U.S. Department of Justice (DOJ)

Logical Entity Exchange Specification 3.1 (LEXS)

User Guide

Revision 11

Change History

Rev	Date	Author	Description of Revision
1	10/30/2007	Debby Park	Initial version, chapters 1 and 2 only (LEXS 3.1.0)
2	11/17/2007	Debby Park	Updates to chapters 1 and 2, initial version of chapters 3-6 and appendices A-F (LEXS 3.1.0)
3	12/18/2007	Debby Park	Updates to revision 2 per LEISP reviewers (LEXS 3.1.0)
4	1/18/2008	Debby Park	Updates to revision 3 per LEISP reviewers (LEXS 3.1.0)
5	2/12/2008	Debby Park Jack Wallace	Updates for LEXS 3.1.1, and additional information on associations
6	3/10/2008	Debby Park Jack Wallace	Updates to revision 5 per LEISP reviewers (LEXS 3.1.1)
7	8/7/2008	Jack Wallace	Updates for LEXS 3.1.3
8	1/15/2009	Debby Park Priscilla Walmsley Jack Wallace	Expands text for Number of Structured Payload Terms and Match Score, corrects typos, includes minor wording changes, and adds chapter 7 (LEXS 3.1.3)
9	1/27/2009	Jack Wallace	Includes explicit notation for changes between LEXS versions and draft updates for LEXS 3.1.4
10	2/6/2009	Benjamin Shrom	Updated background sections to be more current, and made general wordsmithing updates based on reviewer feedback.
11	6/23/2009	Debby Park	Adjusted document for conversion to HTML format

Note that this document is for all versions of LEXS 3.1 that were available at the time of writing. Each 3.1 release is backward compatible with all previous 3.1 releases, meaning each new release only includes new elements, loosened cardinality (e.g. from 1-n to 0-n), or type changes that do not impact instances. For example, any instance valid for LEXS 3.1.0 is also valid for LEXS 3.1.1.

The change history above indicates the LEXS version that the guide was updated to include. For example, revision 6 is the last revision made for LEXS 3.1.1 and revision 7 is the first revision that incorporates LEXS 3.1.3 changes.

The most recent version of this document can be used for the current as well as any previous version of LEXS 3.1 since the document indicates additions/changes and the LEXS version for which the change was made. For example, LEXS 3.1.1 added a new Intangible Item Entity, so references in this guide that discuss Intangible Item are noted as "*(3.1.1 and later)*".

Acknowledgments

This document was developed through a collaborative effort sponsored by the U.S. Department of Justice (DOJ) Law Enforcement Information Sharing Program (LEISP) and written by the Georgia Tech Research Institute (GTRI), as a product of a project funded by the Office of Justice Programs (OJP), U.S. Department of Justice,

The following individuals have attended our project meetings, assisted on this research effort, provided input, and reviewed this report:

Debby Park, Editor - GTRI
Benjamin Shrom - GTRI
Jack Wallace - GTRI
John Wandelt - GTRI

Jeremy Warren, CTO - DOJ / LEISP
Boris Shur, Chief Data Architect – DOJ / LEISP
Sudhi Umarji - DOJ / LEISP / Trusted Federal Systems, Inc.
Priscilla Walmsley - DOJ / LEISP / Trusted Federal Systems, Inc.

Table of Contents

1 Introduction

The U.S. Department of Justice (referred to herein as “DOJ”) is transforming the way it shares law enforcement information with its federal, state, local, and tribal law enforcement partners. The vision is to create relationships and methods that allow information to be shared routinely across jurisdictional boundaries to prevent terrorism and to systematically improve the investigation and prosecution of criminal activity. DOJ will achieve its vision by formulating information sharing policies and standard business practices and by creating a unified, DOJ-wide technology architecture that will position DOJ as a committed partner in an information sharing environment of federal, state, local, and tribal law enforcement agencies.

The strategy for DOJ’s transformation is implemented through the DOJ Law Enforcement Information Sharing Program (LEISP). This strategy is the result of a collaborative process involving senior leadership from DOJ component agencies and representatives from across the national law enforcement community. LEISP includes an initiative known as OneDOJ, for sharing DOJ data—from all its components—and is aligned with the Information Sharing Environment (ISE) mandated by the Intelligence Reform and Terrorism Prevention Act of 2004.

National Information Exchange Model, NIEM, is an interagency initiative to provide the foundation and building blocks for national-level interoperable information sharing and data exchange. The NIEM project was formally announced at the Global Justice XML Data Model (Global JXDM) Executive Briefing on February 28, 2005. It was initiated as, and continues to be, a joint venture between the U.S. Department of Homeland Security (DHS) and DOJ with outreach to other departments and agencies. The base technology for NIEM is derived from the Global JXDM.

This document covers the Logical Entity Exchange Specification (LEXS, pronounced "lex"), which leverages and reuses work from both LEISP and NIEM. It is intended to provide sufficient information for development of LEXS-conformant exchanges. The user guide presents a conceptual overview of LEXS version 3.1 and describes the LEXS data model, syntax, semantics, and usage guidance. The document is structured to provide details on the top-level requests and responses and the complex structures they utilize.

1.1 Logical Entity Exchange Specification

LEXS was created to support the primary objectives of LEISP and to minimize the impact of the changing requirements and varied demands for information sharing on the sources and consumers of law enforcement data. While LEXS originated from law enforcement needs, it has been designed for a broad audience and is not limited to use by the law enforcement community.

LEXS is intended to address two aspects of information sharing:

- define and consistently describe units of information to be shared
- define interfaces and protocols to provide (publish) as well as search and retrieve such information

LEXS provides an extensible framework for consistent packaging of the information, with specific placement and markings for various elements of the shared information. The LEXS specification shields both data sources and data recipients from the complexity of multiple interfaces and allows for the multipurpose use of information: for example, a data item created by a source can be consumed by multiple recipients that should be able to understand as much or as little of the data as necessary.

This document describes LEXS 3.1. The different versions of LEXS were released as follows:

- LEXS 1.0 - April 2005
- LEXS 2.0 - February 2006
- LEXS 3.0 - January 2007
- LEXS 3.1 – August 2007

LEXS 1.0 was limited to support the sharing of unstructured data (e.g., text, narratives). LEXS 1.0 is no longer recommended for use because it does not offer support for structured data.

LEXS 2.0 introduced a structured data model to describe real-world objects (e.g., persons, places, locations) and associations between these objects. Since LEXS 2.0 supports the representation of structured data, LEXS 2.0-based systems can support functionality such as link charting / analysis and geo-spatial mapping.

LEXS 3.0 further enhanced the specification by extending the LEXS 2.0 data model in a number of ways based on feedback received from LEXS 2.0 implementation experiences and new capabilities introduced in NIEM 1.0. Most importantly, LEXS 3.0 introduced a mechanism that allows implementers to define customized (e.g., regional, community, domain, agency specific) structured content that can be carried as payload within a LEXS 3.0 envelope

LEXS 3.1 is based on NIEM 2.0. It also adds a Substance entity, and explicitly defines data origin, data destination, data submitter, and data owner. LEXS 3.1 also adds support for search and retrieve in addition to the publication capabilities inherent in LEXS 3.0. LEXS 3.1.1 added an Intangible Item entity, while LEXS 3.1.4 added entities for Credit Card, Document, Hash, Instant Messenger, and Network Address.

It is expected that existing implementations of LEXS will eventually migrate to LEXS 3.1 and any new projects being launched will move directly to this version or the subsequent version in the 3.x series.

LEXS 3.1 specifications can be obtained from the LEXS area on the Office of Justice Programs (OJP) Information Technology (IT) Initiative's Web site at http://it.ojp.gov/jsr/common/viewDetail.jsp?sub_id=256&view=yes&keyword=1. Any updates to this document will also be posted at the same site. In addition, this site also contains downloadable XML schemas for both LEXS 2.0[1] and LEXS 3.0[2].

[1] http://it.ojp.gov/jsr/common/viewDetail.jsp?sub_id=245&view=yes&keyword=1
[2] http://it.ojp.gov/jsr/common/viewDetail.jsp?sub_id=246&view=yes&keyword=1

In supporting the LEISP initiative, DOJ intends to leverage as many of its currently existing or planned information sharing investments as possible. DOJ has identified two investments that will provide capabilities that map well to some of the functions within the LEISP architecture. Deployment of these DOJ investments will facilitate increased timely access to DOJ's law enforcement information. These investments are the OneDOJ and the National Data Exchange (N-DEx) programs. Each investment will be the catalyst to fulfill a portion of the overall LEISP architecture.

1.2 Audience

This document is intended for the reader who needs to understand LEXS 3.1. It also presents introductory and context-setting information regarding LEXS 3.1, which may be of interest to business or policy managers involved in law enforcement and government. This document has been prepared as a reference to communicate best practices and tips to implementers. Subsequent sections of this document contain XML fragments in an attempt to either explain a concept or communicate a best practice. Wherever specific tag names or sample XML fragments differ from the latest versions of the schema, the schema is the authoritative source.

As stated before, this document describes technical specifications. Questions regarding legal, privacy and political matters will not be addressed here. There are no provisions for specific security markings or tearlines. Questions about what data can be shared, what data must not be shared and the process for deciding whether a particular data element is sharable or not is outside the scope of this document. Our focus is to understand specifications that describe how information sharing will occur once a decision has been made that certain information must be shared.

2 LEXS Conceptual Model

To use LEXS effectively, it is important to understand LEXS from a conceptual level as well as the detailed LEXS structures and business rules. This section presents the fundamental concepts underlying the LEXS approach and lays the foundation for the detailed specifications presented later in the document.

2.1 Conceptual Overview

LEXS is based on the structures, standards, and usage guidelines of the National Information Exchange Model, including NIEM Naming and Design Rules (NDR), and is built in the style of NIEM Information Exchange Package Documentation (IEPD). LEXS, however, is not a traditional IEPD. An IEPD is a collection of XML schemas and other documentation that represents a specific information exchange; in contrast, LEXS provides a framework that can be used by a number of different communities to create IEPDs documenting a number of different exchanges. LEXS promotes broad information sharing since any information consumer from any community that understands LEXS can understand and process a basic level of information in any LEXS-conformant exchange.

2.1.1 NIEM Overview

NIEM, the National Information Exchange Model, is the result of a collaborative partnership of agencies and organizations across all levels of government (federal, state, tribal, and local), and with private industry. The purpose for this partnership is to effectively and efficiently share critical information at key decision points throughout the whole of the justice, public safety, emergency and disaster management, intelligence and homeland security enterprise.

NIEM provides structure, standards and methods for defining and sharing information exchanges between and within agencies and domains. Developing and implementing exchanges using NIEM means that the major investments local, state, tribal, and federal governments have made in existing information systems can be leveraged, and that these government agencies can efficiently participate in a national information sharing environment. NIEM standards enable different information systems to share and exchange information irrespective of the particular technologies at use.

NIEM can be thought of as a data model and a reference vocabulary from which XML Schema-based data components are constructed. These components (which are XML data elements) serve as the basis for information exchanges. In NIEM and in this document, the term **element** refers to a unit of information which may be simple (indivisible) or complex (consist of other elements). In conjunction with the concepts and rules that underlie the NIEM structure, maintain its consistency and govern its use, these NIEM data components can be reused by information practitioners to create an Information Exchange Package Documentation (IEPD). An IEPD is a collection of XML schemas, XML instances, and other documentation and artifacts that is the electronic representation of the rules governing an information exchange.

2.1.2 LEXS Overview

LEXS provides a flexible, NIEM-based framework used for the creation of NIEM-conformant IEPDs for information sharing, both for publishing information and for system-to-system federated searches.

LEXS specifies a set of schemas that establish consistent definitions supporting publication, search, and retrieval, both at a data level and a structural level. LEXS data elements and structures are organized into components that support the underlying actions required to accomplish the publication, search, and retrieval operations fundamental to information sharing. All LEXS-conformant implementations are based on well-defined structures allowing for a common understanding of what data is available and how it is organized. The LEXS specification was designed and built based on input from a number of programs and, therefore, LEXS is not limited to a specific exchange or a small set of exchanges. LEXS has been designed for use by a broad audience.

LEXS utilizes a generic paradigm for information sharing called the **data item**. A data item is whatever the source considers a logical unit of information. For example, to an incident-based reporting system, a logical unit of information is an incident report that may contain activities, people, places, and things. To a prison system, a logical unit of information is an inmate record that may just have information about a person. The data item concept provides a single, generic container that can be used to encapsulate different types of data needed by various communities. The data item can be thought of as a collection of structured entities, attributes of these entities, relationships between these entities and unstructured textual information.

Since no set of NIEM subset and extension schemas can support all communities due to varying needs, LEXS supports multiple **levels of understanding**. There is a <u>base</u> level of data that all LEXS implementations understand, plus a well-defined extension mechanism that allows communities to plug in additional content, referred to as **structured payload**, without impacting anyone’s understanding of the base level. Since all LEXS implementations understand the base level, communities only have to develop an Information Exchange Package once in order to be able to share information with all other LEXS implementations. Where additional content is required for a selected community, modules can be written to deal specifically with information in the structured payload rather than rewriting the implementation for a completely different exchange.

LEXS defines a high level set of commonly understood, structured **base objects** which represent real-world objects such as person, location, or vehicle. For each base object, LEXS specifies NIEM content which provides a wide range of data representation capabilities. LEXS includes a large number of NIEM roles and associations and defines additional roles and associations needed by the LEXS community to provide detailed context for all data. These base objects, roles, and associations provide a great deal of flexibility to communities so each can define their own data items. LEXS groups these base objects and their applicable roles into **entities**. For example, a LEXS person entity includes a person base object and roles such as subject, witness and victim. In this document, the term **object** is used to refer to a base object or a role.

LEXS also provides mechanisms so that sources can indicate that two entities are the same, or may be the same, in order to make local information known globally. For example, a data source may have an incident report involving Joe Smith and another incident report involving Joe Smith where the underlying person is known to be one and the same (i.e. same record in the database). So LEXS lets the data source indicate that it knows that both reports refer to the same person. As another example, a data source may have an incident report with Jane Doe and another incident report with Mary Doe, and through some analysis (either due to an investigator making the determination or perhaps through some form of automated entity resolution) the data source indicates that the underlying person is believed to be the same. In this case, LEXS lets the data source indicate that Jane Doe and Mary Doe may be the same person.

2.1.3 LEXS Information Environment

Federal law enforcement agencies, as well as state, local, and tribal organizations, often collectively referred to as regional organizations, have a wealth of criminal history and incident information. When two organizations mutually allow for two-way sharing of law enforcement information, with each party retaining ownership and possession of its information they are partners in law enforcement information sharing and each of the systems is referred to as a **partner system.**

While LEXS is based on a general model for information sharing, it is helpful to consider a specific information sharing initiative, to gain a better understanding of the concepts in the more general model. Consider the OneDOJ initiative referred to earlier. OneDOJ is the data repository for full-text and structured, shareable, sensitive but unclassified (SBU), DOJ law enforcement data, which serves two main functions: providing regional systems with access to DOJ's data and providing DOJ users with access to data in regional systems. Since OneDOJ functions as the data repository on behalf of DOJ, DOJ investigative components (e.g., the Bureau of Alcohol, Tobacco, Firearms, and Explosives (ATF); the Drug Enforcement Administration (DEA); the Federal Bureau of Investigation (FBI), and the U.S. Marshals Service (USMS)) publish their sharable data to OneDOJ. In this specification the term **data source** is used to refer to a system that is operated by an organization (e.g., DOJ investigative component) that publishes information to a data repository. The data repository that receives and ingests the published information is referred to as a **data consumer**, in this case, OneDOJ. The data source publishes periodically as determined by operations guidelines. A **service provider** is the system or application that provides access to the data, such as LInX or ARJIS.

In the LEXS model, queries are performed between partner systems. When a user forms a query by describing the information he is looking for, he gets back a list of data items which match that query. The query may be executed against the local repository where the user is "logged in" and/or the user's system may forward the search request to a partner system that it can interoperate with. The partner system in turn can execute the same query against its own repository and return results to the user's system which then displays the information to the user. In the scenario just described, where a query is forwarded to a partner system we refer to the query as **federated**.

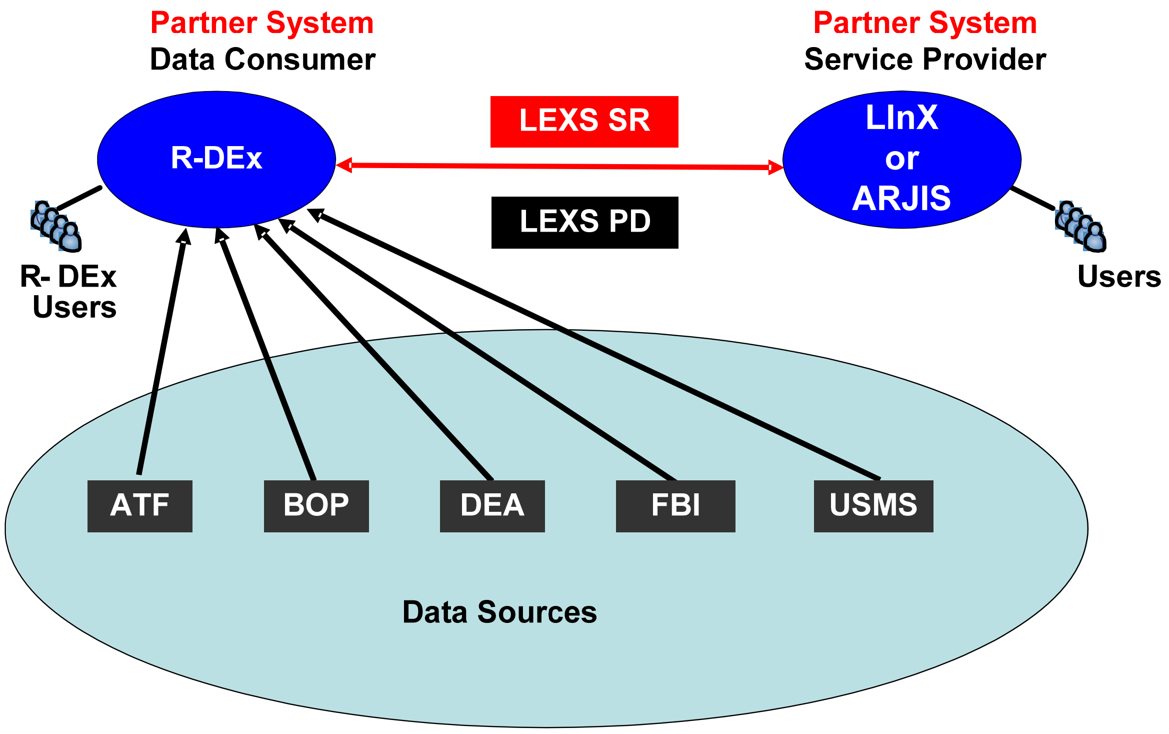


Figure 1. LEXS SR and LEXS PD in OneDOJ

The terms data source, partner system or data consumer do not define system types, but describe roles that systems can have. The same system can have more than one role; hence one or more of these terms may apply to the same system. For illustrative purposes, Figure 1 shows how these terms can be applied to some of the participants in the OneDOJ environment. In this context, the term data source refers to systems operated by ATF, BOP, DEA, FBI, and USMS that are currently publishing (uploading) information to OneDOJ, the data consumer, using LEXS. On the other hand, LInX and ARJIS are not data sources in the context of this definition. Though LInX and ARJIS may operate as data sources or data consumers with respect to other systems, they are not data sources or data consumers for OneDOJ. However, both LInX and ARJIS are partner systems in relation to OneDOJ since this term refers to a system that interacts with OneDOJ by allowing its users to remotely submit queries to OneDOJ. When OneDOJ responds, the partner system presents the user who submitted the query with results from OneDOJ. The same applies when a OneDOJ user makes a query against the partner system. Note that in this example OneDOJ has the role of data consumer in relation to ATF, BOP, DEA, FBI, and USMS and the role of partner system in relation to LInX and ARJIS.

