Clinical Encounters using the HL7 CCD and CDA XML document standards to represent the semantic signifier. This is an implementation of the RLUS Management And Query Interface.

6.2 Platform Independent (PIM) and Platform Specific (PSM) Models

Per the OMG RLUS RFP, this specification contains the definition of a platform independent model that describes the interface without reference to a specific technical implementation. That platform independent description of RLUS is listed in Clauses 7, 8, and 13 of this specification. Also a platform specific model, which is an implementation of the interface for a SOAP, XML-based deployment for HL7 v.3 and HL7 v2.x is listed in Clauses 7, 8, 9, and 13. The descriptions of the platform specific model are normative implementations and are detailed in the supporting machine readable files that supplement this specification.

The text in sub clauses 7, 8, 9, and 13 refer to a specific file in the machine readable supplement that is relevant to that specific clause in this document.

6.3 The Platform Independent Model

The Platform Independent Model (PIM) for RLUS represents a wire and payload implementation independent means to describe the RLUS interfaces.

The PIM is detailed in the sub clauses below containing a UML model and associated descriptions that depicts the structure of the interfaces provided by RLUS. It is important to note that the structure of interfaces is specifically defined for the Service provider and reflects a natural structure of components that may provide the RLUS functionality. These are effectively orthogonal to the groupings of capabilities defined in Functional Profiles that form the basis for conformance and for consumption of the service by Clients.

RLUS embodies a number of key concepts that should be understood up front before attempting to read and understand this specification. A full Glossary is included later in the document, but key terms are described below.

Term	**Description**
Semantic Signifier	This is a schema definition and an associate set of validation instructions. It Semantic Signifier defines the structure and the means by which that structure should be validated. In XML for example, it would be an XSD and associated schematron.
Logical Record	This is an instance of the semantic signifier. It could either be in document or message format depending on the RLUS implementation.
Logical Record ID	This is a structure similar to the HL7 RIM II type. It is a series of pairs of alphanumeric identifiers. A pair of IDs represents the record identifier and the system identifier. There shall be one pair for each location / system in which the logical record is stored. The system identifier must be represented as a GUID to ensure global uniqueness across a federated deployment.

7 Platform Independent Model - RLUS Management and Query Interface

7.1 General

This is the minimum, necessary interface for RLUS normative compliance. It provides the core functions for reading, writing, and locating information with an RLUS-compliant service interface.

For example, Describe() is used to return the data structure for a semantic signifier, Get()/List()/Locate() are used to search for instances of that semantic signifier, and Put()/Discard() are used to write instances of that semantic signifier.

7.2 Get

This is an operation for retrieving a single logical record based on parameters supplied by the RLUS Search Struct that uniquely identify a single record.

Input Parameters	1. searchStruct - The structure that describes the filter criteria or query-by-example representation used to retrieve data. 2. logicalRecordID - The identifier for the logical record represented by the logicalRecordInstance paramter.
Output Parameters	1. RLUSStatusCode - Structure which contains the success or failure codes of the executed service operation. Sent as the return value of the operation. 2. logical Record Instance - This is the structure which represents the data the RLUS is retrieving (such as a CCR or CCD for a patient health record or a CDA document for an encounter visit summary in XML).
Error Codes	1. record not found 2. More than one record found. This is an error condition as the expected use of this operation is for a uniquely identified record search.
Additional Details	This operation is intended for highly optimized single record or document searches using a uniquely identifying attribute or set of attributes. Most often this will come in the form of a record identifier search, such as a patient, order or encounter record identifier.

7.3 List

This operation is for returning a list of logical record instances based on the contents of the RLUSSearchStruct in a manner consistent with Get() but is capable of streaming many records from the underlying source to the calling client.

Input Parameters	1. searchStruct - The structure that describes the filter criteria or query-by-example representation used to retrieve data. 2. maxResultStreams - The maximum number of “chunks” of data to be returned from List() or Locate(). If a value of 1 is provided, then all data shall be returned in one list through one operation call. A value of 0 shall result in the operation making an optimization decision to return as many “chunks” as required to effectively stream the result set to the client. Any other value than 0 or 1 represent the maximum number of data “chunks” the implementation can allocate to return the full result set. For example if the value is 3, then the implementation must return the result entire result set in no more than 3 calls to this service operation. 3. previousResultID - This is the id token provided from a previous call to List() or Locate() that provides a “cookie” for the streaming functions to work. Set to a value of -1 by the client to signify the initial call by the client and set to a value of 0 by the client to signify a cancellation of the processing request.
Output Parameters	1. RLUS StatusCode - Structure that contains the success or failure codes of the executed service operation. Sent as the return value of the operation. 2. logicalRecordInstances - Array of logical record instances. 3. resultID - The id token / “cookie” that identifies a streaming result set. 4. finishedFlag - Provides an indication as to whether or not a streaming retrieve function is done. A value of 0 means that all records have been retrieved. A value greater than one indicates approximately how many streaming iterations are left before the operation is complete.
Error Codes	1. No results found 2. Internal resource processing error 3. Processing request has timed out
Additional Details	This operation is intended for streaming multiple records or documents results from the underlying repository to the client system. Unlike Get() this is intended to pull back many records that match the parameters of the filter criteria. The use of maxResultStreams and the previousResultID is to support large result sets that might take multiple operation calls to retrieve the entire data set to minimize the latency between when the initial search request is sent and a first set of data is available for display. This is a feed forward operation only, meaning that previously returned resultID values will no longer be valid for returning data once a new resultID has been generated by the operation and returned to the caller. Also, this operation will timeout if the caller does not make the next call to List() while the finishedFlag parameter is still greater than 0. The length of time required for the timeout to occur is an implementation specific detail that varies depending on the specific semantic signifier data type the RLUS interface is processing and would be specified as documentation for the specific implementation. Resource cleanup as a result of a cancelled request by the caller (previousResultID=0) or request timeout is a responsibility of the underlying implementation.