ATF, BOP, DEA, FBI, and USMS are not operating any partner systems with OneDOJ in the context of this definition. Currently users from ATF, BOP, DEA, FBI, and USMS access shared information by logging directly into OneDOJ.

The term LEXS PD (Publication and Discovery) applies to the interface between data sources and data consumers, while the term LEXS SR (Search and Retrieval) applies to the interface between partner systems. It is possible that a given system is both a data source and a partner system.

In addition, it must be noted that we have used the term **data source** to refer to a system that is operated by an organization (e.g., DOJ investigative component) that publishes information to a data repository. In general, unless explicitly stated otherwise, the assumption is that the data source “owns” the information that it publishes. It is indeed also possible for the data repository to republish information that it has previously received from another source. When a data repository republishes information that it does not own, it is acting as an aggregator and to distinguish this case from one in which a data source only publishes what it owns, we use the term **data submitter** to refer to aggregators. We use the term **data owner** to refer to the original owner of the information.

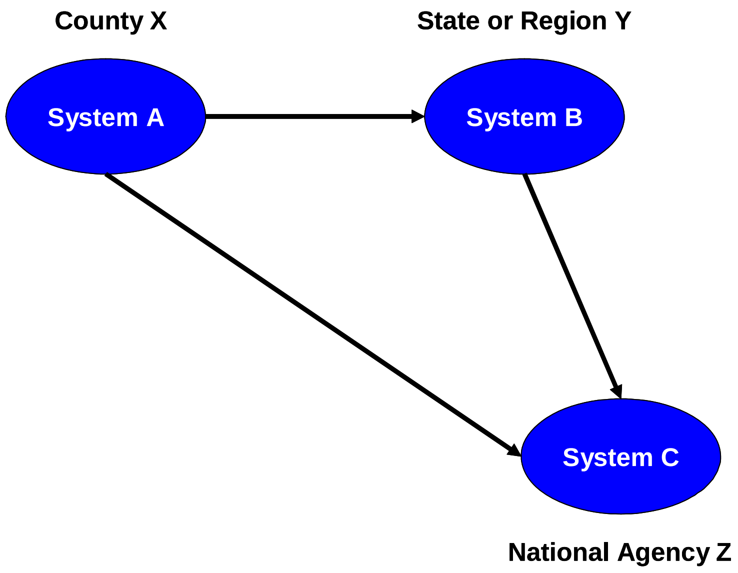


Figure 2. Data Submitters vs. Data Owners

In Figure 2, system B may be a data submitter to system C for information owned by system A. For a particular piece of information that was published to C, through such a route, A is the data owner and B is the data submitter. In addition, system A may also submit directly to C in which case, it is both data owner and data submitter for information published in that operation. See Appendix F for additional details.

2.2 LEXS Supported Operations

Two basic categories of operations are defined by separate LEXS schemas which are supported by shared schemas that provide common definitions of real-world objects and structures.

LEXS Publication and Discovery (LEXS PD) is for publishing and updating data from a source to a consumer. It is currently used for data owners (or submitters) to publish to OneDOJ, N-DEx and other information sharing systems.

LEXS Search and Retrieval (LEXS SR) is for system-to-system federated searches and allows result drill-down to obtain more detailed information. It is currently used for interconnections between OneDOJ and partner systems such as LInX, ARJIS, T-DEx, etc.

2.2.1 LEXS PD

LEXS PD supports the action of publishing sharable information to a data repository. LEXS PD provides the data elements and structures required to represent the data and metadata associated with publishing data. The data repository receives and records the published information, generally, for the purpose of analysis, making it easier, faster, and less expensive to share data. In publishing data, a source system submits one or multiple data items in a single, one-way action to a data consumer; LEXS does not define an acknowledgment action.

2.2.2 LEXS SR

LEXS SR represents a number of requests and corresponding responses that support data search and retrieval. LEXS SR supports requests that fall into two broad categories, data requests and service provider requests.

LEXS data requests provide the mechanisms for retrieving data from partner systems and include four specific requests: text search, structured search, data item request, and attachment request.

LEXS service provider requests ask for system information about partner systems including capabilities, data owners, and availability.

2.2.2.1 Data Requests

In LEXS, search and retrieval of data is a multi-step process. A search request uses information that broadly identifies the target being sought and returns possible candidates for the user to examine further. In general, a search query is followed by a data item or attachment request. For instance, a search request for information about a person might be issued using a first and last name. Usually this is not enough information to specifically identify a single person. As a result of the search, the queried systems respond with information related to any person having the specified first and last name. The intent is that the user can narrow the search by reviewing the search response and then request more detailed information on a specific data item or a limited set of data items.

LEXS SR provides two search requests. A **structured search request** looks for specific elements with certain values, such as a first name of "John" and a last name of "Smith." A **text search request** is used to search for a value in any context, that is, no element name is specified. Depending on the implementation, the text search could be performed on unstructured data, such as a report, or on structured data, such as a name or narrative element. For example, a text search for the phrase "John Smith" on unstructured data might find a match in an incident report. A text search on structured data might find "John Smith" in the PersonFullName field.

The response from either a structured search or a text search can contain information about data items and/or attachments, such as mug shots and documents, associated with the search parameters. A **data item request** is used to get details on a specific data item. An **attachment request** asks for one or more Attachments described in a search response. Either a data item or an attachment request is a follow-up query to a search request and uses the unique identifier supplied by the search response to identify a single target in a single partner system.

A search response may be sparsely populated and data items may contain only elements that are commonly used to help discriminate one data item from another. A data item response should contain all available details and contextual information associated with the requested data items.

2.2.2.2 Service Provider Requests

For service provider requests, LEXS defines requests and corresponding responses that ask for specific types of information. For instance, one request allows a data consumer to ask for information about a service provider's data owners. The response contains the list of data owners available, including capabilities supported over each data owner's data set, such as whether the data owner supports structured searches, unstructured searches or both. There are additional requests and responses defined to determine the capabilities and the availability of the service provider. These requests can be issued alone, preceding or following any other LEXS request.

2.3 Key Concepts

There are a number of key concepts that underlie the design of LEXS. To understand how to use LEXS it is essential to understand these fundamental building blocks.

2.3.1 Message

The **message** is the basic structure that wraps LEXS requests and responses. In reality, LEXS defines a number of different message structures for different purposes, but all message structures contain metadata, and usually additional information specific to the request or response they support. For example, a Publish Message includes one or more data items plus attachments and message metadata. Search response messages are similar but may also include warning or error messages, identifiers that can be used to "drill down" for additional details, and contain information about attachments without the actual attachments. Other messages are defined for retrieval of specific data items and attachments, for querying data, and requesting system information from a service provider. The details of the various messages and other LEXS structures are discussed later in the document. The Publish Message in the diagram below is shown as an example and provides a basis for the discussion of other key LEXS concepts.

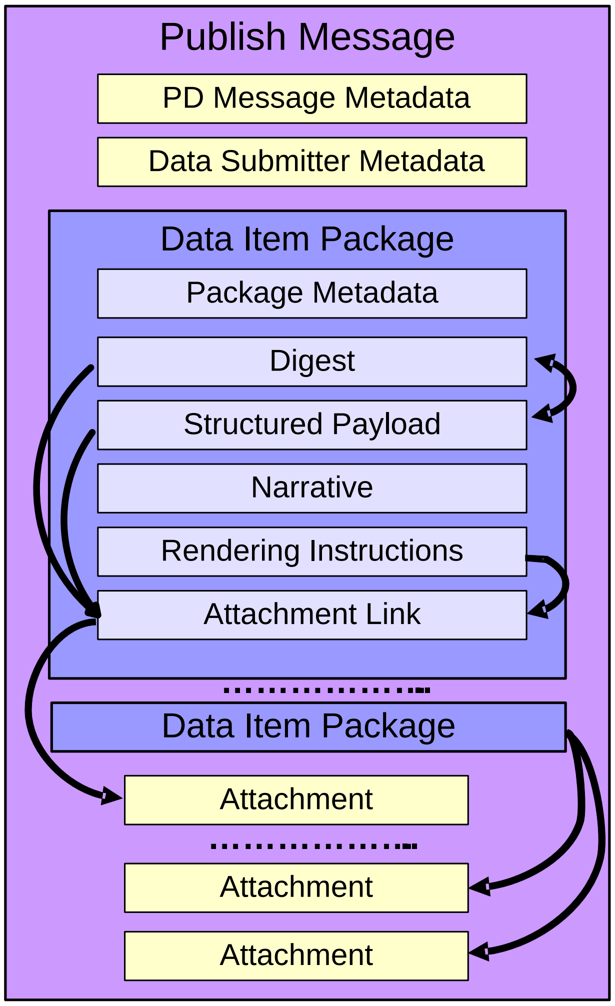


Figure 3. Publish Message

2.3.2 Package

In LEXS, the term **package** refers to a standard representation of any data item used for information sharing. LEXS defines two similar package structures to support data item publication and data item retrieval. Both package structures contain metadata, a Digest, and optionally multiple Structured Payloads, Rendering Instructions, and/or Attachments. As mentioned above, LEXS uses the term data item to refer to the standard, generic, flexible unit of information sharing as it may exist at the source system. For instance, data item might refer to a record in a database. When an application extracts this data item or record and encodes it in a format that is compliant with the LEXS schema, the data item is contained within a package structure. The data source may or may not encode all the information in the data item into the package based on the business rules that it must follow. Conceptually, a package represents a unit of information that is self-contained and able to convey the knowledge from the data source to the data consumer or between partner systems. The data source (owner of information) ultimately decides what information is relevant and should be shared, while the definition of a package in LEXS allows the data source to map such information to a standard format.

2.3.3 Metadata

Metadata, which is information about the data being shared, exists at several levels and is organized into different structures based on content and usage. For example, the Publish Message illustrated in Figure 3 includes PD Message Metadata and Package Metadata. The PD Message Metadata contains the LEXS version used within the document, the date and time of the message, a message sequence number, and an indication of the sensitivity of the data being shared. The Package Metadata provides a unique identifier for the package, contact information for the individual who can provide additional information about the data item, data owner information, dissemination criteria, and a number of other pieces of information about the data item contained within the package. Some metadata structures are specific to one LEXS component and others are used in a number of LEXS components. For instance, the PD Message Metadata is specific to the Publish Message, but Package Metadata is used in all messages that contain data item packages.

2.3.4 LEXS Digest

The concept of the **Digest** is central to the power of LEXS as a framework for information sharing. A LEXS data item may represent any kind of underlying information—a warrant, an incident, a customs manifest. The Digest is the common anchor for systems to use to handle this heterogeneous data without having to understand the specific context and meaning of the source. As long as the entities relevant to the packaged data item are represented in the Digest, users will be able to discover, link, map, etc. the information within.

Even though the Digest provides the common level of understanding, it does not mean that all sources have to populate all elements, or that all consumers have to use all elements; merely that at a schema level all applications understand the Digest. Implementers only need to build one module in order to produce or consume a basic set of data understandable by many. It also means that implementers do not have to develop large applications for each exchange, but rather build one that handles the basics and then additional smaller modules in order to produce or consume more complex exchanges.

The Digest defines a high level set of entities, each containing a basic set of NIEM elements, and in some cases extensions. LEXS 3.1 defines the following high level entities:

Activity	Firearm	Person
Aircraft	Hash *(3.1.4 and later)*	Substance
Credit Card *(3.1.4 and later)*	Instant Messenger *(3.1.4 and later)*	Tangible Item
Document *(3.1.4 and later)*	Intangible Item *(3.1.1 and later)*	Telephone Number
Drug	Location	Vehicle
Email Address	Network Address *(3.1.4 and later)*	Vessel
Explosive	Organization	

The objective of the Digest is to present the most common characteristics of these real-world objects that can be supported by any (including the least sophisticated) data source or data consumer. Digest-level data objects may be further augmented or described with additional details in the Structured Payload or the unstructured Narrative portion of the package. The Digest may also contain associations and roles related to the base objects.

2.3.5 Structured Payload

The **Structured Payload** provides an extension mechanism so that different communities can "extend" the Digest to incorporate richer data sets without compromising compatibility across applications. Therefore, users of LEXS are not limited to just the high level entities shown above and their included elements/roles/associations, but can define Structured Payloads to build upon the contents of the Digest.

Each Structured Payload is based on schemas created by communities outside of LEXS. Ideally these independent schemas are built to be aware of the LEXS Digest, that is, Structured Payload entities build on and link to LEXS Digest entities. A Structured Payload may also define a new entity where the LEXS Digest does not provide a foundation. Structured payloads can be ignored if not recognized/understood/implemented by consumers. Exchange instances do not have to include any Structured Payloads. Each Structured Payload includes metadata that specifies what community defined the Structured Payload schema. Consumers can use that metadata to determine if the contents can be processed or needs to be ignored. If the consumer can understand Structured Payloads from that particular community, then the content of the Structured Payload is extracted and processed based on the schema for that particular Structured Payload.

Since Structured Payloads are based on separate schemas, Structured Payload instances must be valid as standalone instances. While the data contents of the Structured Payload may not make logical sense by itself without the contents of the Digest, the Structured Payload instance must be valid against the Structured Payload schema.

Exchange instances can include multiple Structured Payloads and a consumer might understand some and not others. Structured payloads can build upon the contents of the Digest, or even upon the contents of another Structured Payload. Some consumers may understand everything in an instance, others may understand the Digest and one Structured Payload, and still others may only understand the Digest.

LEXS includes a mechanism to connect elements in the Structured Payload to elements in the Digest. In general, LEXS is designed so that Structured Payloads augment the contents of the Digest; it is not expected that Structured Payloads include Digest contents plus additional elements. LEXS does allow for the use of Structured Payloads that are not LEXS-aware, and therefore cannot "point up" to elements in the Digest. There is also a mechanism that allows elements in the Digest to "point down" to elements in the Structured Payload(s).

2.3.6 Narrative

The Narrative structure provides a container for **unstructured data** associated with the data item. The LEXS provision for unstructured data allows for the inclusion of data sources that are text-based or elements from any data source that are free-form in nature and can not easily be represented as attribute-value data components.

2.3.7 Rendering Instructions

Rendering instructions are used to display the information in a package in a specific viewing or output format for human users. A data source may want to influence the way in which the information it shares is viewed by a user at a consuming system. Using a specific rendering allows a source to present data items more intuitively to users who may not be familiar with the underlying context of the data. This is especially true when the package contains information that is specific to an agency or community and the data source is knowledgeable about how best to format the content to make it more comprehendible and user friendly. Since LEXS allows for the sharing of heterogeneous data from different sources with different contexts, user interfaces for displaying LEXS data will often need to be very generic, sacrificing intuitiveness.

Rendering instructions come in two general forms: instructions (i.e., XSL) for converting the XML data in the package into a suitable display format, and instructions to display a pre-rendered version of the data item (i.e., an attached image or document). The HTML rendering format is frequently the choice for viewing reports online and other rendering formats, such as Adobe Acrobat Portable Document Format (.pdf) files, are ideal for distributing large page-oriented reports that can be sent to a printer (assuming of course, that printing is authorized) or viewed using an Acrobat Reader.

2.3.8 Attachment and Attachment Link

LEXS Attachments are based on the NIEM Binary element, and may contain data-related binaries such as a mug shot or fingerprint, or may contain binaries or stylesheets used by LEXS Rendering Instructions. Attachments are at the message level rather than the package level since each may be part of multiple packages, such as a message with multiple incident reports that include the same person and his mug shot.

An Attachment Link provides a mechanism to connect an Entity in a package to an Attachment, such as a person to a mug shot Attachment or a person physical feature to a tattoo image Attachment.

2.3.9 Associations

LEXS uses the NIEM association mechanism. An **association** represents a specific relationship between objects. Associations are used when a simple NIEM or LEXS element is not sufficient to specify the relationship clearly and when there are properties of the relationship that are not attributable to the objects being related. A NIEM association type represents a specific relationship between objects, captures additional information about the relationship, and contains references to the objects it associates. For instance, use of the Marriage Association indicates a marriage between person entities. This association contains references to each spouse, perhaps additional information such as the date that the marriage started, the date that the marriage ended, and the status of the marriage (e.g., married, separated, divorced).

Most associations in LEXS are imported from NIEM, but the LEXS Digest defines a number of LEXS-specific associations. Most of the LEXS associations add relationships that LEXS stakeholders need to be able to represent, but that are not available in NIEM. LEXS provides associations to relate a digest-level entity to a specific message attachment relevant to that entity, for example to relate a person to a fingerprint image or a telephone to a recording of the conversation. There are also associations to link an object in the Structured Payload to the corresponding Digest object.

The associations available in LEXS allow exchanges to provide context for the data, beyond what is available in the entities themselves. For example, LEXS includes a simple activity entity, rather than all the various flavors of activity that exist in NIEM, such as incident or case. LEXS includes some associations for activities in general as well as more specific associations such as one for an incident witness. Therefore, exchanges can provide a great deal of context for the data without LEXS becoming too large to use. Structured payloads can define additional associations if those provided in the digest are not sufficient.

2.3.10 Roles

LEXS uses the NIEM concept of **role** to represent a particular context or activity for a base object. The role object represents data specific to the role and contains a reference to the base object. For instance, weapon is considered a role of an item. A weapon has a reference to the item that is the weapon (the base object) plus additional information specific to weapons, including usage text and references to the activity the weapon was involved in and the person who used the weapon. The LEXS Digest schema defines a few additional roles, such as Informant, Inmate, and Protected Party, which are needed by LEXS stakeholders but not available in NIEM.

2.3.11 Levels of understanding

The LEXS Digest concept allows data sources to encode a well-defined representation of people, places, activities, and things as well as associations among them as described previously in this document. LEXS also includes a layered mechanism for communities to define entities, roles, associations, structures and elements that are not defined in the LEXS schema. Communities that need additional data not provided in the Digest can supply that data in one or more Structured Payloads. This allows law enforcement groups or projects who are interested in information sharing to leverage LEXS as a base while developing their own specialized schemas targeted to address their own business missions. For example, N-DEx has defined Information Exchange Packages (IEPs) based on LEXS 3.0 and 3.1.

This layering concept provides multiple **levels of understanding** by allowing communities to extend the LEXS Digest with community-specific Structured Payloads. Layering and levels of understanding are best illustrated through examples described below.

An incident community needs to exchange information beyond what is available in the Digest and, thus, creates its own Structured Payload schema that builds upon what is in the Digest. Bomb and arson, and cyber communities review the Digest contents along with the incident Structured Payload schema and determine that what the incident community has provided doesn’t include everything they need, but that it supplies a good foundation. So rather than creating their own Structured Payload that just builds upon the Digest, each develops an individual Structured Payload schema that builds upon what is in both the incident Structured Payload and Digest. There is no limit to how many layers there can be and yet another specialized community, the terrorism bomb and arson community, can utilize the bomb and arson payload that in turn builds upon the incident payload. The arrest community reviews what is available, and determines that the incident payload is not really suitable, and so develops its own Structured Payload built on the Digest. The federal arrest community also reviews what is available, and determines that the arrest payload provides a better foundation than any other available payload, so the federal arrest community builds a payload that in turn builds upon the arrest payload. These layers and how they build upon each other are shown in Figure 4.

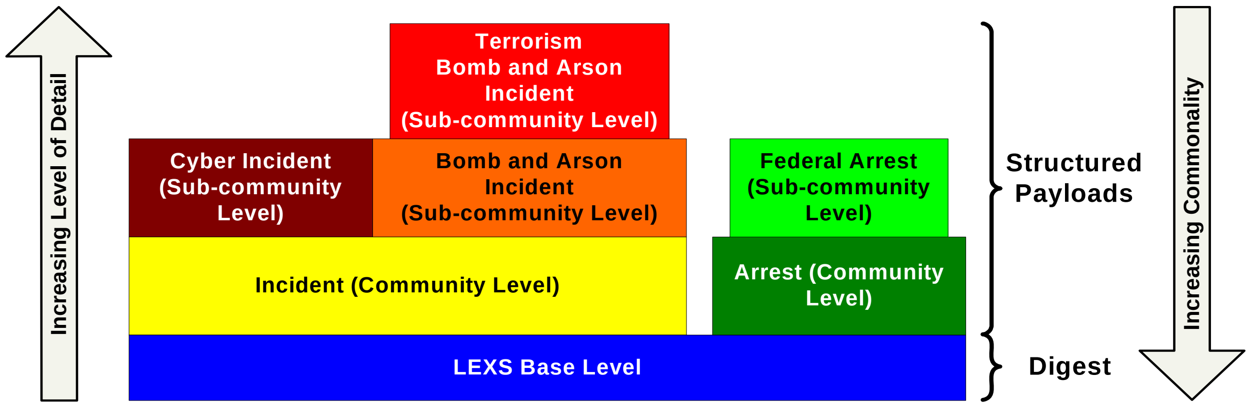

Increasing Level of Detail
Terrorism Bomb and Arson Incident (Sub-community Level)
Cyber Incident (Sub-community Level)
Bomb and Arson Incident (Sub-community Level)
Federal Arrest (Sub-community Level)
Incident (Community Level)
Arrest (Community Level)
LEXS Base Level
Structured Payloads
Digest
Increasing Commonality

Figure 4. Structured Payload Layering

To illustrate the levels of understanding, the diagram above can be used in conjunction with a data element example.

The LEXS Digest defines a person to include a person name, using the NIEM Person and PersonName. The Digest's PersonName only includes first, middle, last and full name. If the Incident community determines that they need a person's title as well, their Structured Payload would include a PersonName with NIEM's PersonNamePrefixText. So the Incident community can build an instance that represents a person with the name Dr. John Doe. If the Cyber Incident community determines that the Incident Structured Payload plus the Digest has most of what they need but they also need a hacker name, then the Cyber Incident community's Structured Payload might include a new element called HackerName. So the Cyber community can represent Dr. John Doe with hacker name Doc.

If the Cyber community sends such an instance to the Arrest community, the Arrest consumer would be able to determine that it does not understand either the Cyber or Incident Structured Payloads, but would still understand John Doe. So in this case, the Arrest community only understands one level, the Digest.

If the Cyber community sends the same instance to the Incident community, the Incident consumer would be able to determine that it understands the Incident Structured Payload, but not the Cyber Structured Payload. So the Incident consumer would understand Dr. John Doe. So in this case, the Incident community understands two levels, but not the third.

If the Incident community sends an instance to the Cyber community, the Cyber consumer could understand everything in the instance, such as Dr. John Doe. But an Incident Community instance would never include a hacker name. So in this case, the Cyber community understands both levels available from the producer (the Incident community), but cannot get information in the Cyber communities third level.

3 Publish Operation

There is a single publish operation, doPublish, which is used to submit information to a data consumer. The doPublish operation uses a PublishMessage. LEXS allows multiple PublishMessages in a single doPublish operation and wraps the messages in a PublishMessageContainer.

The structure below shows a high level representation for the PublishMessageContainer. The PublishMessage is described below. Descriptions for the elements are provided, along with cardinality which is shown in brackets. The structures that support this message are addressed in section 5.

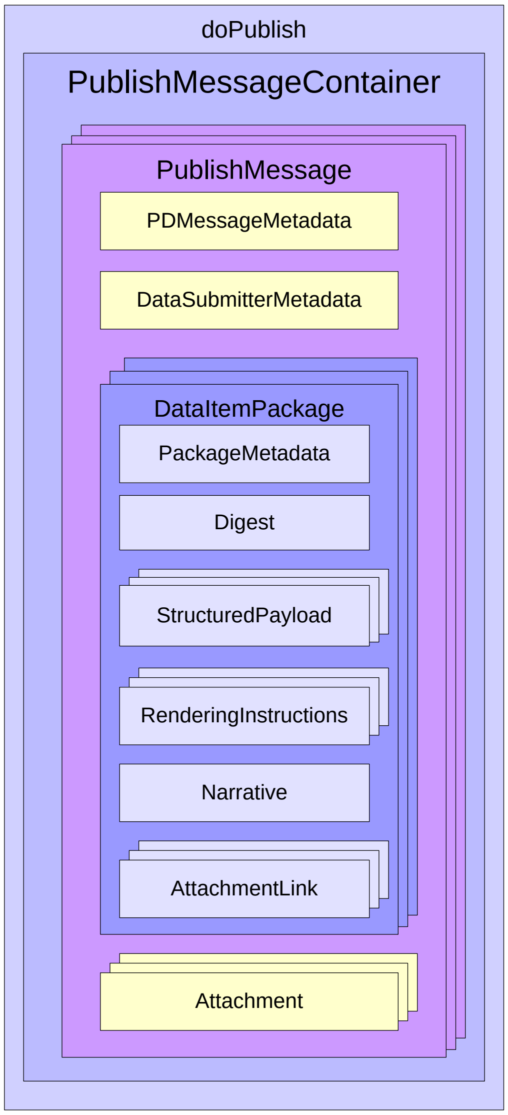


Figure 5 - High level view of doPublish

PublishMessage [1-n] – container.

PDMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, and an identifier for the message. See PDMessageMetadata in section 5.1.13 for a more detailed description.

DataSubmitterMetadata [1] – Includes the organization, the system name, and contact information for the submitting agency and system. See MessageOriginMetadata and DataSubmitterMetadata in section 5.1.10 for a more detailed description.

DataItemPackage [0-n] – Includes Package Metadata (which includes the owner of the data item), Digest, and possibly Structured Payload(s), Rendering Instructions, and Narrative. See DataItemPackage in section 5.1.4 for a more detailed description.

Attachment [0-n] –Includes the AttachmentURI, the Attachment binary, domain specific attributes. See Attachment in section 5.1.1 for a more detailed description.

4 Search and Retrieve Operations

Search and retrieve operations are supported in LEXS 3.1 by a number of requests and corresponding responses as shown in the following table.

Request Message	Response Message
<doTextSearchRequest>	<doSearchResponse>
<doStructuredSearchRequest>	<doSearchResponse>
<getDataItemRequest>	<getDataItemResponse>
<getAttachmentRequest>	<getAttachmentResponse>
<getCapabilitiesRequest>	<getCapabilitiesResponse>
<getDataOwnersRequest>	<getDataOwnersResponse>
<getAvailabilityRequest>	<getAvailabilityResponse>
<getDomainRequest> *(3.1.4 and later)*	<getDomainResponse> *(3.1.4 and later)*

The order in which these messages are discussed is based on usage. A search is generally the first operation, which is often followed by a request for a data item and/or an attachment. Requests for information about a data provider can occur at any time or not at all. Each request and response message is described below and the structures that support these operations are addressed in section 5.

4.1 doTextSearchRequest

doTextSearchRequest is used to search for a value in any context, that is, no element name is specified. This request is similar to a Google-type search. Depending on the implementation, the text search could be performed on unstructured data, such as a report, or on structured data, such as a name or narrative element. For example, a text search for the phrase “John Smith” on unstructured data might find a match in an incident report. A text search on structured data might find “John Smith” in the PersonFullName field.

This request includes information on the user making the request, the number of matches requested, what kinds of data should be returned as part of the response, data item categories requested, what data owners should be queried, what Structured Payloads should be returned (for structured data responses), sort order when there are too many matches to return all hits, identifier for this request, LEXS version being used by the requestor, the date/time the request was generated, and the query itself.

The structure below shows a high level representation for TextSearchRequestMessage. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

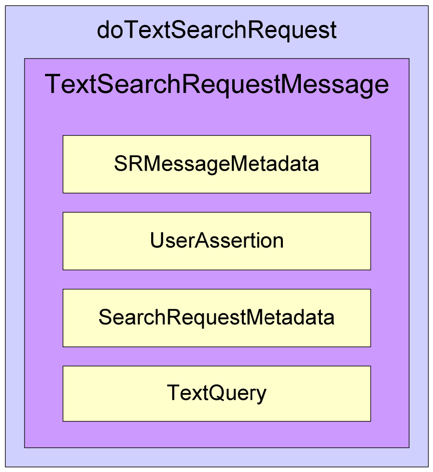


Figure 6 - High level view of doTextSearchRequest

SRMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination. See SRMessageMetadata in section 5.1.19 for a more detailed description.

UserAssertion [1] – Includes information about the user making the search request. See UserAssertion in section 5.1.24 for a more detailed description.

SearchRequestMetadata [1] – Includes information about the number of hits requested, what kind of data is requested, data item categories requested, data owners to be queried, what Structured Payloads should be returned (for structured data responses), how long the requester will wait for a response, and an identifier for the request. Additional fields are included to support the request for additional hits. See SearchRequestMetadata in section 5.1.16 for a more detailed description.

TextQuery[1] – LEXS 3.1 SR supports an unstructured search capability, including wildcards and matching all terms, any terms, or exact phrase. Logical operators AND as well as OR may be specified in the string. See TextQuery in section 6 for a more detailed description.

4.2 doStructuredSearchRequest

doStructuredSearchRequest looks for specific elements with certain values, such as a first name of “John” and a last name of “Smith.”

This request includes information on the user making the request, the number of matches requested, what data owners should be queried, what Structured Payloads should be returned (for structured data responses), what kinds of data should be returned as part of the response, data item categories requested, roles requested, sort order when there are too many matches to return all hits, identifier for this request, LEXS version being used by the requestor, the date/time the request was generated, and the query itself.

The structure below shows a high level representation for the StructuredSearchRequestMessage. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

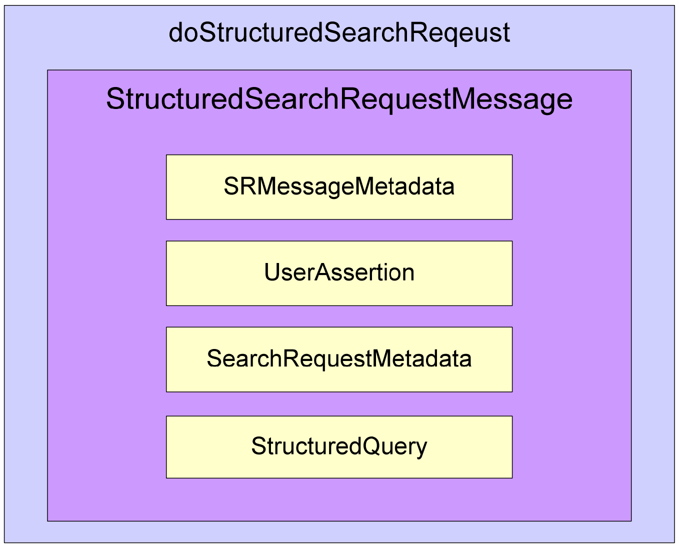


Figure 7 - High level view of doStructuredSearchRequest

SRMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination. See SRMessageMetadata in section 5.1.19 for a more detailed description.

UserAssertion [1] – Includes information about the user making the search request. See UserAssertion in section 5.1.24 for a more detailed description.

SearchRequestMetadata [1] – Includes information about the number of hits requested, what kind of data is requested, data item categories, what data owners should be queried, what Structured Payloads should be returned (for structured data responses), how long the requester will wait for a response, and an identifier for the request. Additional fields are included to support the request for additional hits. See SearchRequestMetadata in section 5.1.16 for a more detailed description.

StructuredQuery [1] *([1-n] - 3.1.1 and later)* – LEXS 3.1 SR supports a structured search capability including wildcards, numeric and date ranges, fuzzy matches, multiple values for individual fields, and the ability to limit searches to specific roles. See StructuredQuery in section 6 for a more detailed description.

4.3 doSearchResponse

doSearchResponse contains information on "hits" resulting from a doTextSearchRequest or doStructuredSearchRequest. The response may be sparsely populated and data items may contain only elements that are commonly used to help discriminate one data item from another.

Response may include information about Attachments (such as images or documents), but does not include the Attachments themselves.

This response result includes information on the LEXS version used by the responder, the date/time the response was generated, an identifier for the response, an identifier for the request that this result is in response to, the number of hits, the match score, whether the number of hits returned was restricted by a service provider limit, plus information that can be used to request additional hits.

The structure below shows a high level representation for the SearchResponseMessage. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

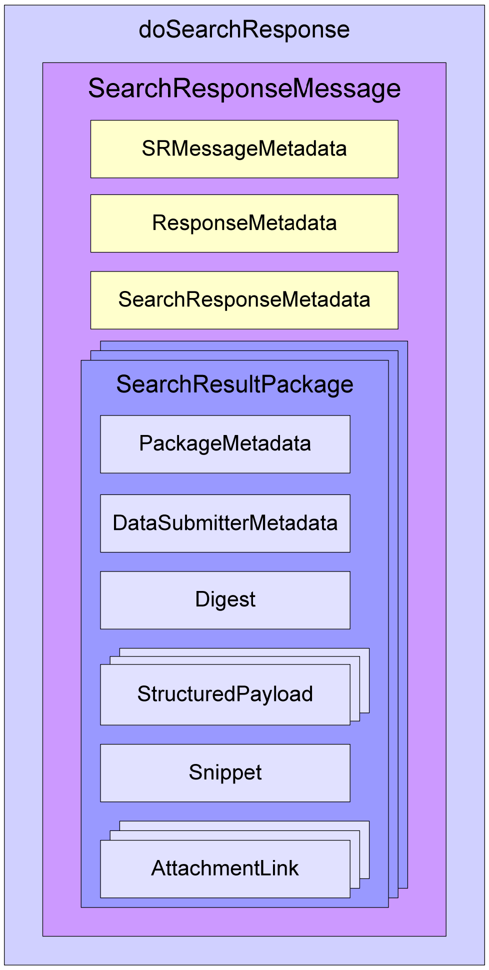


Figure 8 - High level view of doSearchResponse

SRMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination. See SRMessageMetadata in section 5.1.19 for a more detailed description.

ResponseMetadata [1] – Includes a result code, and advisory, and an identifier for the request this result is in response to. See ResponseMetadata in section 5.1.15 for a more detailed description.

SearchResponseMetadata [1] – Includes the number of hits requested, the number returned, whether service provider limits were reached for the number of hits, match score, and information that can be used to request additional hits. See SearchResponseMetadata in section 5.1.17 for a more detailed description.

SearchResultPackage [1-n] *([0-n] - 3.1.2 and later)* – Includes Package Metadata (which includes the owner of the data item), data submitter information, Digest and/or Snippet, and possibly Structured Payload(s) and information about available Attachments. See SearchResultPackage in section 5.1.18 for a more detailed description.

4.4 getDataItemRequest

A search response may be sparsely populated and data items may contain only elements that are commonly used to help discriminate one data item from another. getDataItemRequest is used to get details on a specific data item.

This request includes information on the user making the request, identifier for this request, LEXS version being used by the requestor, the date/time the request was generated, what Structured Payloads should be returned, and the data item requested.

The structure below shows a high level representation for the DataItemRequestMessage. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

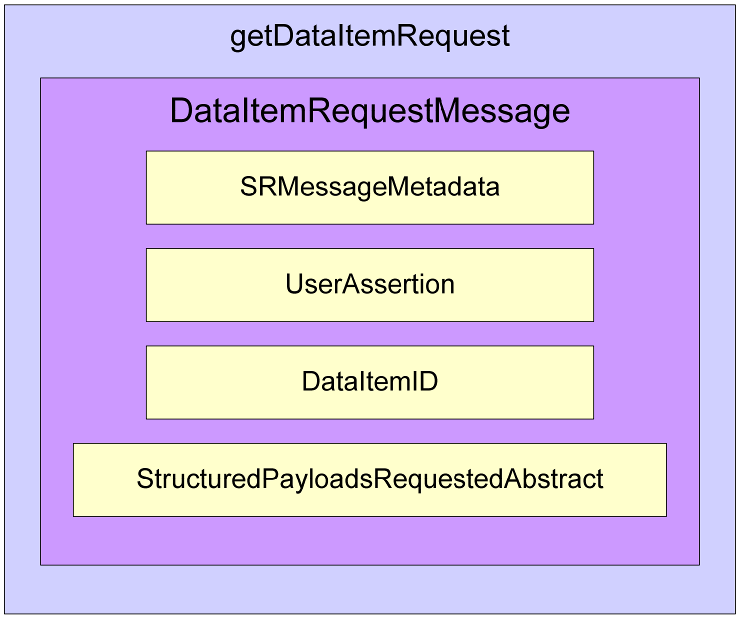


Figure 9 - High level view of getDataItemRequest

SRMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination. See SRMessageMetadata in section 5.1.19 for a more detailed description.

UserAssertion [1] – Includes information about the user making the request. See UserAssertion in section 5.1.24 for a more detailed description.

DataItemID [1] – string. The value of a DataItemID element provided in a previous result.