7.4 Locate

This operation is for returning a list of RLUS service locations where the desired logical record can be found.

Input Parameters	1. searchStruct - The structure that describes the filter criteria or query-by-example representation used to retrieve data. 2. maxResultStreams - The maximum number of "chunks" of data to be returned from List() or Locate(). If a value of 1 is provided, then all data shall be returned in one list through one operation call. A value of 0 shall result in the operation making an optimization decision to return as many "chunks" as required to effectively stream the result set to the client. Any other value than 0 or 1 represent the maximum number of data "chunks" the implementation can allocate to return the full result set. For example if the value is 3, then the implementation must return the result entire result set in no more than 3 calls to this service operation. 3. previousResultID - This is the id token provided from a previous call to List() or Locate() that provides a "cookie" for the streaming functions to work. Set to a value of -1 by the client to signify the initial call by the client and set to a value of 0 by the client to signify a cancellation of the processing request.
Output Parameters	1. RLUSStatusCode - Structure that contains the success or failure codes of the executed service operation. Sent as the return value of the operation. 2. logicalRecordInstanceLocations - Array of logical record locations as specified by the RLUSLocationsResultSet described in Clause 8. 3. resultID - The id token / "cookie" that identifies a streaming result set. 4. finishedFlag - Provides an indication as to whether or not a streaming retrieve function is done. A value of 0 means that all records have been retrieved. A value greater than one indicates how many streaming iterations are left.
Error Codes	1. No results found 2. Internal resource processing error 3. Processing request has timed out
Additional Details	This method is intended for retrieving record keys associated to record instance locations rather than the document and/or record instances themselves. Useful in preparing indexes (say hyperlinks on web pages) or in navigating the locations of copies of a document on an information network. The information needed to generate the location of an RLUS logicalRecordInstance is provided via the Put() operations updateRequestSrcStruct parameter. Just like List(), this is a feed forward operation only, meaning that previously returned resultID values will no longer be valid for returning data once a new resultID has been generated by the operation and returned to the caller. Also, this operation will timeout if the caller does not make the next call to Locate() while the finishedFlag parameter is still greater than 0. The length of time required for the timeout to occur is an implementation specific detail that varies depending on the specific semantic signifier data type the RLUS interface is processing and would be specified as documentation for the specific implementation. Resource cleanup as a result of a cancelled request by the caller (previousResultID=0) or request timeout is a responsibility of the underlying implementation.

7.5 Put

This operation is for writing an instance of the logical record to the RLUS implementation. This includes insert / update (“upsert”) operations.

Input Parameters	1. writeCommand - Input structure that defines what action to take (UPSERT, INSERT-ONLY, or UPDATE-ONLY). Default value is UPSERT. 2. updateRequestSrcStruct - This is an input structure populated by the caller that provides application and source location context to the RLUS, which makes future calls to Locate() possible. 3. logicalRecordInstance - This is the input structure that represents the data the RLUS is writing (such as a CCR or CCD for a patient health record or a CDA document for an encounter visit summary). If this document or message instance does not conform to the schema definition described by the semantic signifier in the updateRequestSrcStruct parameter, then an invalid semantic signifier error will result.
Output Parameters	1. RLUSStatusCode - Structure that contains the success or failure codes of the executed service operation. Sent as the return value of the operation. 2. logicalRecordID - The identifier for the logical record represented by the logicalLRecordInstance parameter.
Error Codes	1. Invalid semantic signifier 2. Duplicate key 3. Record not found for update 4. Internal resource processing error
Additional Details	This is the primary operation for inserting and updating records. Through a parameter, the caller can specify whether to INSERT only (meaning fail if a duplicate record exists), UPDATE only (meaning fail if an existing record to update is not found), or UPSERT (meaning determine whether if an insert or update is the appropriate operation). The behavior for an UPSERT is to first determine if a matching record exists in the underlying repository and if so, then issue an update; but if not, then to insert a new record. The default behavior of this operation unless otherwise specified by the client caller in the write Command input parameter is UPSERT. It is the responsibility of the underlying implementation to ensure correct transaction behavior and data integrity as a result of the successful or failed execution of this operation.

7.6 Discard

This operation is for discarding (either physically or logically deleting records from the underlying source). A selection filter is created and all logical records that meet that filter criteria are discarded.