StructuredPayloadsRequestedAbstract [1] – abstract element. The substitutable elements indicate whether the requestor wants responses to include any available Structured Payloads, no Structured Payloads, or specific Structured Payloads. See StructuredPayloadsRequestedAbstract in section 5.1.22 for a more detailed description.

4.5 getDataItemResponse

getDataItemResponse contains all available details and contextual information associated with the requested data items. The response may include information about Attachments (such as images or documents), but does not include the Attachments themselves.

This response includes information on the LEXS version used by the responder, the date/time the response was generated, an identifier for the response, and an identifier for the request that this result is in response to.

The structure below shows a high level representation for the DataItemResponseMessage. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

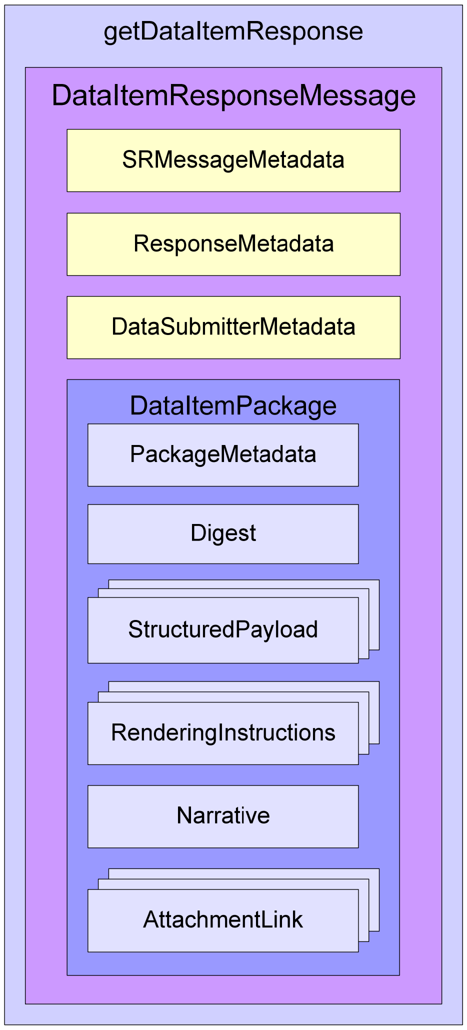


Figure 10 - High level view of getDataItemResponse

SRMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination. See SRMessageMetadata in section 5.1.19 for a more detailed description.

ResponseMetadata [1] – Includes a result code, and advisory, and an identifier for the request this result is in response to. See ResponseMetadata in section 5.1.15 for a more detailed description.

DataSubmitterMetadata [1] – Includes the organization, the system name, and possibly the contact information for the data submitter. See MessageOriginMetadata and DataSubmitterMetadata in section 5.1.10 for a more detailed description.

DataItemPackage [1] *([0-1] – 3.1.4 and later)* – Includes Package Metadata (which includes the owner of the data item), Digest, and possibly Structured Payload(s) and information about available Attachments. See DataItemPackage in section 5.1.4 for a more detailed description.

4.6 getAttachmentRequest

The responses to searches and get data item requests may include information about Attachments, but in the interest of limiting the sizes of these results do not include the Attachments themselves. getAttachmentRequest allows a consumer to request the Attachments. Multiple Attachments may be specified in a single request.

This request includes information on the user making the request, identifier for this request, LEXS version being used by the requestor, the date/time the request was generated, and the Attachment(s) requested.

The structure below shows a high level representation for the AttachmentRequestMessage. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

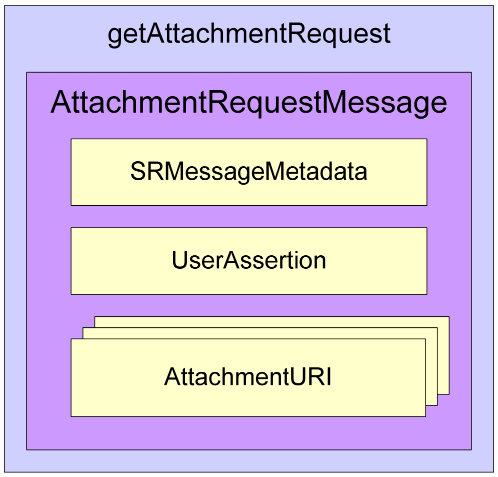


Figure 11 - High level view of getAttachmentRequest

SRMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination. See SRMessageMetadata in section 5.1.19 for a more detailed description.

UserAssertion [1] – Includes information about the user making the request. See UserAssertion in section 5.1.24 for a more detailed description.

AttachmentURI [1-n] – string. The value of the AttachmentURI element provided in a previous result. The AttachmentURI uniquely identifies the Attachment and must follow all rules for an URI (Uniform Resource Identifier). Note that if the URI is provided as a URL, it does not imply that the attachment is available over the Internet at that address, similar to the way URLs in XML namespaces do not imply that the address is resolvable.

4.7 getAttachmentResponse

getAttachmentResponse is returned by a service provider in response to a getAttachmentRequest, and may include multiple Attachments.

This response includes information on the LEXS version used by the responder, the date/time the response was generated, an identifier for the response, an identifier for the request that this result is in response to, and one or more Attachments.

The structure below shows a high level representation for the AttachmentResponseMessage. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

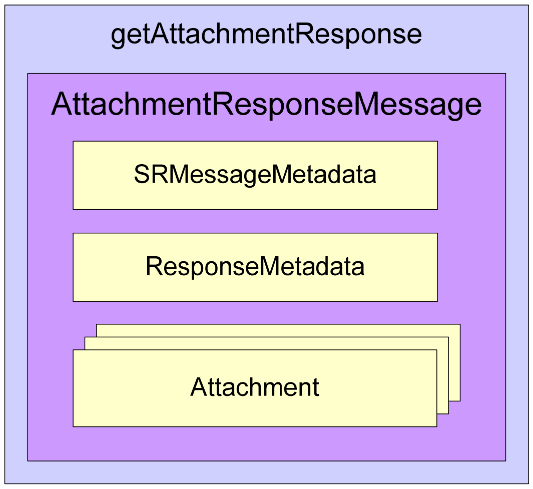


Figure 12 - High level view of getAttachmentResponse

SRMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination. See SRMessageMetadata in section 5.1.19 for a more detailed description.

ResponseMetadata [1] – Includes a result code, and advisory, and an identifier for the request this result is in response to. See ResponseMetadata in section 5.1.15 for a more detailed description.

Attachment [1-n] *([0-n] –3.1.4 and later)* – see Attachment in section 5.1.1 for a more detailed description. This includes the AttachmentURI, the Attachment binary, domain specific attributes.

4.8 getCapabilitiesRequest

getCapabilitiesRequest allows a consumer to ask about the capabilities of a service provider, including whether it supports text and/or structured searches and the features of each, whether paging is supported, and what data item categories are available.

This request includes an identifier for this request, LEXS version being used by the requestor, and the date/time the request was generated.

The structure below shows a high level representation for the ServiceProviderRequestMessage which is used in the getCapabilitiesRequest as well as the getDataOwnersRequest and getAvailabilityRequest. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

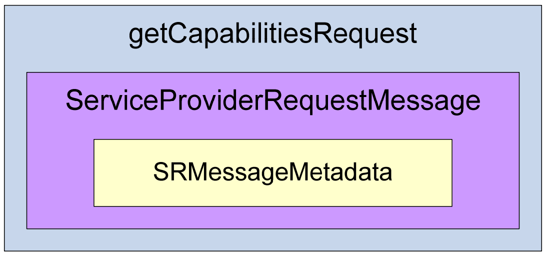


Figure 13 - High level view of getCapabilitiesRequest

SRMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination. See SRMessageMetadata in section 5.1.19 for a more detailed description.

4.9 getCapabilitiesResponse

getCapabilitiesResponse is returned by a service provider in response to a getCapabilitiesRequest. Some capabilities apply at the service provider level, and some only apply to structured or unstructured searches.

Capabilities Common to Structured and Text Searches
At the service provider level, the result indicates the maximum number of hits the service provider can return in a single response, and whether the service provider supports structured searches and/or unstructured searches. Optionally, the service provider can also report the data item categories that are supported by the service provider as a whole.

Unstructured (text) Search Capabilities
For text searches, the result indicates whether the service provider supports interface control in terms of allowing users or applications to request additional matches for text searches, whether searches can be performed on exact phrases, whether the logical operators AND and OR are supported, and whether wildcards are supported.

Structured Search Capabilities
For text searches, the result indicates whether the service provider supports interface control in terms of allowing users or applications to request additional matches for text searches, multiple values for individual fields, fuzzy searches, date and numeric ranges, and whether wildcards are supported.

The structure below shows a high level representation for the CapabilitiesResponseMessage. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

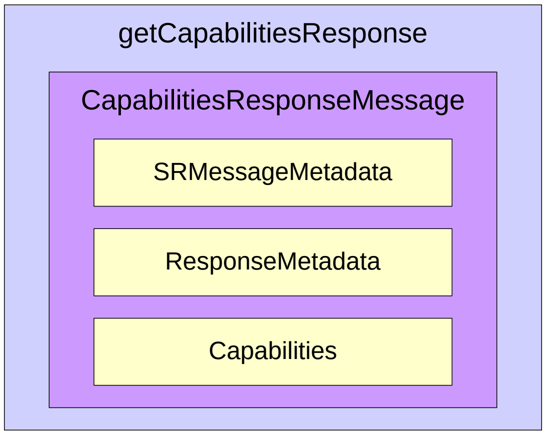


Figure 14 - High level view of getCapabilitiesResponse

SRMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination. See SRMessageMetadata in section 5.1.19 for a more detailed description.

ResponseMetadata [1] –Includes a result code, and advisory, and an identifier for the request this result is in response to. See ResponseMetadata in section 5.1.15 for a more detailed description.

Capabilities [1] – container for capabilities information.

The XML fragment below shows the organization of the Capabilities element.

```
<Capabilities>
        <DataItemCategory/>
        <PagingIndicator/>
        <MaxHitsReturned/>
        <StructuredSearch>
                <MultipleValueIndicator/>
                <WildcardIndicator/>
                <FuzzySearchIndicator/>
                <DateRangeIndicator/>
                <NumericRangeIndicator/>
        </StructuredSearch>
        <TextSearch>
                <ExactPhraseIndicator/>
                <LogicalOperatorIndicator/>
                <WildcardIndicator/>
        </TextSearch>
        <DomainAttribute/>
</Capabilities>
```

DataItemCategory [0-n] –Provides optional list of data item categories supported by the service provider. See DataItemCategory in section 5.1.3 for a more detailed description.

PagingIndicator [1] – Boolean. Indicates whether the service provider supports requests for additional hits from a search. See Appendix D for more information about how this capability works.

MaxHitsReturned [1] – positive integer. Indicates the maximum number of hits that the service provider can return in a response.

StructuredSearch [0-1] – container for capabilities specific to structured searches. Note that if the StructuredSearch element exists, then the service provider supports structured searches.

MultipleValueIndicator [1] – Boolean. Indicates whether this service provider allows for multiple values in individual elements in a structured search. For example, a search on a person with last name “Smith” or “Jones”.

WildcardIndicator [1] – Boolean. Indicates whether the service provider allows wildcard character(s) in structured searches. This is not handled at the service provider since there may be service providers that support wildcards for one type of search and not the other.

FuzzySearchIndicator [1] – Boolean. Indicates whether the service provider supports some form of fuzzy search, such as soundex or metaphone. Note that service providers are not limited to just soundex or metaphone.

DateRangeIndicator [1] – Boolean. Indicates whether the service provider supports queries on ranges for date elements, such as a range on a date of birth. This also includes “open-ended” ranges, such as a search on a data of birth before a specified date. Note that a service provider may support range searches on some dates and not others; this Boolean does not indicate what elements, just that the service provider supports some level of date range searches.

NumericRangeIndicator [1] – Boolean. Indicates whether the service provider supports queries on ranges for numeric elements, such as a range on a boat length. This also includes “open-ended” ranges, such as a search on a boat length less than a specified value. Note that a service provider may support range searches on some numbers and not others; this Boolean does not indicate what elements, just that the service provider supports some level of numeric range searches.

TextSearch [0-1] – container for capabilities specific to text searches. Note that if the TextSearch element exists, then the service provider supports text searches.

ExactPhraseIndicator [1] – Boolean. Indicates whether the service provider allows exact phrases in text searches.

LogicalOperatorIndicator [1] – Boolean. Indicates whether the service provider allows the use of logical AND and OR in text searches.

WildcardIndicator [1] – Boolean. Indicates whether the service provider allows wildcard character(s) in text searches. This is not handled at the service provider since there may be service providers that support wildcards for one type of search and not the other.

DomainAttribute [0-n] – Contains element names, element values, and XML blocks that are useful to specific service providers or consumers but that are not part of the LEXS specification. See DomainAttribute in section 5.1.9 for a more detailed description.

4.10 getDataOwnersRequest

This request allows a consumer to ask for the list of data owners available from a service provider, including whether each data owner supports structured searches, unstructured searches or both. Optionally, information about supported data item categories may be supplied for each data owner.

This request includes an identifier for this request, LEXS version being used by the requestor, and the date/time the request was generated.

The structure below shows a high level representation for the ServiceProviderRequestMessage which is used in the getDataOwnersRequest as well as the getCapabilitiesRequest and getAvailabilityRequest. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

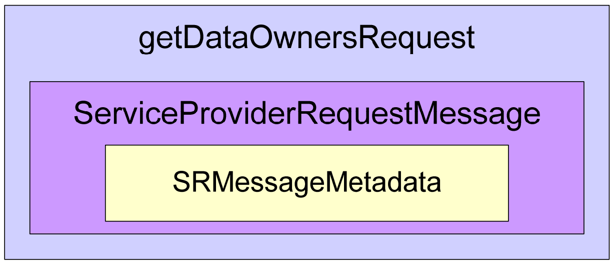


Figure 15 - High level view of getDataOwnersRequest

SRMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination. See SRMessageMetadata in section 5.1.19 for a more detailed description.

4.11 getDataOwnersResponse

getDataOwnersResponse is returned by a service provider in response to a getDataOwnersRequest.

This response includes information on the LEXS version used by the responder, the date/time the response was generated, an identifier for the response, and an identifier for the request that this result is in response to.

The structure below shows a high level representation for the DataOwnersResponseMessage. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

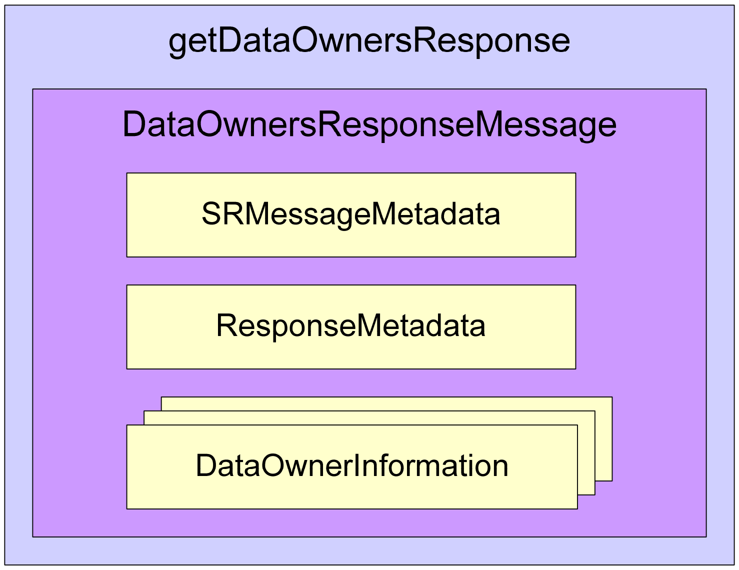


Figure 16 - High level view of getDataOwnersResponse

SRMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination. See SRMessageMetadata in section 5.1.19 for a more detailed description.

ResponseMetadata [1] – Includes a result code, and advisory, and an identifier for the request this result is in response to. See ResponseMetadata in section 5.1.15 for a more detailed description.

DataOwnerInformation [1-n] – container for information about each data owner.

The XML fragment below shows the organization of the DataOwnerInformation element.

```
<DataOwnerInformation>
        <DataOwnerMetadata/>
        <StructuredSearchIndicator/>
        <TextSearchIndicator/>
        <DataItemCategory/>
        <DomainAttribute/>
</DataOwnerInformation>
```

DataOwnerMetadata [1] – Includes the agency, the system name, and contact information for the data owner. See DataOwnerMetadata in section 5.1.7 for a more detailed description.

StructuredSearchIndicator [1] – Boolean. Indicates whether the data owner includes structured data and can be searched using doStructureSearchRequest.

TextSearchIndicator [1] – Boolean. Indicates whether the data owner includes unstructured data and can be searched using doTextSearchRequest.

DataItemCategory [0-n] – Provides optional list of data item categories supported by the data owner. See DataItemCategory in section 5.1.3 for a more detailed description.

DomainAttribute [0-n] – Contains element names, element values, and XML blocks that are useful to specific service providers or consumers but that are not part of the LEXS specification. See DomainAttribute in section 5.1.9 for a more detailed description.

4.12 getAvailabilityRequest

getAvailabilityRequest allows a consumer to request an indication as to whether the service provider is available.

This request includes an identifier for this request, LEXS version being used by the requestor, and the date/time the request was generated.

The structure below shows a high level representation for the ServiceProviderRequestMessage which is used in the getAvailabilityRequest as well as the getDataOwnersRequest and getCapabilitiesRequest. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

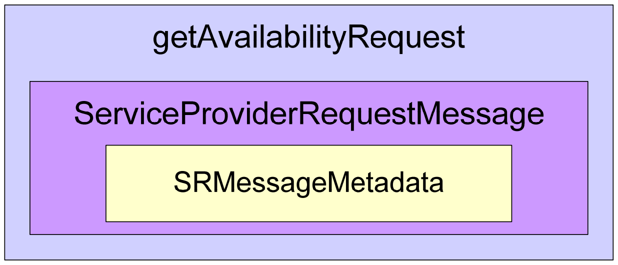


Figure 17 - High level view of getAvailabilityRequest

SRMessageMetadata [1] – see SRMessageMetadata in section 5.1. This includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination.

4.13 getAvailabilityResponse

getAvailabilityResponse is returned by a service provider in response to a getAvailabilityRequest.

This response includes information on the LEXS version used by the responder, the date/time the response was generated, an identifier for the response, and an identifier for the request that this result is in response to.

The structure below shows a high level representation for the AvailabilityResponseMessage. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

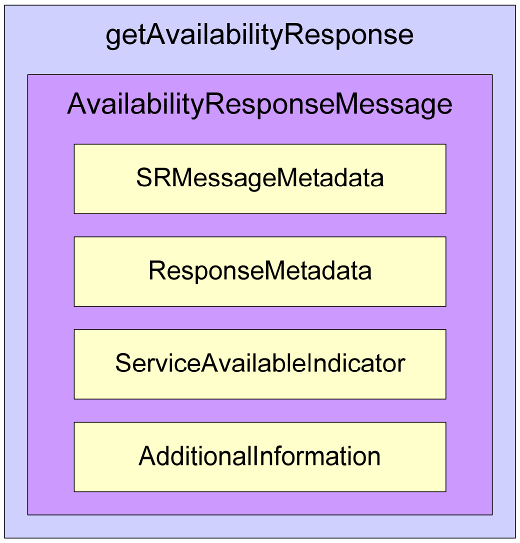


Figure 18 - High level view of getAvailabilityResponse

SRMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination. See SRMessageMetadata in section 5.1.19 for a more detailed description.

ResponseMetadata [1] – Includes a result code, and advisory, and an identifier for the request this result is in response to. See ResponseMetadata in section 5.1.15 for a more detailed description.

ServiceAvailableIndicator [1] – Boolean. True if the service provider is available. False if there are problems, such as the database is down for maintenance. Note that if there is no reply to the request, it may indicate that the service provider is down, that there are network problems, that the requestor is not properly configured, etc.

AdditionalInformation [0-1] – string. This element may be populated by the service provider when the ServiceAvailable is false to provide additional information; for example that the database is down for maintenance and is scheduled to be back up by 4PM EST. Note that there may be cases where a service provider does not store data from a data owner, but accesses a data owner in order to service requests. In this case, the service provider could be up but one or more data owners “behind” it are down. In this case, the service provider would report that it is available, and utilize this additional information element to document what sources are down.

4.14 getDomainRequest (3.1.4 and later)

getDomainRequest allows a consumer to submit a domain specific request. This is a non-normative request message and could be used for out-of-band, non-LEXS conformant information exchanges between systems. Use of this message by no means classifies any system as LEXS conformant and should be reserved for functionality not supported by LEXS.

This request includes an identifier for this request, LEXS version being used by the requestor, and the date/time the request was generated.

The structure below shows a high level representation for the DomainRequestMessage which is used in the getDomainRequest. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

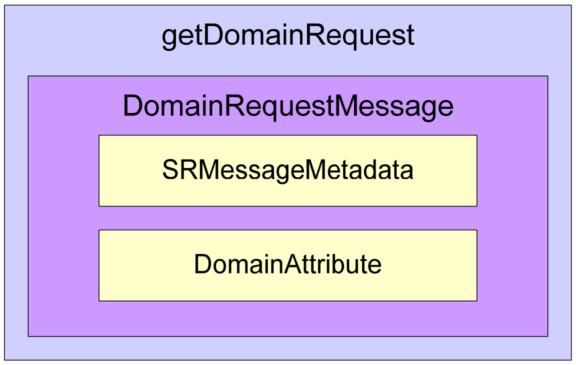


Figure 19 - High level view of getDomainRequest

SRMessageMetadata [1] – see SRMessageMetadata in section 5.1. This includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination.

DomainAttribute [1] – Contains element names, element values, and XML blocks that are not part of the LEXS specification. See DomainAttribute in section 5.1.9 for a more detailed description.

4.15 getDomainResponse (3.1.4 and later)

getDomainResponse is returned by a service provider in response to a getDomainRequest.

This response includes information on the LEXS version used by the responder, the date/time the response was generated, an identifier for the response, and an identifier for the request that this result is in response to.

The structure below shows a high level representation for the DomainResponseMessage. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

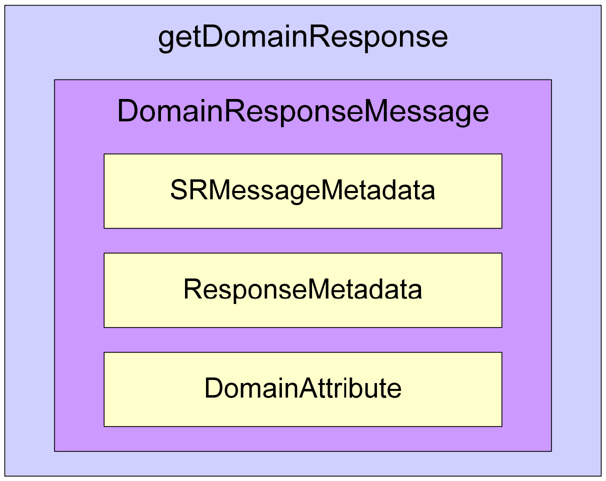


Figure 20 - High level view of getDomainResponse

SRMessageMetadata [1] – Includes the LEXS version utilized by the requesting system, message date and time, an identifier for the message, the message origin, and message destination. See SRMessageMetadata in section 5.1.19 for a more detailed description.

ResponseMetadata [1] – Includes a result code, and advisory, and an identifier for the request this result is in response to. See ResponseMetadata in section 5.1.15 for a more detailed description.

DomainAttribute [1] – Contains element names, element values, and XML blocks that are not part of the LEXS specification. See DomainAttribute in section 5.1.9 for a more detailed description.

5 Supporting Structures

The messages involved in the Publish operation and the Search and Retrieve operations described above are supported by numerous LEXS and NIEM structures. These structures, as well as LEXS referencing mechanisms, are addressed in this chapter. Since all structures described below support numerous messages, this chapter is organized to serve as a reference guide and the structures are presented alphabetically within each section.

5.1 LEXS Structures

5.1.1 Attachment

The Attachment element describes details about an Attachment. The structure uniquely identifies an Attachment by an AttachmentURI, holds an attachment in binary format, and allows for domain specific attributes.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<Attachment>
        <AttachmentURI/>
        <Binary>
                <BinaryID/>
                <BinaryObject/>
                <BinaryCaptureDate/>
                <BinaryDescriptionText/>
                <BinaryFormatID/>
                <BinaryFormatStandardName/>
                <BinarySizeValue/>
                <BinaryCategoryText/>
        </Binary>
        <DataOwnerMetadata/> (3.1.3 and later)
        <DomainAttribute/>
</Attachment>
```

AttachmentURI [1] – string. The value of the AttachmentURI element provided in a previous result. The AttachmentURI uniquely identifies the Attachment and must follow all rules for an URI (Uniform Resource Identifier). Note that if the URI is provided as a URL, it does not imply that the attachment is available over the Internet at that address, similar to the way URLs in XML namespaces do not imply that the address is resolvable.

Binary [1] – NIEM nc:Binary. Attachment object in binary format.

BinaryID [0-1] – string. An identifier that references the binary object.

BinaryObject [0-1] – abstract element. Can be substituted with BinaryBase64Object which specifies that the binary data uses base64 encoding.

BinaryCaptureDate [0-1] – date. The date on which the binary object was captured or created.

BinaryDescriptionText [0-1] – string. The description of the binary object.

BinaryFormatID [0-1] – string. An identifier for the file format or content type of the binary object.

BinaryFormatStandardName [0-1] – string. The name of the standard or protocol used to classify binary content.

BinarySizeValue [0-1] – non-negative decimal. The size of the binary object in kilobytes.

BinaryCategoryText [0-1] – string. The kind of object that has been encoded.

DataOwnerMetadata [0-1] – *(3.1.3 and later)* Includes the agency, the system name, and contact information for the owning agency and system. See DataOwnerMetadata in section 5.1.7 for a more detailed description.

DomainAttribute [0-n] – Contains element names, element values, and XML blocks that are useful to specific service providers or consumers but that are not part of the LEXS specification. See DomainAttribute in section 5.1.9.

5.1.2 AttachmentLink

The AttachmentLink element serves as an anchor in the Data Item Package for connecting Digest and/or Structured Payload entities to Attachments. The structure describes the Attachment and uniquely identifies it by an AttachmentURI.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<AttachmentLink>
        <AttachmentURI/>
        <AttachmentViewableIndicator/>
        <BinaryDescriptionText/>
        <BinarySizeValue/>
        <BinaryCategoryText/>
</AttachmentLink>
```

AttachmentURI [1] – string. The value of the AttachmentURI element provided in a previous result. The AttachmentURI uniquely identifies the Attachment and must follow all rules for an URI (Uniform Resource Identifier). Note that if the URI is provided as a URL, it does not imply that the attachment is available over the Internet at that address, similar to the way URLs in XML namespaces do not imply that the address is resolvable.

AttachmentViewableIndicator [1] – boolean. A flag which indicates whether an Attachment can be displayed (true, such as for a mug shot) or not (false, such as for a stylesheet used for rendering).

BinaryDescriptionText [1] – string. Description of the Attachment.

BinarySizeValue [0-1] – non-negative decimal. The size of a Attachment binary in kilobytes.

BinaryCategoryText [0-1] – string. The type of Attachment, for example a mug shot, driver license picture, audio confession.

5.1.3 DataItemCategory

DataItemCategory provides information about the type of event or data type. This may be used by service providers to indicate what types of events or data are incorporated in the service provider and/or its data owners, or in an individual data item. Groups may work together to define an appropriate list of categories that can be used in searches among the groups to allow users to limit search results. This element could also be used by user interfaces for sorting purposes.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<DataItemCategory>
        <DataItemCategoryText/>
        <DataItemCategoryDescription/>
</DataItemCategory>
```

DataItemCategoryText [1] – string. Indicates the type of event or data type. This should be a user-friendly and reasonably intuitive string that can be used in a list or as a check-box label in a GUI. For example, this element might contain the value “Incident Report”, or “Call Log”.

DataItemCategoryDescription [0-1] – string. This element provides a longer description which can be used to better explain the meaning of the DataItemCategoryText value provided above, and could be multiple sentences. For a relatively well understood DataItemCategoryText value, such as “Incident Report”, no additional description may be deemed necessary. However, a description may be provided if desired; for example, if the DataItemCategoryText value is “Call Log”, a description might be provided such as “Inmate call log from the ITS system”.

5.1.4 DataItemPackage

DataItemPackage contains Package Metadata, a Digest, and possibly one or more Structured Payloads, a Narrative, Rendering Instructions, and Attachment Link elements. DataItemPackage is contained in a PublishMessage as part of a doPublish request, and in a DataItemResponseMessage as part of a getDataItemResponse.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<DataItemPackage>
        <PackageMetadata/>
        <Digest/>
        <StructuredPayload/>
        <RenderingInstructions/>
        <Narrative/>
        <AttachmentLink/>
</DataItemPackage>
```

PackageMetadata [1] – Contains metadata about LEXS Package. See PackageMetadata in section 5.1.12 for a more detailed description.

Digest [1] – Contains LEXS Entities and Associations between Entities. See Digest in section 5.1.8 for a more detailed description.

StructuredPayload [0-n] – Contains the Structured Payload. See StructuredPayload in section 5.1.20 for a more detailed description.

RenderingInstructions [0-n] – Contains Rendering Instructions. See RenderingInstructions in section 5.1.14 for a more detailed description.

Narrative [0-1] – string. Unstructured (text) content for the package.

AttachmentLink [0-n] – Contains information about a link to an Attachment. See AttachmentLink in section 5.1.2 for a more detailed description.

5.1.5 DataOwnerContact / SystemContact / DataItemContact

DataOwnerContact, SystemContact, and DataItemContact share a common structure which includes contact information for the data owner or a system or a data item, and includes a person and organization to contact and their phone number and email address.

The structure below shows a representation for this element. Descriptions for the sub-elements are also provided, along with cardinality which is shown in brackets. DataOwnerContact is shown below; however, the structure and element names included in SystemContact and DataItemContact are identical.

```
<DataOwnerContact>
        <PersonGivenName/>
        <PersonMiddleName/>
        <PersonSurName/>
        <PersonFullName/>
        <ContactMeans/>
        <OrganizationName/>
        <DomainAttribute/> (3.1.3 and later)
</DataOwnerContact>
```

PersonGivenName [0-1] – string. First name for the person to contact regarding systems or individual data items.

PersonMiddleName [0-1] – string. Middle name for the person to contact regarding systems or individual data items.

PersonSurName [1] – string. Last name for the person to contact regarding systems or individual data items.

PersonFullName [0-1] – string. Full name for the person to contact regarding systems or individual data items.

ContactMeans [0-n] – abstract element. Can be substituted with the ContactEmailID (as string), or telephone numbers (of TelephoneNumberType) ContactTelephoneNumber, ContactFaxNumber, ContactMobileTelephoneNumber or ContactPagerNumber.

OrganizationName [0-1] – string. Organization name for the person to contact regarding systems or individual data items.

DomainAttribute [0-n] – *(3.1.3 and later)* Contains element names, element values, and XML blocks that are useful to specific service providers or consumers but that are not part of the LEXS specification. See DomainAttribute in section 5.1.9.

5.1.6 DataOwnerIdentifier

DataOwnerIdentifier uniquely identifies an agency and a system in an agency that owns data. This identifier is used in DataOwnerMetadata that is part of PackageMetadata, and in DataOwnerInformation that is part of DataOwnersResponseMessage.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<DataOwnerIdentifier>
        <AgencyIDAbstract/>
        <OrganizationName/>
        <SystemID/>
</DataOwnerIdentifier/>
```

AgencyIDAbstract [1] – abstract element. Can be substituted with ORI or OriginatingAgencyID, both of which are text elements. Specifies an agency identifier such as an ORI code, for Law Enforcement agencies that use ORI, or some other agency identification code where ORI is not appropriate.

OrganizationName [1] – string. Specifies the name of the agency that owns the data.

SystemID [1] – string. Specifies the agency system where the data resides. An ID or well known acronym for the system should be supplied rather than spelling out the system.

5.1.7 DataOwnerMetadata

DataOwnerMetadata provides information about the owner of the data, including the agency, the system in the agency where the data resides, and contact information for the data owner. This metadata is included in PackageMetadata and Attachment. The PackageMetadata is in turn used in PublishMessage in doPublish, and in SearchResultPackage in doSearchResponse, and in DataItemResponseMessage in getDataItemResponse.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<DataOwnerMetadata>
        <DataOwnerIdentifier/>
        <DataOwnerContact/>
        <DomainAttribute/>
</DataOwnerMetadata>
```

DataOwnerIdentifier [1] – Uniquely identifies the owning agency and system in the agency where the data resides. See DataOwnerIdentifier in section 5.1.6 for a more detailed description.

DataOwnerContact [0-1] – Provides contact person, contact organization, telephone number, and email address for the agency and system that owns the data. See DataOwnerContact/SystemContact/DataItemContact in section 5.1.5 for a more detailed description.

DomainAttribute [0-n] – Contains element names, element values, and XML blocks that are useful to specific service providers or consumers but that are not part of the LEXS specification. See DomainAttribute in section 5.1.9 for a more detailed description.

5.1.8 Digest

Digest contains LEXS entities and associations between entities. The Digest contains the information that provides the base level of understanding in LEXS. The Digest is included in the DataItemPackage which is contained in the PublishMessage for doPublish and in the DataItemResponseMessage for getDataItemResponse. The Digest is also included in the SearchResultPackage contained in the SearchResponseMessage for doSearchResponse.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<Digest>
        <Entity/>
        <Associations/>
</Digest>
```

Entity [1-n] – abstract element used to represent high level, real-world objects in LEXS. Can be replaced with any LEXS Entity. If there are multiple entities they can appear in any order. LEXS entities include:

EntityActivity	EntityIntangibleItem *(3.1.1 and later)*
EntityAircraft	EntityLocation
EntityCreditCard *(3.1.4 and later)*	EntityNetworkAddress *(3.1.4 and later)*
EntityDocument *(3.1.4 and later)*	EntityOrganization
EntityDrug	EntityPerson
EntityEmail	EntitySubstance
EntityExplosive	EntityTangibleItem
EntityFirearm	EntityTelephoneNumber
EntityHash *(3.1.4 and later)*	EntityVehicle
EntityInstantMessenger *(3.1.4 and later)*	EntityVessel

See Entity Examples for entity examples.

Associations [0-n] – container. Contains all associations for the data item described by the Digest. Associations can relate entities or between entities and attachments. If there are multiple associations, they must appear in the order defined by the Digest schema. LEXS associations are categorized by entity in Appendix B.

5.1.9 DomainAttribute

DomainAttribute provides an extension point for element names, element values, and XML blocks that are useful to specific service providers or consumers but that are not part of the LEXS specification. This element can be used to capture data such as region information, system id, and purpose code of the query that were addressed by explicit elements in earlier versions of LEXS, but that are not included in LEXS 3.1 since they are focused on specific systems.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<DomainAttribute>
      <AttributeName/>
      <AttributeValue/>
      <Domain/>
      <any/>
<DomainAttribute>
```

AttributeName [0-1] – Specifies the name of domain attribute, for example, RegionInfo.

AttributeValue [0-1] – Specifies the value of domain attribute, for example, SEA.

Domain [1] – Specifies the domain to which this block applies, for example, OneDOJ.

any [0-1] – xsd:any. Allows the inclusion of XML data for complex domain-specific content that can not be represented using the AttributeName/AttributeValue construct. (For usage of xsd:any see W3C XML Schema Primer Section 5.5 "Any Element, Any Attribute" at http://www.w3.org/TR/xmlschema-0/#any.)

5.1.10 MessageOriginMetadata and DataSubmitterMetadata

These elements uniquely identify an organization and a system and include contact information for the system.

The structure below shows a representation for this element. Descriptions for the sub-elements are also provided, along with cardinality which is shown in brackets. MessageOriginMedata is shown below; however the structure and element names included in DataSubmitterMetadata are identical.

```
<MessageOriginMetadata>
        <SystemIdentifier>
                <AgencyIDAbstract/>
                <OrganizationName/>
                <SystemID/>
        </SystemIdentifier>
        <SystemContact/>
        <DomainAttribute/>
</MessageOriginMetadata>
```

SystemIdentifier [1] – container. Includes information specifying an organization name, an organization identifier, and a system identification element.

AgencyIDAbstract [0-1] – abstract element. Can be substituted with ORI or OriginatingAgencyID, both of which are text elements. Specifies an agency identifier such as an

ORI code, for Law Enforcement agencies that use ORI, or some other agency identification code where ORI is not appropriate.

OrganizationName [1] – string. Specifies the organization name for the system where the message is being sent.

SystemID [1] – string. Specifies the system where the message originating or being submitted from. An ID or well known acronym for the system should be supplied rather than spelling out the system.

SystemContact [1] – Provides contact person, contact organization, telephone number, and email address for the system originating or submitting the message. See DataOwnerContact/SystemContact/DataItemContact in section 5.1.5 for a more detailed description.

DomainAttribute [0-n] – Contains element names, element values, and XML blocks that are useful to specific service providers or consumers but that are not part of the LEXS specification. See DomainAttribute in section 5.1.9 for a more detailed description.

5.1.11 NumberOfStructuredPayloadTerms

This element is part of the SearchRequestMetadata and provides a count of search terms (i.e. the elements or attributes being searched for) in an included Structured Payload. The purpose of this element is to give service providers that do not understand one or more Structured Payloads the information necessary to calculate the match score in the response. The StructuredQueryMatchScore (see section 5.1.17) represents the percentage of the search elements that were matched in response to a structured search query. In order to calculate an accurate match score the service provider needs to have a count of all terms including those that were ignored because they applied to a Structured Payload that the service provider did not understand. There is a NumberOfStructurePayloads element for each Structured Payload included in the query.