Input Parameters	1. searchStruct - The input structure that describes the filter criteria or query-by-example representation used to select what data to discard. 2. updateRequestSrcStruct - This is an input structure populated by the caller that provides application and source location context to the RLUS in order to process the discard request.
Output Parameters	1. RLUSStatusCode - Structure that contains the success or failure codes of the executed service operation. Sent as the return value of the operation.
Error Codes	1. Invalid semantic signifier 2. Internal resource processing error
Additional Details	This is the operation for deleting (logically or physically) from the underlying data store. Whether the operation is a logical or physical delete in the underlying data store is an implementation specific decision. However, due to the nature of healthcare data it is likely to be a logical deletion in most practical implementations.

7.7 Describe

This operation is used to output the detailed schema definition of the associated semantic signifier.

Input Parameters	1. semanticSignifierName - Human readable text string that corresponds to the name of the semantic signifier to process. If the RLUS implementation does not support the specified name, then the invalid semantic signifier error code is returned to the caller.
Output Parameters	1. RLUSStatusCode - Structure that contains the success or failure codes of the executed service operation. Sent as the return value of the operation. 2. semanticSignifier - The output structure that defines the properties of the semantic signifier the RLUS is processing.
Error Codes	1. Invalid semantic signifier 2. Internal resource processing error
Additional Details	This operation is used to return the implementation of the structure of the semantic signifier to a calling application. This will most often take the form of an XML schema (XSD), but is not necessarily required to be such a structure. The platform specific implementation will determine the specific expected structure of the resulting semanticSignifier parameter.

7.8 Initialize

This optional operation is used to send a record from an internal source system onto an RLUS network in response to record create, update, or delete events “inside-out” from a local system. That is, the local system is generating a logicalRecordInstance in response to system generated event (like SQL table insert) and the data generated is “initialized” in order to be shared across a network of RLUS service instances.

Input Parameters	1. writeCommand - Input structure that defines what action to take (UPSERT, INSERT-ONLY, or UPDATE-ONLY). Default value is UPSERT. 2. updateRequestSrcStruct - This is an input structure populated by the caller that provides application and source location context to the RLUS that makes future calls to Locate() possible. 3. logicalRecordInstance - This is the input structure that represents the data the RLUS is writing (such as a CCR or CCD for a patient health record or a CDA document for an encounter visit summary).
Output Parameters	1. RLUSStatusCode - Structure that contains the success or failure codes of the executed service operation. Sent as the return value of the operation. 2. logicalRecordID - The identifier for the logical record represented by the logicalLRecordInstance parameter.
Error Codes	1. Invalid semantic signifier 2. Duplicate key 3. Record not found for update 4. Internal resource processing error
Additional Details	This operation may be optionally implemented. It is a variation of the Put() operation and is utilized to handle specific interoperability scenarios. Specifically, when an RLUS information network is assembled, network participants add data to that network through system generated events that require different rule processing than might be required for the Put() operation. Examples might include incremental logging or incremental duplicate data checks.

8 Platform Independent Model - RLUS Data Structures

8.1 General

This is a high level description of the RLUS data structures. The detailed implementation of the normative description of the structures is in the machine readable files associated with this specification.

RLUS Data Structure Name	**Summary Description**
RLUSSemanticSignifier	Meta-data structure to definition of a data type processed by RLUS. Consists of the following elements: 1. Uniquely identified and human readable name of the semantic signifier that is used to "name the type"/ 2. Globally unique alphanumeric ID, represented as a GUID. 3. Version identifier. 4. Schema Definition (often listed as a URI to XSD in XML). Is implemented as a URI to XSD in the normative platform specific model. 5. Rendering Definition (often listed as a URI to XSL in XML). Is implemented as a URI to XSL in the normative platform specific model. For example, this is used to create an HTML viewable page from the logicalRecordInstance XML payload. 6. Integrity Ruleset Definition (listing to a URI to SCHEMATRON or WSDL reference in XML). Is implemented as a URI to SCHEMATRON or WSDL in the normative platform specific model.
RLUSWriteCommandEnum	Enumeration that codifies: • UPSERT • INSERT-ONLY • UPDATE-ONLY
RLUSSearchStruct	Wrapper structure to describe a "query-by-example" or filter criteria for an RLUS search. The structure consists of the following: 1. Name of the semantic signifier to search. 2. Flag indicating a query by example or filtered criteria search. 3. If QUERY-BY-EXAMPLE, then a representative document to search on. 4. If FILTERED SEARCH, then a structure or expression to establish: • fields to "select" • filter expressions • ordering criteria See sub clause 9.2 to see the use of this structure in detail.

RLUSLogicalRecordID	Structure that identifies a logical record managed by RLUS. The structure consists of the following: 1. Alphanumeric or GUID ID representing the record ID. 2. Optional - Alphanumeric or GUID ID representing the system / location IDImplemented in XSD and WSDL as the II (id) structure from HL7 v3. This is most often implemented as part of the XML document structure that makes up the schema for a specific logical record instance. In the machine readable files included this is part of the CCD.XSD schema.
RLUSStatusCode	Exception / result structure from the RLUS operations, containing: 1. boolean value = TRUE if operation is successful, and FALSE if not successful. 2. structure that shall return a set of status-condition in a property named status-conditions that summarize in greater detail the success or failure of the api call. Each instance of a status-condition structure shall contain information pertaining to an individual condition.
RLUSUpdateRequestSrcStruct	Wrapper structure for application calling context for RLUS to support future Put() and Initialize() requests. Logically, it is a similar structure for both Put() and Initialize(), however in the machine readable normative reference files there is: RLUSPutRequestSrcStruct for the Put() operation and RLUSInitializeRequestSrcStuct for the Initialize() operation. The structure consists of the following: 1. Name of the semantic signifier 2. Caller’s Security Context (user/system id, role) 3. Caller’s Network Context (hostname, ip) 4. System ID Context (GUID uniquely identifying the sending system).
RLUSResultSet	Logical record(s) in the format specified by the semantic signifier.
RLUSLocationsResultSet	Array of structures that provides the locations of RLUS services, which can process the targeted data type as specified by the semantic signifier. Each element of the array consists of a structure that contains the following fields: 1. Name of the semantic signifier 2. RLUSLogicalRecordID 3. Caller’s Network Context (hostname, ip) 4. System ID Context (SYSTEMID structure from IXS, aka domain, system id, and application id).

8.2 Details on error and operation status handling

- All API calls shall return the specified status object that represents the outcome (success/failure, code, message, etc.) of what occurred during the API call.
- All API calls shall return an instance of an RLUSStatusCode structure. Example method signature: RL USStatus api_call(param1, param2, ... paramN).
- The RLUSStatusCode structure shall indicate the success or failure of the api call.
- The RLUSStatusCode structure shall return a set of status-condition in a property named status-conditions that summarize in greater detail the success or failure of the api call. Each instance of a status-condition structure shall contain information pertaining to an individual condition.
- For unsuccessful API calls, implementers must return at least one instance of status-condition corresponding one of the total set of conditions encountered during the call. The implementer may also choose to return (n) status-conditions that fully communicate all the individual conditions encountered during the call.
- Implementers choosing to implement a single status-condition for each call should give priority to Errors over Info and Warning severity. But this choice is up to the implementer. They could choose to simply return the first error encountered by whatever error checking algorithm they implement.
- Each error-condition shall have a status-severity enumeration property. The severity enumeration shall consist of INFO, WARN, ERROR, and SEVERE. INFO is considered to have the lowest severity and SEVERE to have the highest.
- Error conditions are to be ordered by status-severity of most severe to least and within severity from generic to specific. Example: An RLUS Put operation:
  - Condition 0 - SEVERE: storage error occurred
  - Condition 1 - ERROR: invalid semantic signifier value (specific)
  - Condition 2 - ERROR: missing one or more required semantic signifier fields (generic)
- There is no inherent mechanism provided within the array of status-condition returned within the IXS State structure to relate any given error-condition to another. Implementers may choose to imply or hint at these types of relationships via the status-detail property of the status-condition structure.
- The status-name component of shall be implemented as an enumeration (need to confirm general capability of communicating enumerations over-the-wire via tests).
- RLUS implementers can optionally provide localized status-message values and if done, return messages in the default locale (language) for that server instance.

9 Platform Specific Model - HL7 v.3

9.1 General

This is a normative reference of implementing the RLUS service interface using the HL7 v.3 RIM in a few specific XML structures:

a. A patient history using the continuity of care document format (CCD).

b. A clinical order using the clinical document architecture (CDA).

c. Descriptions of supporting immunization records, etc. with any defined XSD schema.

9.2 Implementation - Common Structures

Below are common XML schemas to represent the common data structures used by an RLUS service.

RLUSTypes.XSD

This schema file contains most of the RLUS normative data types described in Clause 8. Some are not declared here but in the normative WSDL definitions listed in sub clauses 9.3 - 9.6, as they are specific to a particular operation on the RLUS service interface.

RLUSExpression.XSD

This schema file contains the RLUS normative expression data type that is used to formulate a search expression for the Get(), List(), and Locate() operations described in Clause 7.

NOTE: All machine readable files are available in OMG document number dtc/2010-12-02.

9.3 Profile alignment by Service Interface and Operations

What this profile does is line up the platform independent model to create a specific service interface that supports the Get(), List(), Put(), Discard(), etc. operations but does so by creating a specific WSDL for each data type (Patient, Order, and Immunization) and then types the semantic signifier payload to the specific operations.

Now this could also be done more generically, with an RLUS-HSSP interface with only generic signifiers to represent the data payloads and then the caller could dynamically determine the required schema definition by inspecting it at run-time via the Describe() operation. This is similar to using an anyor variant data type instead of a specific structure to represent the data payload. This anyor variant that represents a logicalRecordInstance parameter on the RLUSManagementAndQuery interface is the minimum support required for a compliant service implementation. The normative WSDL that represents minimum compliance to the RLUS specification is titled “RLUSGenericService.wsdl” in the machine readable file supplement to this specification.

The specific methods used below are more type-safe and have some advantages when using modern web service development tools since the more type explicit approach allows the development tools to create object wrappers around the explicit data types, where the generic approach requires the developer to write code against the XML text format of the semantic signifier directly.

The necessary WSDLs for supporting the Patient Profile and History and Clinical Order are listed in the machine readable OMG document. Also, the framework for how any data type can be supported via the RLUS framework is described in some detail in the next several sub clauses.

The approach to transforming the platform independent model into platform specific implementations based on the concept of a semantic signifier is part of this specification. However, submitters expect for various organizations such as professional societies or other organizations to be developing and evolving semantic signifier definitions and therefore RLUS-compliant platform specific implementations over time. Listed below are 2 key normative representations of the RLUS specification, however many more compliant implementations of the RLUS specification will be developed by these professional societies and other organizations yet will not require revisions of this OMG specification.