For example, assume there is a query with a Digest and two Structured Payloads for community A and community B. The query might be sent to a service provider that understands A only, or B only, both A and B, or neither A nor B. In order for a service provider that does not understand both A and B to calculate an accurate match score, the number of terms in each Structured Payload is required. Assume that the Digest has 4 terms, A has 3, and B has 1. If the service provider understands A and not B, the service provider still needs to know there are 8 total terms in order to calculate the match score. Since the service provider can process the Digest and Structured Payload A, it calculates that the Digest has 4 terms and A has 3 terms, and uses the NumberOfStructuredPayloadTerms for B to determine that the total number of terms requested is 8.

Note that each search term is a single element or attribute that describes a characteristic that distinguishes the object of interest from other objects. For example, if the search is against a Structured Payload person to find someone with a 10 inch scar, the person entity provides the context for the search, the element that indicates the feature is a scar is one search term, the element that indicates it has length 10 is a search term, and the element that indicates the unit of measure is inches is a separate search term; three search terms in total. If a unit of measure uses an attribute rather than an element, it still counts as a search term.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<NumberOfStructuredPayloadTerms>
        <CommunityURI/>
        <NumberOfTerms/>
</NumberOfStructuredPayloadTerms>
```

CommunityURI [1] – string. The URI that uniquely identifies the community for the Structured Payload as provided in the Structured Payload Metadata.

NumberOfTerms [1] – positive integer. The number of search terms in the Structured Payload.

5.1.12 PackageMetadata

PackageMetadata provides information about the data item contained in the package.
The PackageMetadata is used to describe the DataItemPackage which is contained in the PublishMessage for doPublish and in the DataItemResponseMessage for getDataItemResponse. It also describes the SearchResultPackage contained in the SearchResponseMessage for doSearchResponse.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

Prior to LEXS 3.1.4

```
        <PackageMetadata>
                <DataItemID/>
                <DataItemCompleteIndicator/> (3.1.1 and later)
                <DataItemContact/>
                <DataItemDate/>
                <DataItemReferenceID/>
                <DataItemPublishInstruction/>
                <DataItemStatus/>
                <DataOwnerMetadata/>
                <DisseminationCriteria/>
                <DataItemCategory/>
                <DomainAttribute/>
        </PackageMetadata>
```

LEXS 3.1.4 and later

```
<PackageMetadata>
        <DataItemID/>
        <DataItemCompleteIndicator/> (3.1.1 and later)
        <DataItemContact/>
        <DataItemDate/>
        <DataItemLEXSVersion> (3.1.4 and later)
        <DataItemReferenceID/>
        <DataItemPublishInstruction/>
        <DataItemStatus/>
        <DataOwnerMetadata/>
        <DisseminationCriteriaAbstract/> (3.1.4 and later*)
        <DataItemCategory/>
        <DomainAttribute/>
</PackageMetadata>
```

DataItemID [1] – string. Unique identifier for the data item. DataItemID is unique for a given service provider.

DataItemCompleteIndicator [0-1] – boolean. *(3.1.1 and later)* A flag which indicates whether the package includes all available information (true) or whether more is available (false). If the indicator does not exist, it does not imply either complete or incomplete; merely that the data owner either did not supply the indicator or does not want to indicate completeness for this data item.

DataItemContact [0-n] – Provides contact person, contact organization, telephone number, and email address for the data item. See DataOwnerContact/SystemContact/DataItemContact in section 5.1.5 for a more detailed description.

DataItemDate [1] – date. Date the data item was produced/generated.

DataItemLEXSVersion [0-1] – string. *(3.1.4 and later)* Specifies the LEXS version used within the Data Item. This version could differ from the LEXSVersion element used in MessageMetadata, for example if a LEXS 3.1.4 message includes a LEXS 3.1.1 Data Item.

DataItemReferenceID [1] – string. A human readable unique identifier that is meaningful to the owning system. May be the same as the Data Item ID

DataItemPublishInstruction [0-1] – value from an enumerated code list. Indicates the disposition of the data item. The values and their meanings for dissemination criteria are shown below.

- Insert – Package content replaces all content that corresponds to this DataItemID. If the data item is not already in the system it must be added.
- Delete – Content corresponding to this DataItemID must be deleted. If the DataItemID does not exist it is a protocol logic error.

DataItemStatus [1] – string. Free text field that shows the status of the data item such as Valid/Invalid, etc.

DataOwnerMetadata [1] – Includes the agency, the system name, and contact information for the owning agency and system. See DataOwnerMetadata in section 5.1.7 for a more detailed description.

DisseminationCriteriaAbstract [1] – abstract element *(3.1.4 and later).* The element is a placeholder for substitutable elements DisseminationCriteria or DisseminationCriteriaValue, which are provided by the data owner to indicate who is allowed to see the information. For publish it provides information to the service provider about what can be returned as the result of a search. For search or get results, it provides information to the user so they know the criteria for sharing this information. Note that this is different from the data sensitivity included at the message level. Substitutable elements consist of DisseminationCriteria and DisseminationCriteriaValue, both described below.

DisseminationCriteria *(mandatory prior to 3.1.4, substitution for an abstract DisseminationCriteriaAbstract element in 3.1.4 and later)* – value from an enumerated code list. This information is provided by the data owner to indicate who is allowed to see the information. For publish it provides information to the service provider about what can be returned as the result of a search. For search or get results, it provides information to the user so they know the criteria for sharing this information. Note that this is different from the data sensitivity included at the message level. The values and their meanings for dissemination criteria are shown below.

- White – all users are allowed access
- Gray – the data owner has full access; all other users are provided pointer (i.e. who to contact and how) information only
- Black – the data owner has full access; all other users receive no information and no indication that there was a hit when performing searches

DisseminationCriteriaValue *(substitution for an abstract DisseminationCriteriaAbstract element in 3.1.4 and later)* – complex element. Provides a mechanism for domains to define dissemination criteria when the enumerated values provided by DisseminationCriteria are not appropriate. DisseminationCriteriaValue includes a domain element to which the value applies, and the value itself. For example, one domain might use the values “red”, “green”, and “yellow” to denote specific criteria, while another domain might use the values “1”, “2”, “3”, and “4”. This complex element allows any domain to utilize criteria values that meet their specific needs.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<DisseminationCriteriaValue>
        <DisseminationCriteriaDomainText/>
        <DisseminationCriteriaText/>
<DisseminationCriteriaValue/>
```

DisseminationCriteriaDomainText [1] – string. Provided by the data owner to indicate the domain that defines the dissemination criteria.

DisseminationCriteriaText [1] – string. Indicates who is allowed to see the information. For publish it provides information to the service provider about what can be returned as the result of a search. For search or get results, it provides information to the user so they know the criteria for sharing this information. Note that this is different from the data sensitivity included at the message level.

DataItemCategory [0-n] – Provides an optional list of data item categories that apply to the package. See DataItemCategory in section 5.1.3 for a more detailed description.

DomainAttribute [0-n] – Contains element names, element values, and XML blocks that are useful to specific service providers or consumers but that are not part of the LEXS specification. See DomainAttribute in section 5.1.9 for a more detailed description.

5.1.13 PDMessageMetadata

PDMessageMetadata includes information about the version of LEXS used to generate the message, the date and time the message was generated, data sensitivity, and a message identifier. This element is used in a doPublish.

The structure below shows a representation for this element. Descriptions for the sub-elements are also provided, along with cardinality which is shown in brackets.

```
<PDMessageMetadata>
        <LEXSVersion/>
        <MessageDateTime/>
        <MessageSequenceNumber/>
        <DataSensitivity/>
        <DomainAttribute/>
</PDMessageMetadata>
```

LEXSVersion [1] – string. Version of LEXS used to generate this request or response.

MessageDateTime [1] – dateTime. Date and time the message was created.

MessageSequenceNumber [1] – positive integer. Uniquely identifies a message from a specific application or service provider. This element may be used to indicate the order of processing for PD messages, for auditing purposes, to track messages for troubleshooting, and to tie results to the originating request.

DataSensitivity [0-1] – string. Information security classification level (e.g., SBU = Sensitive but Unclassified. Note that this is different from dissemination criteria, which is included at the data item level. DataSensitivity is required per business rule for doPublish requests, and can be supplied in doSearchResponse, getDataItemReponse, and getAttachmentResponse. It is not applicable to any other requests or responses and should be ignored if included.

DomainAttribute [0-n] – Contains element names, element values, and XML blocks that are useful to specific service providers or consumers but that are not part of the LEXS specification. See DomainAttribute in section 5.1.9 for a more detailed description.

5.1.14 RenderingInstructions

The RenderingInstructions element is used to display package information in a specific format. This element contains either instructions for converting the XML data in the package into a specified display format or instructions to display a pre-rendered version of the data item.

The Rendering Instructions include information about the rendering method and a reference to the associated Attachment or Narrative. Optional elements include rendering parameters and a description of the rendering.

The structure below shows a representation for this element. Descriptions for the sub-elements are also provided, along with cardinality which is shown in brackets.

```
<RenderingInstructions>
        <RenderingMethod/>
        <RenderingReference/>
        <RenderingParameter>
                < RenderingParameterName/>
                < RenderingParameterValue/>
        </RenderingParameter>
        <RenderingDescription/>
</RenderingInstructions>
```

RenderingMethod [1] – value from an enumerated code list. Enumerates possible rendering methods. The values and their meanings for rendering are shown below.

- XHTMLStyleSheet- XSTL Transformation results in XHTML
- HTMLStyleSheet - XSTL Transformation results in HTML
- PDFStyleSheet - XSLT Transformation results in PDF (XSL-FO)
- TextStyleSheet - XSLT Transformation results in TXT
- OtherStyleSheet - Other Stylesheet Transformation
- RenderedBinary - Attachment was pre-rendered into a binary
- Narrative - Narrative element contains pre-rendered content

RenderingReference [1] - reference to an Attachment Link or a Narrative element. Note in the case of an Attachment Link, this is not direct reference to an Attachment itself, but rather to an AttachmentLink located within DataItemPackage. The type of rendering is indicated by the RenderingMethod element. When the RenderingMethod indicates a stylesheet, the referenced Attachment is a stylesheet that can be applied to the package; RenderedBinary means the referenced Attachment is a pre-rendered document; and Narrative indicates that the Narrative element in this package contains pre-rendered content.

RenderingParameter [0-n] – container. Optional rendering parameters for XSLT provided by name/value (string) pairs.

RenderingParameterName [1] – string. XSLT rendering parameter name.

RenderingParameterValue [1] – string. XSLT rendering parameter value.

RenderingDescription [0-1] – string. Description of the rendered document.

For more information about working with rendering instructions see section 7.

5.1.15 ResponseMetadata

ResponseMetadata indicates whether the request resulted in an error or warning or success, provides additional information about the result in cases of error or warning, and provides an identifier for the message this is the result for. This element is included in all results.

The structure below shows a representation for this element. Descriptions for the sub-elements are also provided, along with cardinality which is shown in brackets.

```
<ResponseMetadata>
        <ResultCode/>
        <Advisory>
                <AdvisoryCategory/>
                <AdvisoryText/>
        <InResponseToMessageSequenceNumber/>
</ResponseMetadata>
```

ResultCode [1] – value from an enumerated code list containing Success, Warning, and Error.

- Success means everything worked as expected, and the request was handled exactly as provided.
- Warning means that the request was performed, but not exactly as provided. For example, an element or attribute being search for (i.e. search term) was ignored because it is unsupported, or a different quantity was returned (user asked for 50 and responder will only return 10 at a time), or a search option was ignored (user requested exact and responder only does soundex), etc.
- Error means the request could not be performed.

Advisory [0-n] – container. Advisory is not used if the result code is Success. However, if there is an error or warning, there must be one or more advisory elements to provide additional details about the error or warning.

AdvisoryCategory [1] – value from an enumerated code list. Provides a machine-interpretable error/warning message that can be used programmatically. For example, if the Advisory Category indicates that a service provider was offline, a client application could report it to the user so that the user could retry the query later, or the client application could automatically try the query later without user intervention. The possible values for the AdvisoryCategory element are provided in an enumerated list and are included in Appendix E.

AdvisoryText [0-1] – string. Free form text describing the warning or error. Since this text may be provided to end-users, service providers should populate this text field with user friendly messages rather than technical jargon or stack traces.

InResponseToMessageSequenceNumber [1] – positive integer. Uniquely identifies a request message that resulted in this response.

5.1.16 SearchRequestMetadata

SearchRequestMetadata provides information about the search, such as the number of hits requested, what kinds of data should be returned as part of a response, data item categories requested, roles requested, a count of the elements and attributes being searched for in any included Structured Payload blocks, information specifying additional hits, and an identifier for the request. This metadata is used in StructuredSearchRequestMessage in a doStructuredSearch request and in TextSearchRequestMessage in a doTextSearch request.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<SearchRequestMetadata>
        <DataOwnerIdentifier/>
        <MaxItemMatches/>
        <MatchBeginAfter/>
        <MatchEndBefore/>
        <ServiceProviderSearchID/>
        <RequestedData/>
        <DataItemCategoryText/>
        <SortOrder/>
        <TimeoutDuration/>
        <NumberOfStructuredPayloadTerms/>
        <StructuredPayloadsRequestedAbstract/>
        <DomainAttribute/>
</SearchRequestMetadata>
```

DataOwnerIdentifier [1] – Uniquely identifies the owning agency and system in the agency which owns the data. See DataOwnerIdentifier in section 5.1.6 for a more detailed description.

MaxItemMatches [1] – positive integer. The maximum number of hits the user (or application acting on behalf of the user) wants back in a response. Note that the service provider may override this value if the service provider’s limit is less than the number requested.

MatchBeginAfter [0-1] – string. See Appendix D for additional information on how this process works.

MatchEndBefore [0-1] – string. See Search Result Paging Support for additional information on how this process works.

ServiceProviderSearchID [0-1] – string. See Appendix D for additional information on how this element is used.

RequestedData [1] *([1-n] - 3.1.1 and later)* – value from an enumerated code list containing the following values:

- All
- Activity
- Aircraft
- CreditCard *(3.1.4 and later)*
- Document *(3.1.4 and later)*
- Drug
- Email
- Explosive
- Firearm
- Hash *(3.1.4 and later)*
- InstantMessenger *(3.1.4 and later)*
- IntangibleItem *(3.1.1 and later)*
- TangibleItem
- Location
- NetworkAddress *(3.1.4 and later)*
- Organization
- Person
- Snippet
- Substance
- TelephoneNumber
- Vehicle
- Vessel

Query may request a single type of LEXS entity or Snippet for responses. If a single type is requested, it is expected that this entity will be more fully populated than would be the case if the response included all kinds of LEXS entities.

DataItemCategoryText [0-n] – string. Indicates the data item categories that should be returned in a response.

SortOrder [0-1] – value from an enumerated code list containing

- Date
- Relevance.

This element specifies whether the most recent or most relevant hits should be returned first.

TimeoutDuration [0-1] – number. Numeric element that specifies the time in seconds within which the requesting system would like to receive a response. This element would primarily be used to convey a time period past which a response would not be useable by the requesting system. The time period specified excludes time for connection overhead to avoid the need for time synchronization between systems and thus should be slightly shorter than the period needed by the requesting system. The service provider, if capable of supporting such functionality, should cancel the search and return a timeout advisory response within the time limit specified. It is suggested that the search query should be timed out by the service provider prior to the time limit specified in order to allow time for canceling the query and returning a response before the period expires. Requesting systems are ultimately responsible handling timeouts and need to handle cases where the service provider cannot honor the timeout.

NumberOfStructuredPayloadTerms [0-n] – integer. See NumberOfStructuredPayloadTerms in section 5.1. Provides a count of the elements and attributes being searched for in any included Structured Payload elements so that service providers that do not understand one or more Structured Payloads have a count of terms that were ignored for the purposes of calculating an accurate MatchScore.

StructuredPayloadsRequestedAbstract [1] – abstract element. The substitutable elements indicate whether the requestor wants responses to include any available Structured Payloads, no Structured Payloads, or specific Structured Payloads. See StructuredPayloadsRequestedAbstract in section 5.1.22 for a more detailed description.

DomainAttribute [0-n] – Contains element names, element values, and XML blocks that are useful to specific service providers or consumers but that are not part of the LEXS specification. See DomainAttribute in section 5.1.9 for a more detailed description.

5.1.17 SearchResponseMetadata

SearchResponseMetadata provides information about the number of hits and the number of hits requested. Additional optional elements provide information that can be used in a subsequent query in order to get additional hits.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<SearchResponseMetadata>
        <StructuredQueryMatchScore/>
        <MaxItemMatchesRequested/>
        <NumberItemMatches/>
        <ServerLimitIndicator/>
        <MatchBeginPoint/>
        <MatchEndPoint/>
        <ServiceProviderSearchID/>
        <DomainAttribute/>
</SearchResponseMetadata>
```

StructuredQueryMatchScore [0-1] – positive integer between 0 and 100 representing a percentage. Service providers should make their best effort to answer queries the way they are asked. StructuredQueryMatchScore represents the percentage of the search elements that were matched for a structured search query. A result that matches all requested fields is a perfect match, represented by the value 100. Match scores for structured queries where the service provider had to ignore one or more elements or attributes specified in the search use numbers less than 100 to represent the percentage of the search terms that were matched by the response. StructuredQueryMatchScore is not applicable for text search results.

For example, assume there is a query with a Digest and two Structured Payloads for community A and community B. The query might be sent to a service provider that understands A only, or B only, both A and B, or neither A nor B. A service provider that does not understand both A and B uses the number of terms in any ignored Structured Payload to calculate an accurate match score. Assume that the Digest has 4 search terms, A has 3, and B has 1. If the service provider understands A and not B, the service provider still needs to know there are 8 total terms in order to calculate the match score. Since the service provider can process the Digest and Structured Payload A, it calculates that the Digest has 4 terms and A has 3 terms, and uses the NumberOfStructuredPayloadTerms for B to determine that the total number of terms requested is 8.

Note that each search term is a single element or attribute that describes a characteristic that distinguishes the object of interest from other objects. For example, if the search is against a Structured Payload person to find someone with a 10 inch scar, the person entity provides the context for the search, the element that indicates the feature is a scar is one search term, the element that indicates it has length 10 is a search term, and the element that indicates the unit of measure is inches is a separate search term; three search terms in total. If a unit of measure uses an attribute rather than an element, it still counts as a search term.

Again assume that a query has two Structured Payloads, the Digest has 4 search terms, A has 3, B has 1 term and the service provider can understand Structured Payload A and not B. If response data matches all 5 Digest terms, and the scar search term, but not the length and unit terms from Structured Payload A, then the match score is 62, calculated by dividing the 5 search terms matched by the 8 total search terms.

MaxItemMatchesRequested [1] – positive integer. The maximum number of hits the user (or application acting on behalf of the user) wants back in a response. Note that the service provider may override this value if the service provider’s limit is less than the number requested.