9.4 Implementation - Patient Profile and History

Supporting a Patient Profile and History with RLUS could be accomplished a number of valid ways, but the specifics provided here is by using an HL7 v.3 Continuity of Care Document (CCD).

Effectively, CCD is a specific XML Schema that is used as an RLUS semantic signifier and used to type the logical record parameters in the specific WSDL that implements the Patient Profile and History RLUS service instance.

Specifically:

a. The URI XSD that is held in the RLUStypes:RLUSsemanticSignifier meta-data structure points to a CCD.XSD XML schema.

b. The following is inserted as the semanticSignifierDocument type <xs:element ref="ccd:ClinicalDocument" minOccurs="1"> in the Get(), Put(), List() parameter that corresponds to the logicalRecordInstance(s) parameter in the platform independent model.

The specific file where this RLUS service is implemented is in the OMG readable file titled “RLUSPatientService.wsdl” in the machine readable file supplement to this specification.

9.5 Implementation - Clinical Orders Data

Supporting a clinical order (like a drug prescription or lab test) with RLUS could be accomplished a number of valid ways, but the specifics provided here is by using an HL7 v.3 Clinical Document Architecture (CDA).

Effectively, CDA is a specific XML Schema that is used as an RLUS semantic signifier and used to type the logical record parameters in the specific WSDL that implements clinical order RLUS service instance.

Specifically:

a. The URI XSD that is held in the RLUStypes:RLUSsemanticSignifier meta-data structure points to a CDA.XSD XML schema.

b. The following is inserted as the semanticSignifierDocument type <xs:element ref="cda:ClinicalDocument" minOccurs="1"> in the Get(), Put(), List() parameter, which corresponds to the logicalRecordInstance(s) parameter in the platform independent model.

The specific file where this RLUS service is implemented is in the OMG readable file titled “RLUSOrderService.wsdl” in the machine readable file supplement to this specification.

9.6 Implementation - Immunizations, etc.

Immunizations, Lab summaries, Discharge summaries, or any other form of clinical data in document or message form that can be normalized in an XSD with an optionally associated schematron can be processed by the RLUS interface in WSDL like the Patient History and Clinical Order specifics listed previously in the above sub clauses.

The key differences in supporting another semantic signifier type include:

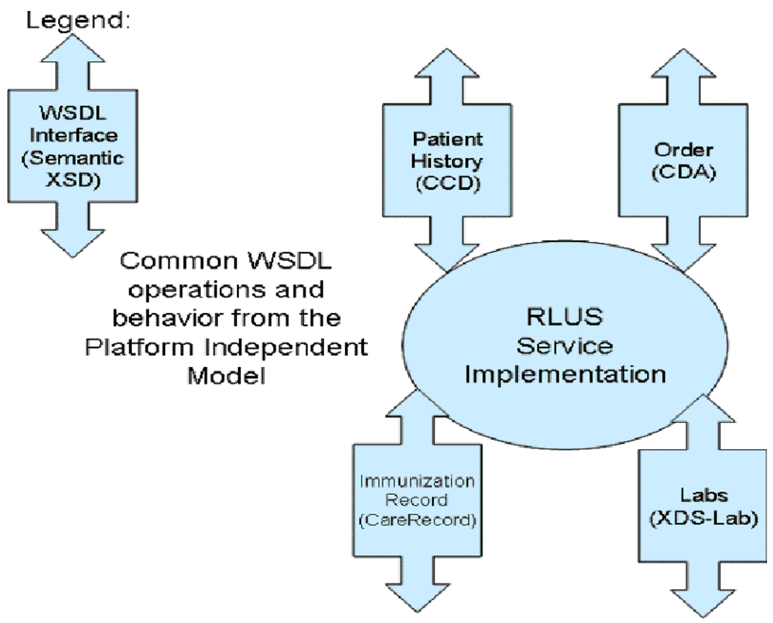
a. The URI XSD that is held in the RLUStypes:RLUSsemanticSignifier meta-data structure that describes the structure of the data to be handled by the interface.

b. The <xs:element ref="cda:ClinicalDocument" minOccurs="1"> element in the Get(), Put(), List() parameter that corresponds to the logicalRecordInstance(s) parameter in the platform independent model.

So, for example to support the care record message or an MS-Lab content profile, an RLUS service would:

- have its metadata registry configured to point to a URI of an appropriate defined XSD to meet requirement (a), and then
- have a specific WSDL, which is defined incorporating that XSD as a specific type of the logicalRecordInstance(s) parameter in each applicable SOAP request and response satisfying the requirement listed in (b) above.

This approach allows for a flexible and loosely coupled deployment of an RLUS service that supports many different kinds of data types. This specific implementation is described in the following diagram.


Legend:
WSDL Interface (Semantic XSD)
Common WSDL operations and behavior from the Platform Independent Model
Patient History (CCD)
Order (CDA)
RLUS Service Implementation
Immunization Record (CareRecord)
Labs (XDS-Lab)

This approach also extends to HL7 version 2.x, which is described later in this specification.

10 Platform Specific Model - HL7 Version 2.x

10.1 General

Supporting version 2.x messages requires only an incremental XSD wrapper and then the same approach can be applied to supporting all H7 2.x messages just as we support HL7 3.x messages and documents listed previously.

The RLUS specification is based on a meta-data container called the “semantic signifier” to describe the structure and key properties of data type. In the HL7 3.x specifics listed above, we used a CCD document to describe a patient history and a CDA document to represent a clinical order. The semantic signifier contains the XSD schema definition which then “types” the semantic signifier. From there all operations of a service implementing an RLUS interface knows that patient history = CCD (for example).

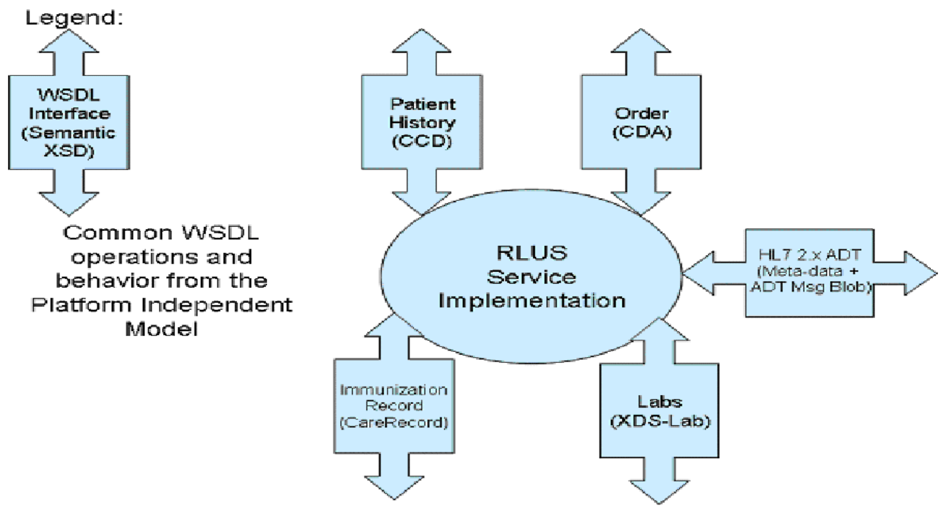
10.2 HL7 2.x Messages

HL7 2.x messages can either be handled by (a) creating an XML schema to represent the 2.x message, or (b) create an XML structure that has the following pseudo form:

```
<HL72.xPatientHistory <PatientID>
<PatientFirstName>
<PatientLastName>
<etc... other key fields>
<HL7message>
/>
```

This XML structure pseudo form uses XML to explicitly type important searchable, key fields as meta-data as XML and then provides a field into which the full HL7 2.x message can be put as a “blob.” This is a similar approach to IHE XDS, which provides a semantically rationalized way to Get(), Put(), List(), Locate(), etc. an HL7 message, but leave it to the underlying implementation to process the full extent of the message content.

Therefore an RLUS service implementation could support 3.x and 2.x messages in the following architecture:


Legend:
WSDL Interface (Semantic XSD)
Common WSDL operations and behavior from the Platform Independent Model
Patient History (CCD)
Order (CDA)
RLUS Service Implementation
HL7 2.x ADT (Meta-data + ADT Msg Blob)
Immunization Record (CareRecord)
Labs (XDS-Lab)

11 Platform Specific Model - Usage of RLUS for Storage and Retrieval of Health Care Documents and Messages

11.1 General

This clause covers how to use the main aspects of RLUS platform specific model to perform its primary functions of brokering storage and retrieval requests between a clinical data consumer and a clinical data provider.

11.2 RLUS for Storage

The primary operation in RLUS to initiate a storage request is through the Put() operation. In this case the client of the RLUS service prepares the following parameters in the SOAP request:

- XML document instance that is valid against the XSD specified in the semantic Signifier meta-data definition for this particular RLUS service interface instance. For example, when using the Patient Profile and History RLUS service interface as described above this XML document instance must be a valid CCD document or the Put() operation will fail on execution. In the case of the Order specifics, it must be a valid CDA document. In the case of Immunization, it would need to be a valid CareRecord, etc.
- A valid RLUSUpdateRequestSrcStruct that provides context to the call providing RLUS with the necessary data support later calls to the Locate() operation.
- A value for the writeCommandEnum. This value if not explicitly provided defaults to an “upsert” behavior, meaning the RLUS implementation is expected to update the record if it already exists in the underlying storage repository or insert if not.
- A specific SOAP request is listed here in the patient _put.xml machine readable file in the specification attachment.
- Handling of data integrity and transactions is a responsibility of the underlying and associated implementation, not the RLUS interface.

11.3 RLUS for Retrieval

The primary operations in RLUS to retrieve data from the underlying clinical storage implementation is through the Get() or List() operations.