NumberItemMatches [1] – string. The number of matches found, as opposed to returned, by the service provider. See Appendix D for additional information on how this process works.

ServerLimitIndicator [1] – Boolean.   Indicates whether the number of hits was restricted due to a service provider limit. True means the number returned was limited due to a server restriction.

MatchBeginPoint [0-1] – string. See Appendix D for additional information on how this process works.

MatchEndPoint [0-1] – string. See Appendix D for additional information on how this process works.

ServiceProviderSearchID [0-1] – string. See Appendix D for additional information on how this element is used.

DomainAttribute [0-1] – Contains element names, element values, and XML blocks that are useful to specific service providers or consumers but that are not part of the LEXS specification. See DomainAttribute in section 5.1.9 for a more detailed description.

5.1.18 SearchResultPackage

SearchResultPackage contains information specific to the results of searches and get data items, information about the data owner that provided the data, a Digest, and possibly one or more Structured Payloads, Narrative, Snippet, Rendering Instructions, and Attachment Link elements. SearchResultPackage is contained in a SearchResponseMessage.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<SearchResultPackage>
        <PackageMetadata/>
        <Digest/>
        <StructuredPayload/>
        <AttachmentLink/>
        <DataSubmitterMetadata/>
        <Snippet/>
</SearchResultPackage>
```

PackageMetadata [1] – Contains metadata about LEXS Package. See PackageMetadata in section 5.1.12 for a more detailed description.

DataSubmitterMetadata [0-1] – Includes the organization, the system name, and possibly the contact information for the data submitter. See MessageOriginMetadata and DataSubmitterMetadata in section 5.1.10 for a more detailed description.

Digest [0-1] – Contains LEXS Entities and Associations between Entities. Note that the Digest is optional here since a query can specify that only a Snippet should be returned. See Digest in section 5.1.8 for a more detailed description.

StructuredPayload [0-n] – Contains the Structured Payload. See StructuredPayload in section 5.1.20 for a more detailed description.

Snippet [0-1] – string. Snippet provides summary information that represents the context of this data item in the responding system.

AttachmentLink [0-n] – Contains information about a link to an Attachment. See AttachmentLink in section 5.1.2 for a more detailed description.

5.1.19 SRMessageMetadata

SRMessageMetadata includes information about the version of LEXS used to generate the message, the date and time the message was generated, data sensitivity, a message identifier, message origin and destination. This element is used in all SR requests and responses.

The structure below shows a representation for this element. Descriptions for the sub-elements specific to this element are also provided, along with cardinality which is shown in brackets.

```
<SRMessageMetadata>
        <LEXSVersion/>
        <MessageDateTime/>
        <MessageSequenceNumber/>
        <DataSensitivity/>
        <MessageOriginMetadata/>
        <MessageDestinationIdentifier>
                <AgencyIDAbstract/>
                <OrganizationName/>
                <SystemID/>
        </MessageDestinationIdentifier>
        <DomainAttribute/>
</SRMessageMetadata>
```

LEXSVersion [1] – string. Version of LEXS used to generate this request or response.

MessageDateTime [1] – dateTime. Date and time the message was created.

MessageSequenceNumber [1] – positive integer. Uniquely identifies a message from a specific application or service provider. Used for auditing purposes, to track messages for troubleshooting, and to tie results to the originating request.

DataSensitivity [0-1] – string. Information security classification level (e.g., SBU = Sensitive but Unclassified. Note that this is different from dissemination criteria, which is included at the data item level. DataSensitivity is required per business rule for doPublish requests, and can be supplied in doSearchResponse, getDataItemReponse, and getAttachmentResponse. It is not applicable to any other requests or responses and should be ignored if included.

DomainAttribute [0-n] – Contains element names, element values, and XML blocks that are useful to specific service providers or consumers but that are not part of the LEXS specification. See DomainAttribute in section 5.1.9 for a more detailed description.

MessageOriginMetadata [1] – Specifies the organization and system, plus contact information for the system, where a message originated or was submitted from. See MessageOriginMetadata/DataSubmitterMetadata in section 5.1.10 for a more detailed description.

MessageDestinationIdentifier [1] – container. Specifies the organization and system that an SR message is being sent.

AgencyIDAbstract [0-1] – abstract element. Can be substituted with ORI or OriginatingAgencyID, both of which are text elements. Specifies an agency identifier such as an ORI code or similar.

OrganizationName [1] – string. Specifies the organization name for the system where the message is being sent.

SystemID [1] – string. Specifies the system where the message is being sent. An ID or well known acronym for the system should be supplied rather than spelling out the system.

5.1.20 StructuredPayload

The StructuredPayload element is the LEXS mechanism that allows communities, projects, systems to include base object, roles, associations, structures and elements in a LEXS message that are not defined within the LEXS. Communities that need data not provided in the Digest can supply that data in one or more StructuredPayloads.

A consuming system that encounters Structured Payload elements does not have to understand or process every possible Structured Payload. All LEXS-compliant systems must recognize this tag and its existence if it occurs, but do not have to process the content of the StructuredPayload element. The “processContents” attribute associated with “lexs:PackageStructuredPayloadType” is defined as “skip” in the “lexs schema”. This implies that no validation is necessary; the element must simply be well formed.

This element contains a metadata element that identifies the community or system that defined the payload, but the rest of the definition within the structure is entirely up to the particular community or system.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<StructuredPayload>
        <StructuredPayloadMetadata/>
        <any/>
</StructuredPayload>
```

StructredPayloadMetadata [1] –Identifies the community or system that defined the payload Version of LEXS used to generate this request or response. See StructredPayloadMetadata in section 5.1.21 for a more detailed description.

any – xsd:any. Used here to represent a Structured Payload defined outside of LEXS. The “processContents” attribute is defined as “skip”, meaning that no validation will be performed; the element must simply be well formed. (For usage of xsd:any see W3C XML Schema Primer Section 5.5 "Any Element, Any Attribute" at http://www.w3.org/TR/xmlschema-0/#any.)

Note that since the data contents of the Structured Payload (the “any” above) are based on a separate schema, the contents must be valid as a standalone instance. While the data contents of the Structured Payload may not make logical sense by itself without the Digest, the Structured Payload data contents must be validatable against the Structured Payload schema.

5.1.21 StructuredPayloadMetadata

This element describes and identifies the Structured Payload. The CommunityURI and CommunityVersion are mandatory fields used in combination to uniquely identify the Structured Payload schema so that consumers know how to process any Structured Payload instances based on that schema.

The structure below shows a representation for this element. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<StructuredPayloadMetadata>
        <CommunityURI/>
        <CommunityDescription/>
        <CommunityVersion/>
        <CommunityPedigreeURI/>
</StructuredPayloadMetadata>
```

CommunityURI [1] – string. Structured payload community Uniform Resource Identifier. Each community is uniquely identified by CommunityURI. Value must follow the established rules for a URI. Used in combination with CommunityVersion to identify the Structured Payload uniquely.

CommunityDescription [0-1] – string. Description of the community that originated a particular Structured Payload.

CommunityVersion [1] – string. Structured payload community version. Used in combination with CommunityURI to identify the Structured Payload uniquely.

CommunityPedigreeURI [0-n] – string. A Uniform Resource Identifier that uniquely identifies a Structured Payload schema that this Structured Payload is built on and links to. Pedigree number attribute indicates community pedigree order. The LEXS Digest is assigned pedigree 0, community receives pedigree 1, and etc. The value must follow the established rules for an URI. As an example, Figure 4 shows how the cyber incident community can build upon the incident community's content. In this case, the cyber incident community's Structured Payload would populate the CommunityPedigreeURI element with the URI of the incident community to indicate that cyber incident community's Structured Payload builds upon the incident community's Structured Payload. In this case, the attribute would be populated with the value "1" " since the incident community's Structured Payload builds upon the Digest. For communities that build directly upon the Digest, such as the incident community used in the example, the CommunityPedigreeURI element would not be provided since the pedigree is just for documenting Structured Payloads building upon other Structured Payloads.

5.1.22 StructuredPayloadsRequestedAbstract

StructuredPayloadsRequestedAbstract is an abstract element, which can be substituted for by the elements StructuredPayloadsRequestedCode or StructuredPayloadsRequestedList.

StructuredPayloadsRequestedCode can have the values “All” or “None”. “All” indicates that the requesting system would like to receive any Structured Payloads that the service provider can return. “None” indicates that the requesting system does not want any Structured Payloads.

StructuredPayloadsRequestedList provides a wrapper for one or more StructuredPayloadMetadata elements. If this list element is provided instead of the code element, it means that the requesting system does not want to receive responses with any Structured Payloads except for the ones specified.

When providing StructuredPayloadsRequestedList, the StructuredPayloadMetadata elements need only contain the CommunityURI and CommunityVersion. These fields in combination uniquely identify the Structured Payload schema so that consumers know how to process any Structured Payload instances based on that schema. For the purposes of identifying Structured Payloads to be included in responses, the optional elements, CommunityDescription and CommunityPedigreeURI, do not provide any benefit.

It should be noted that requesting systems may need to take into account whether they utilize Rendering Instructions when specifying whether all, none, or specific Structure Payloads should be included in responses. For example, if a service provider includes Rendering Instructions that the requesting system may want to utilize, the requesting system may need all Structured Payloads in order to utilize stylesheet-based rendering. Alternatively, if the requesting system will not utilize Rendering Instructions, or will only utilize Rendering Instructions that are not stylesheet-based, then the requesting system may not want any Structured Payloads that it cannot process directly.

5.1.23 StructuredQuery

This element contains a structured query, stated using the LEXS Digest and Structured Payload entities by supplying field values for selected elements. This solution allows implementers to leverage tools and constructs developed for processing XML instances that utilize the well-defined Digest and Structured Payload constructs, and provides a straightforward means of validating the contents of a structured query. A simple query looks like a response or a publish request. There are organizational elements that sit “above” the Digest and Structured Payload elements to provide modifiers (such as fuzzy or greater than), as well as to allow queries to include multiple values for individual elements.

The structure below shows a representation for this complex element. Descriptions for the sub-elements are also provided, along with cardinality which is shown in brackets. Section 6.2 provides details on the requirements for structured queries and examples for how each requirement is handled in actual queries.

```
<StructuredQuery>
        <DigestQueryStatement>
                <DigestQueryField/>
                <QueryMatch/>
        </DigestQueryStatement>
        <StructuredPayloadQueryStatement>
                <StructuredPayloadMetadata/>
                <StructuredPayloadQueryField/>
                <QueryMatch/>
        </StructuredPayloadQueryStatement>
        <RoleList>
                <RoleInclusiveIndicator/>
                <RoleType/>
        </RoleList>
</StructuredQuery>
```

DigestQueryStatement [0-n] – container. Provides a wrapper for the Digest elements being searched, their values, and any modifiers (such as "fuzzy"). This element may not exist if the search is only being performed on Structured Payload elements. This element includes a single Digest element with a single value; if multiple Digest elements are to be searched, or if multiple values are supplied for a single element, there must be multiple DigestQueryStatement blocks. So for example, if a query is for a person last name and first name, there will be one DigestQueryStatement block with the last name, and a second DigestQueryStatement block with the first name. If a query is on a person last name of either Smith or Jones, there will be one DigestQueryStatement block with the last name Smith, and a second DigestQueryStatement block with the last name Jones.

DigestQueryField [1] – container. Provides a wrapper for a single LEXS element with a single value.

QueryMatch [1] – value from an enumerated code list containing

- fuzzy – approximate or loose match
- exact – equal
- gt – greater than
- ge – greater than or equal to
- lt – less than
- le – less than or equal to
- wildcard – value includes one or more wildcard characters

StructuredPayloadQueryStatement [0-n] – container. Provides a wrapper for Structured Payload elements being searched, their values, and any modifiers (such as "fuzzy"). This element may not exist if the search is only being performed on Digest elements. This element includes a single Structured Payload element with a single value; if multiple Structured Payload elements are to be searched, or if multiple values are supplied for a single element, there must be multiple StructuredPayloadQueryStatement blocks.

StructuredPayloadQueryField [1] – container. Provides a wrapper for a single Structured Payload element with a single value.

StructuredPayloadMetadata [1] – Provides information that uniquely identifies a Structured Payload, including its unique namespace, a description what this payload is for, and a version number for the namespace. See StructredPayloadMetadata in section 5.1.21 for a more detailed description.

RoleList [0-1] – container. Provides a wrapper for requested role types plus an indicator as to whether the supplied role types should be included or excluded from the response.

RoleInclusiveIndicator [1] – Boolean. Indicates whether the role types supplied in the RoleTypes element(s) should be taken as an inclusion list or an exclusion list. So for example, if the RoleType is specified as just SubjectType, the inclusive indicator set to True would mean that the response should only include people who have SubjectType roles (such as Subject, Suspect, etc.), while False would mean that response should include all people except those with SubjectType roles.

RoleTypes [1-n] – value from an enumerated code list containing all role types defined in the LEXS 3.1 data schema, such as SubjectType, WitnessType, MissingPersonType, etc. See Appendix C for the complete list. A query may specify one or more of these types be returned in responses. For example, a user may only be interested in who has done something and may request only SubjectType. Alternatively, a user may want everything but enforcement officials and may specify EnforcementOfficialType.

5.1.24 UserAssertion

UserAssertion describes the user submitting any request that is done by a user, or on behalf of a user, by an application. Note that requests that come from applications, such as get capabilities, get data owners, and get availability do not include a user assertion. This element is included in StructuredSearchRequestMessage in a doStructuredSearch request, a TextSearchRequestMessage in a doTextSearch request, a GetDataItemRequestMessage in a getDataItem request, and in a getAttachmentRequestMessage in a getAttachmentRequest.

The UserAssertion should be included at the transport level (such as SOAP) as well as in the SR schemas. The information should be available at the transport level to facilitate its use by web services, precluding the need for such services to “dig” into included payloads. The information is also available at the SR schema level to facilitate its use by service providers, precluding the need for higher level service (such as the web service) to pass the information along separately to lower level functions that may need to make data decisions based on user information.

The structure below shows a high level representation for this request. Descriptions for the elements are also provided, along with cardinality which is shown in brackets.

```
<UserAssertion>
        <UserID/>
        <PersonGivenName/>
        <PersonSurName/>
        <ContactMeans/>
        <OrganizationName/>
        <AgencyIDAbstract/>
        <DomainAttribute/>
</UserAssertion>
```

UserID [1] – string. Username or userID for the user on the originating system.

PersonGivenName [0-1] – string. First or given name for the user making the request.

PersonSurName [0-1] – string. Last or surname for the user making the request.

ContactMeans [0-n] – abstract element. Can be substituted with the ContactEmailID (as string), or telephone numbers (of TelephoneNumberType) ContactTelephoneNumber, ContactFaxNumber, ContactMobileTelephoneNumber or ContactPagerNumber.

OrganizationName [1] – string. Organization name the user belongs or is assigned to when performing a request.

AgencyIDAbstract [0-1] – abstract element. Can be substituted with ORI or OriginatingAgencyID, both of which are text elements. Specifies an agency identifier such as an ORI code, for Law Enforcement agencies that use ORI, or some other agency identification code where ORI is not appropriate.

DomainAttribute [0-n] – Contains element names, element values, and XML blocks that are useful to specific service providers or consumers but that are not part of the LEXS specification. See DomainAttribute in section 5.1.9 for a more detailed description.

5.2 NIEM Structures

5.2.1 NIEM Metadata

NIEM's Metadata defines information that supports the actual content of XML instances. The metadata feature provides a mechanism for attaching structured properties that describe the pedigree or source (ReportedDate, LastUpdateDate, CommentText, SourceIDText, etc.) of instance data to any component of the model in any namespace. LEXS extends the NIEM Metadata structure to add an element called LogicalIDText which is discussed later in this document.

As shown in the XML fragment below, a Metadata component is identified uniquely within a message using an "s:id" attribute. Other components of the message refer to the Metadata block by specifying that identifier as the value of an “s:metadata” attribute. Any number of objects can refer to the same Metadata block.

The example below illustrates the use of Metadata to mark the drug entity with a specific SourceIDText (Drug111). The Drug object (Drug1) references the MDrug1 metadata block indicating that Drug111 is the Source ID for this Drug object.