Initiating the request to retrieve is primarily configured through the RLUSSearchStruct, which has the following normative structure (this is a snippet from the machine readable attachment. This structure is in RLUSTypes.XSD):

```
<xs:element name="RLUSSearchStruct">
       <xs:complexType>
           <xs:complexContent>
               <xs:extension base="RLUStypes:RLUSSearchStructType">
                   <xs:attribute name="semantic-signifiername" type="xs:string" use="optional"/>
               </xs:extension>
           </xs:complexContent>
       </xs:complexType>
   </xs:element>
```

```
<xs:complexType name="RLUSSearchStructType">
    <xs:choice>
        <xs:element name="searchByCriteria">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="FilterCriteria" type="RLUStypes:FilterCriteriaType" 
minOccurs="0"/>
                    <xs:element name="OrderCriteria" type="RLUStypes:OrderCriteriaType" 
minOccurs="0"/>
                    <xs:element name="SearchAttributes" type="RLUStypes:SearchAttributes-
Type"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="searchByExample" type="xs:anyType"/>
    </xs:choice>
</xs:complexType>
<!-- field elements -->
<xs:complexType name="FieldType">
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="qualifier" type="xs:string" use="optional"/>
</xs:complexType>
<!-- order elements -->
<xs:complexType name="OrderType">
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="direction" use="required">
        <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
                <xs:enumeration value="ASC"/>
                <xs:enumeration value="DESC"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
</xs:complexType>
<!-- the type of search fields -->
<xs:complexType name="SearchAttributesType">
    <xs:sequence>
        <xs:element name="Field" type="RLUStypes:FieldType" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<!-- the type of filtering criteria -->
<xs:complexType name="FilterCriteriaType">
    <xs:sequence>
        <xs:element name="Expression" type="RLUSexp:ExpressionType"/>
    </xs:sequence>
</xs:complexType>
<!-- the type of ordering criteria -->
<xs:complexType name="OrderCriteriaType">
    <xs:sequence>
        <xs:element name="Order" type="RLUStypes:OrderType" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

In preparing this structure for use in the SOAP request for Get() or List() the following and executing it in the underlying service implementation requires following:

- A valid semantic signifier name. The underlying service implementation will use this name to internally execute the Describe() operation and obtain a valid XSD that is used to format and validate the returned XML documents that correspond to the retrieved data from the retrieval processing.
- A choice of search by criteria type that describes whether the search will commence using an Expression or Query-by-Example approach.
- In the Query-by-Example scenario, a valid XML document is expected in the searchByExample element. The RLUS implementation must validate the document against the XSD retrieved by the internal call to Describe() given the semantic signifier name. In executing the search, the underlying RLUS service implementation is expected to find corresponding records or documents whose fields and/or tags match the provide input document with "AND-centered" evaluation. This means the search should select records and/or documents from the underlying storage platform that match all of the fields provided by the input document, not just one.
- In the Expression based scenario, filter criteria, order criteria, and search attributes need to be properly formed for the retrieval to operate successfully. First to note is each of these elements are optional. If an empty sequence of filter criteria is sent that is the equivalent of sending an SQL SELECT statement with no WHERE clause, which is all data of that semantic signifier type, is requested for retrieval. If there is no order clause, then the first record and/or document retrieved by the underlying storage system is the first returned from the RLUS interface, and if search attributes is empty, then a fully formed XML instance is expected for each record and/or document returned.
- The key aspect of each of these expression sequences is how to represent the field name. Since most often the returned and in many cases the underlying data is represented as XML documents, the general form of a field name = <schema path>/< element name>. For example, take the following CCD code for a patient name and address:

```
<recordTarget>
    <patientRole>
    <id extension=" 12345"
    root="76b242b9-1246-4691 -bb1 d-91bb7336e76f"/>
        <patient>
                <name>
                    <given>Henry</given>
                    <family>Levin</family>
                    <suffix>the 7th</suffix>
                </name>
                <birthTime value=" 19320924"/>
        </patient>
    </patientRole>
</recordTarget>
```

To tell an RLUS to search by patient first name, the name in the FieldType would be ".../recordTarget/patientRole/id/patient/name/#given"

- The order clause is done by providing a field name and a direction (either ascending or descending). Using the specifics provided above, if you wanted to order a search by patient last name, the name field of the OrderType structure in the RLUSSearchStruct would be:

```
".../recordTarget/patientRole/id/patient/name/#family"
```

So to find all patients with the first name of "Henry" and the last name of "Levin" and order the results by the last name, the search structure would look like the following:

```
      <rlustypes:searchByCriteria>
   <rlustypes:FilterCriteria>
      <rlustypes:Expression>
         <exp:BinaryTerm text="#given" type="Text"/>
         <exp:Operator type="EqualTo"/>
         <exp:BinaryTerm text="Henry" type="Text"/>
      </rlustypes:Expression>
      <rlustypes:Expression>
         <exp:BinaryTerm text="#family" type="Text"/>
         <exp:Operator type="EqualTo"/>
         <exp:BinaryTerm text="Levin" type="Text"/>
      </rlustypes:Expression>
   </rlustypes:FilterCriteria>
      <rlustypes:OrderCriteria>
      <rlustypes:Order name="#family" direction="ASC"/>
      </rlustypes:OrderCriteria>
   <rlustypes:SearchAttributes>
   <rlustypes:Field 
   name="org.hl7.v3.POCDMT000040ClinicalDocument/recordTarget/patientRole/patient/name/given" 
   qualifier="#given"/>
<rlustypes:Field 
   name="org.hl7.v3. POCDMT000040ClinicalDocument/recordTarget/patientRole/patient/name/fam-
   ily" qualifier="#family"/>
</rlustypes:SearchAttributes> </rlustypes:searchByCriteria>
```

Here is the simpler case where you would be searching directly by a known record ID (likely as part of a Get() operation):

```
      <rlustypes:searchByCriteria>
   <rlustypes:FilterCriteria>
      <rlustypes:Expression>
         <exp:BinaryTerm text="#ext" type="Text"/>
         <exp:Operator type="EqualTo"/>
         <exp:BinaryTerm text="2" type="Text"/>
   </rlustypes:Expression>
   </rlustypes:FilterCriteria>
      <rlustypes:SearchAttributes>
      <rlustypes:Field 
   name="org.hl7.v3.POCDMT000040ClinicalDocument/recordTarget/patientRole/id/extension" qual-
   ifier=" #ext" />
</rlustypes:SearchAttributes> </rlustypes:searchByCriteria>
```

RLUS is not intended to replace the full query capability of an SQL engine in data analytic use cases, but rather to support a standard way to retrieve health care record and document data given very common elements like patient identity, clinical provider identity, etc. Therefore, the structures are designed to facilitate that usage. There is support for queries from a single semantic signifier source per Get(), List(), or Locate() operation using expressions, which are supportable by the structure as specified. There is no support for join, complex sub expressions, or other advanced query techniques by this specification.

12 Compatibility with Other Standards

12.1 Compatibility with HL7

RLUS provides the means to exchange HL7 2.x and 3 messages and documents as semantic signifier and logical record payloads.

RLUS heavily reuses HL7 standards and IHE content profiles for these data payloads. The focus of RLUS is only on the service interface to initiate and handle requests to read and write data from many different implementations of clinical applications and data stores but do so across a common, simple, and type-safe service oriented API.

12.2 Compatibility with IHE XDS Standards

Functionally, RLUS is equivalent to XDS.a and XDS.b. The major difference is that instead of requiring ebXML as the implementation of the service, any suitable storage implementation for the user requirements will do which implements the minimum RLUSManagementAndQueryInterface as specified above. Also, IHE profiles such as PIX/PDQ, ATNA, CT, and XDS content profiles like XDS-Lab can be incorporated alongside RLUS in a site deployment.

Finally, RLUS provides a standard interface for reading and writing data from any source or destination of data in a health information network whether the deployment is point-to-point, hub-and spoke, or a bus / network oriented architecture. XDS is primarily a hub and spoke deployment and XDR is point to point.

13 Semantic Profiles

13.1 General

This interface is to provide an API for the definition and manipulation of RLUS semantic signifiers. This is an optional interface and is not required to be implemented for specification conformance. This is to provide an RLUS implementation with dynamic meta-data configuration capability that can be accessed from various graphical user interfaces or other system APIs.

13.2 CreateSemanticSignifier

This operation provides the means to create a semantic signifier definition.

Input Parameters	1. semanticSignifier - This is the input structure of the semantic signifier to be created.
Output Parameters	1. RLUSStatusCode - Structure that contains the success or failure codes of the executed service operation. Sent as the return value of the operation.
Error Codes	1. Invalid semantic signifier 2. Internal resource processing error
Additional Details	This registers a new data type that is to be managed by the RLUS service.

13.3 Find Semantic Signifier

This operation provides the means to retrieve the semantic signifier definition by name.

Input Parameters	1. semanticSignifierName - This is the name of the semantic signifier to be found.
Output Parameters	1. RLUSStatusCode - Structure that contains the success or failure codes of the executed service operation. Sent as the return value of the operation. 2. semanticSignifier - This is the output structure of the semantic signifier found.
Error Codes	1. Semantic signifier not found 2. Internal resource processing error
Additional Details	This returns the structure of the RLUS semantic signifier data type that the service interface has previously registered by that interface.

13.4 UpdateSemanticSignifier

This operation provides the means to update a semantic signifier structure.

Input Parameters	1. semanticSignifierName - This is the name of the semantic signifier to be updated. 2. semanticSignifier - This is the input structure of the semantic signifier that represents the updated to be applied.
Output Parameters	1. RLUSStatusCode - Structure that contains the success or failure codes of the executed service operation. Sent as the return value of the operation.
Error Codes	1. Semantic signifier not found 2. Invalid semantic signifier 3. Internal resource processing error
Additional Details	None

13.5 ListSemanticSignifiers

This operation provides the means to list all available semantic signifiers that an RLUS service implementation supports.

Input Parameters	None
Output Parameters	1. RLUSStatusCode - Structure that contains the success or failure codes of the executed service operation. Sent as the return value of the operation. 2. ListOfSemanticSignifierNames
Error Codes	1. Internal resource processing error
Additional Details	None

13.6 ListConformanceProfiles

Returns the list of named conformance profiles that the service implementation supports.

Input Parameters	None
Output Parameters	1. RLUSStatusCode - Structure that contains the success or failure codes of the executed service operation. Sent as the return value of the operation. 2. conformanceProfiles
Error Codes	1. Internal resource processing error
Additional Details	This lists a series of named conformance profiles that the RLUS service interface supports in its implementation. A conformance profile consists of a specific semanticSignifier using a specific data payload and structure that has been approved through HL7 or other healthcare SDO. This list is registered within the implementation of the service and cannot be changed through an RLUS API at runtime. This list could be hard-coded or part of a meta-data registry within the specific RLUS implementation.