```
<lexsdigest:EntityDrug>
   <lexsdigest:Metadata s:id="MDrug1">
      <nc:SourceIDText>Drug111</nc:SourceIDText>
   </lexsdigest:Metadata>
   <nc:Drug s:id="Drug1" s:metadata="MDrug1">
          --------------
   </nc:Drug>
</lexsdigest:EntityDrug>
```

In LEXS, Metadata is especially important since the SourceIDText element is used to indicate that objects are the same, and the LogicalIDText element is used to indicate that objects may be the same. This use of Metadata is described in section 5.3.6.

5.2.2 NIEM Roles

A role represents a particular context or activity for a data object. The object to which the role applies is called the base object. Any single base object instance may have multiple roles for a given context or activity. For example, a single person may take the role of "ArrestingOfficial", "Victim", and "Witness" in the same instance.

In XML Schema, a new role is represented as a XML Schema complex type. The type contains a particular "RoleOf<BaseObject>” XML element. This element represents the base object to which this role applies. The LEXS Digest defines several roles in addition to the NIEM roles it uses. See Appendix C for the complete list.

A role may be specific to time, incident, or employment. For example, if “Billy Bob Guy” picks up an object and hits “John Jacobs” with it, the object will take on the role of a weapon, and “Billy Bob Guy” will take on the role of a "subject", and “John Jacobs” will take on the role of "victim". For example, the weapon role applies to an object, and may include a user of the weapon, an activity in which it was involved, and a description of how the weapon was used. Consider the EntityFirearm shown below:

```
<lexsdigest:EntityFirearm>
    <nc:Firearm>
        <nc:ItemDescriptionText>Big Gun</nc:ItemDescriptionText>
            <nc:ItemSerialIdentification>
                <nc:Identification>123ABC4578</nc: Identification >
            </nc: ItemSerialIdentification >
            <nc:FirearmMakeCode>NOM</nc:FirearmMakeCode>
            <nc:FirearmCategoryCode>P</nc:FirearmCategoryCode>
            <c:FirearmCategoryDescriptionCode>R</nc:FirearmCategoryDescriptionCode>
            <nc:FirearmAutomaticIndicator>true</nc:FirearmAutomaticIndicator>
            <nc:FirearmCaliberCode>9</nc:FirearmCaliberCode>
            <nc:FirearmFinishCode>LAV</nc:FirearmFinishCode>
    </nc:Firearm>
</lexsdigest:EntityFirearm>
```

The same firearm used as a weapon could be represented in the structure:

```
<lexsdigest:EntityFirearm>
    <nc:Firearm s:id="Frm2">
        <nc:ItemDescriptionText>Big Gun</nc:ItemDescriptionText>
        <nc:ItemSerialIdentification>
            <nc:Identification>123ABC4578</nc: Identification >
        </nc: ItemSerialIdentification >
        <nc:FirearmMakeCode>NOM</nc:FirearmMakeCode>
        <nc:FirearmCategoryCode>P</nc:FirearmCategoryCode>
        <c:FirearmCategoryDescriptionCode>R</nc:FirearmCategoryDescriptionCode>
        <nc:FirearmAutomaticIndicator>true</nc:FirearmAutomaticIndicator>
        <nc:FirearmCaliberCode>9</nc:FirearmCaliberCode>
        <nc:FirearmFinishCode>LAV</nc:FirearmFinishCode>
    </nc:Firearm>
    <nc:Weapon>
        <nc:RoleOfItemReference s:ref="Frm2"/>
    </nc:Weapon>
</lexsdigest:EntityFirearm>
```

In the above example, “nc:RoleOfItemReference” is of “s:ReferenceType” and points to the “s:id” attribute of “nc:Firearm” that has a value of “Frm2”.

The following structure shows how additional roles can be represented for “Billy Bob Guy”, showing him as the subject of an arrest and incident, as well as the user of a weapon:

```
<lexsdigest:EntityPerson>
    <lexsdigest:Person s:id="Sub2">
        <nc:PersonName>
            <nc:PersonGivenName>Billy</nc:PersonGivenName>
            <nc:PersonMiddleName>Bob</nc:PersonMiddleName>
            <nc:PersonSurName>Guy</nc:PersonSurName>
        </nc:PersonName>
        <nc:PersonSSNIdentification>
            <nc: Identification >987654321</nc: Identification >
        </nc:PersonSSNID>
    </lexsdigest:Person>
    <j:ArrestSubject>
        <nc:RoleOfPersonReference s:ref="Sub2"/>
    </j:ArrestSubject>
    <j:IncidentSubject>
        <nc:RoleOfPersonReference s:ref="Sub2"/>
    </j:IncidentSubject>
</lexsdigest:EntityPerson>

<lexsdigest:EntityFirearm>
    <nc:Firearm s:id="Frm2">
        <nc:PropertyDescriptionText>Big Gun</nc:PropertyDescriptionText>
        <nc:ItemSerialIdentification>
            <nc:Identification>123ABC4578</nc: Identification >
        </nc: ItemSerialIdentification >
        <nc:FirearmMakeCode>NOM</nc:FirearmMakeCode>
        <nc:FirearmCaliberCode>9</nc:FirearmCaliberCode>
        <nc:FirearmFinishCode>LAV</nc:FirearmFinishCode>
    </nc:Firearm>
    <nc:Weapon>
        <nc:RoleOfPropertyReference s:ref="Frm2"/>
        <nc:WeaponUserReference s:ref="Sub2"/>
    </nc:Weapon>
</lexsdigest:EntityFirearm>
```

The “nc:WeaponUserReference” to “Billy Bob Guy” is included in the Weapon role through the “Sub2” pointer in the Firearm entity. Additional roles for “Billy Bob Guy” as a “j:ArrestSubject” and a “j:IncidentSubject” are included in the Person entity.

5.2.3 NIEM Associations

Associations represent the relationship among objects. In both LEXS and NIEM, these are called association objects, and the corresponding XML types are association types. Figure 21 is a logical representation of some, but not all, of the associations in LEXS.

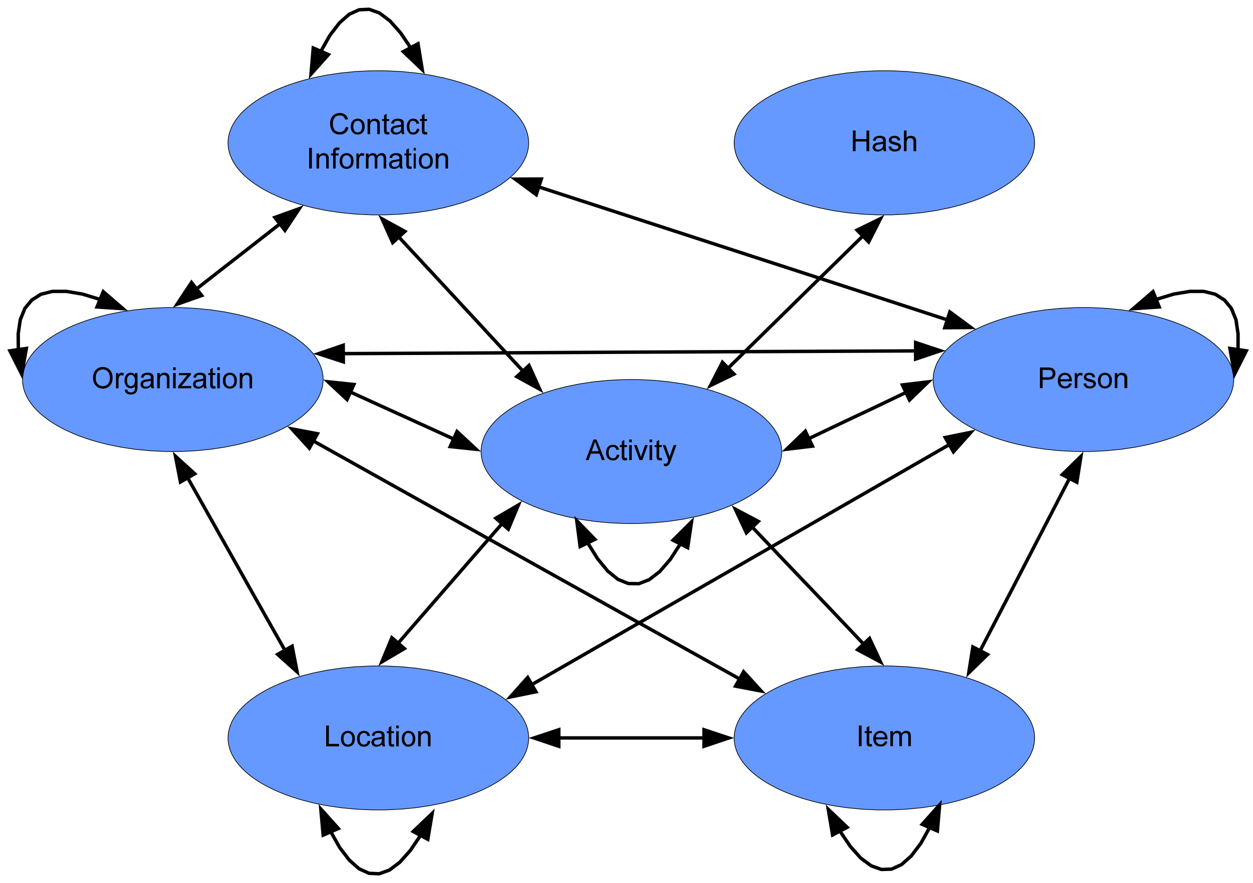


Figure 21. Associations Among Entities

The loops typically represent associations between two different instances of the same kind of entity. In other words, if person X has a relationship with person Y, then an instance of a person object representing X will have an association with a different instance of another person object representing Y. The loop signifies that such a relationship between two instances of a person entity is possible.

LEXS includes a large number of NIEM associations, and defines additional ones to meet the needs of the LEXS community. Communities can define additional associations in their Structured Payloads. In general, associations link two base objects, such as a person and a location. However, some associations link to roles.

Some associations only include a single type of reference, such as the NIEM FriendshipAssociation that only includes references to a Person. For these associations, instances will contain multiple PersonReference elements in order to represent a connection between two people. A FriendshipAssociation element could include a large number of PersonReference elements in order to show that a number of people are friends with each other.

Some associations include specific semantics for reference elements, such as the GuardianAssociation that includes a reference to the person who is a guardian and a person who is the dependent. Again multiples are allowed, so for example a single GuardianAssociation element could include one guardian person and multiple dependents.

Some associations include a number of different types of references, such as the NIEM TransportationAssociation that includes references to a conveyance operator, conveyed people, conveyed items, and the conveyance itself. However, just because all these different references exist, instances are not required to use them all. A TransportationAssociation instance might just include the operator and the conveyance, or perhaps even just conveyed people and the conveyance if it is not known who was driving.

As noted earlier, many associations provide additional context for the data, such as IncidentWitnessAssociation which is much more specific than ActivityInvolvedPersonAssociation. Exchanges should use the most specific association available in order to provide the most concise information to consumers. While additional associations may be defined in structured payloads in order to provide even more concise context, implementers should keep in mind that the information contained in the structured payload may not be understood by all consumers. Therefore implementers need to balance the need for precise context with the compatibility of exchanges. In general, digest associations should be used where feasible to provide as much interoperability as possible, and structured payload associations defined only where no digest associations apply.

LEXS also includes several associations to connect specific attachment types to the appropriate entities. These associations are provided to support user interfaces that may need to know the kinds of attachments available in order to make decisions about where or what to display, such as a mug shot versus a crime scene photo. The following Attachment-related associations are available:

EntityPersonFacialImageAssociation can refer to a mug shot, or driver license photo, or any image that shows a <u>frontal</u> view of a person’s face. This may even be a waist-up image; however, the purpose is for facial identification, so this association should be used with care.

EntityPersonSMTImageAssociation is used for when the image is of a scar, mark, or tattoo.

EntityPersonImageAssociation serves two purposes. One is to support systems that have images connected to a person without any way to determine whether it is a facial image versus an SMT image. Secondly, system may have other kinds of person images beyond just facial or SMT images.

EntitySupportingDocumentationAssociation may be a PDF or MS Word file representing a report or a piece of evidence.

EntityCrimeSceneImageAssociation provides a link between an entity and a crime scene image.

EntityItemImageAssociation links an entity and an item image, such as a picture of a vehicle.

EntityAttachmentLinkAssociation provides a mechanism to link entities to Attachments when none of the other associations are appropriate.

The specific Associations used in LEXS are listed in Appendix B.

All NIEM and LEXS associations include information (defined in schema using appinfo) that specifies what objects may be referenced. For example, the FriendshipAssociation includes PersonReference, and PersonReference is defined as a reference to PersonType. This means that the PersonReference element must refer to an element of PersonType or of a type derived from PersonType. Validating XML parsers will not catch errors in references, so implementers must ensure that references are used properly. EntityPersonFacialImageAssociation, EntityPersonSMTImageAssociation, EntityPersonImageAssociation, EntitySupportingDocumentationAssociation, EntityCrimeSceneImageAssociation, EntityItemImageAssociation, and EntityAttachmentLinkAssociation all include references to Entity elements, such as the EntityPerson element. All other associations reference element inside the entities, such as a Person element.

Below is an XML fragment which includes a person entity and two organization entities. In this example there is an association between the organization elements and an association between the person element and one of the organization elements.

```
<lexs:Digest>
  <lexsdigest:EntityPerson>                                      <!--   Person Arr2   -->
    <lexsdigest:Person s:id="Arr2">
      <nc:PersonName>
        <nc:PersonGivenName>Bubba</nc:PersonGivenName>
        <nc:PersonSurName>Johnson</nc:PersonSurName>
      </nc:PersonName>
    </lexsdigest:Person>
  </lexsdigest:EntityPerson>
  < lexsdigest:EntityOrganization>                               <!--   Gang Org2   -->
    <nc:Organization s:id="Org2">
      <nc:OrganizationName>Westside Renegades</nc:OrganizationName>
      <nc:OrganizationCategoryText>Gang</nc:OrganizationCategoryText>
    </nc:Organization>
  </lexsdigest:EntityOrganization>
  <lexsdigest:EntityOrganization>                                <!--   Gang Org3   -->
    <nc:Organization s:id="Org3">
      <nc:OrganizationName>Snakes</nc:OrganizationName>
      <nc:OrganizationCategoryText>Gang</nc:OrganizationCategoryText>
    </nc:Organization>
  </ lexsdigest:EntityOrganization>
  < lexsdigest: Associations>
    <!--   Organization to Organization Association   -->
    <nc:OrganizationAssociation
      <nc:AssociationBeginDate>
         <nc:Year>1967</nc:Year>
      </nc:AssociationBeginDate>
      <nc:AssociationEndDate>
         <nc:Year>1973</nc:Year>
      </nc:AssociationEndDate>
      <nc:OrganizationReference s:ref="Org2"/>
      <nc:OrganizationReference s:ref="Org3"/>
    </nc:OrganizationAssociation>
    <!--   Person to Gang Association   -->
    <nc:PersonGangAssociation>
      <nc:PersonReference s:ref="Arr2"/>
      <nc:OrganizationReference s:ref="Org2"/>
    </nc:PersonGangAssociation>
  </lexsdigest: Associations>
</lexs:Digest>
```

The XML fragment below shows an example where there is an association between the person entity and an attached image. The association references the person entity and the attachment link; the attachment link refers to the image attachment using the AttachmentURI.

```
<lexs:PublishMessage>
…
  <lexs:DataItemPackage>
…
    <lexs:Digest>
      <lexsdigest:EntityPerson s:id="EArr1">
        <lexsdigest:Person>
          <nc:PersonName>
            <nc:PersonGivenName>Bubba</nc:PersonGivenName>
            <nc:PersonMiddleName>Ray</nc:PersonMiddleName>
            <nc:PersonSurName>Johnson</nc:PersonSurName>
          </nc:PersonName>
        </lexsdigest:Person>
      </lexsdigest:EntityPerson>

      <lexsdigest:Associations>
        <lexsdigest:EntityPersonFacialImageAssociation>
          <lexsdigest:EntityReference s:ref="EArr1"/>
          <lexsdigest:AttachmentLinkReference s:ref="PAttachment1"/>
        </lexsdigest:EntityPersonFacialImageAssociation>
      </lexsdigest:Associations>
    </lexs:Digest>

    <lexs:AttachmentLink s:id="PAttachment1">
      <lexs:AttachmentURI>http://www.xyz.org/LEXS/examples/mugshot1.jpg</lexs:AttachmentURI>
      <lexs:AttachmentViewableIndicator>true</lexs:AttachmentViewableIndicator>
      <nc:BinaryDescriptionText>Mug shot for Bubba Johnson</nc:BinaryDescriptionText>
    </lexs:AttachmentLink>
  </lexs:DataItemPackage>

  <lexs:Attachment>
    <lexs:AttachmentURI>http://www.xyz.org/LEXS/examples/mugshot1.jpg</lexs:AttachmentURI>
    <nc:Binary>
      <nc:BinaryBase64Object/> <!-- empty to shorten example -->
       <nc:BinaryDescriptionText>Mug shot for Bubba Johnson</nc:BinaryDescriptionText>
       <nc:BinaryFormatID>image/jpg</nc:BinaryFormatID>
       <nc:BinaryFormatStandardName>MIME</nc:BinaryFormatStandardName>
       <nc:BinaryCategoryText>Mug Shot</nc:BinaryCategoryText>
    </nc:Binary>
  </lexs:Attachment>

</lexs:PublishMessage>
```

5.3 LEXS References

5.3.1 Attachment Referencing

LEXS utilizes Attachments for two purposes. One is to contain data-related binary information for one or more entities, such as a mug shot or fingerprint. The second is to support LEXS Rendering Instructions, such as rendering stylesheets or prerendered views of data. Attachments can be a part of a PublishMessage or an AttachmentReponseMessage.

An Attachment is identified by an AttachmentURI. The AttachmentURI uniquely identifies the Attachment and must follow all rules for an URI (Uniform Resource Identifier). An AttachmentLink is a structure that contains an AttachmentURI and other descriptive information about the Attachment, and is used to link an Attachment to one of the LEXS image/attachment associations (such as PersonFacialImageAssociation) or to a RenderingInstructions element. Note that if the URI is provided as a URL, it does not imply that the attachment is available over the Internet at that address, similar to the way URLs in XML namespaces do not imply that the address is resolvable.

In the PublishMessage an Attachment may be part of multiple packages, such as a message with multiple incident reports that include the same person and his mug shot. In this case, each incident report is represented in separate DataItemPackages, each of which uses an AttachmentLink to refer to the mug shot Attachment. As illustrated below, within the DataItemPackage an association is used to connect the entity to the AttachmentLink.

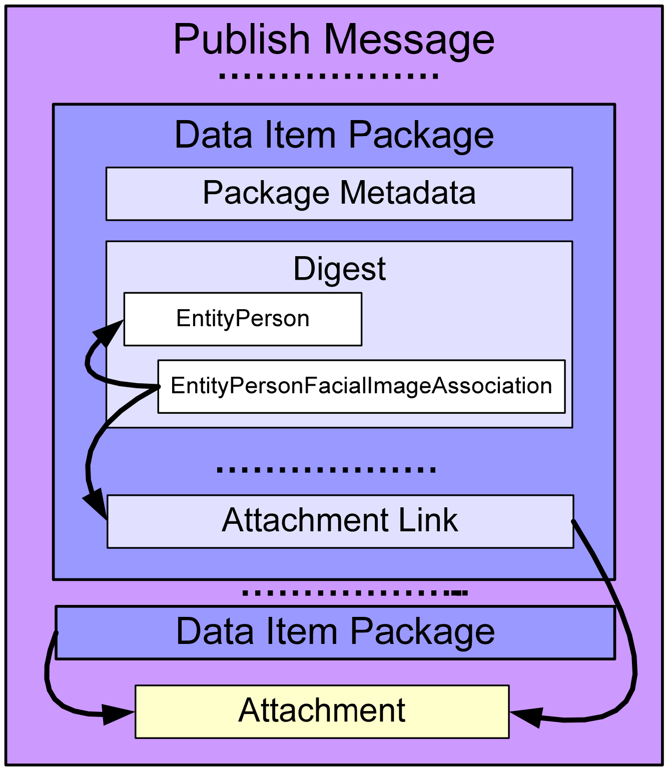


Figure 22. Data Attachment Linkage in Publish Message

The method of associating an entity and an Attachment is also demonstrated in the XML fragment below.

```
<lexs:PublishMessage>
…
  <lexs:DataItemPackage>
…
    <lexs:Digest>
      <lexsdigest:EntityPerson s:id="EArr1">
        <lexsdigest:Person>
          <nc:PersonName>
            <nc:PersonGivenName>Bubba</nc:PersonGivenName>
            <nc:PersonMiddleName>Ray</nc:PersonMiddleName>
            <nc:PersonSurName>Johnson</nc:PersonSurName>
          </nc:PersonName>
        </lexsdigest:Person>
      </lexsdigest:EntityPerson>

      <lexsdigest:Associations>
        <lexsdigest:EntityPersonFacialImageAssociation>
          <lexsdigest:EntityReference s:ref="EArr1"/>
          <lexsdigest:AttachmentLinkReference s:ref="PAttachment1"/>
        </lexsdigest:EntityPersonFacialImageAssociation>
      </lexsdigest:Associations>
    </lexs:Digest>

    <lexs:AttachmentLink s:id="PAttachment1">
      <lexs:AttachmentURI>http://www.xyz.org/LEXS/examples/mugshot1.jpg</lexs:AttachmentURI>
      <lexs:AttachmentViewableIndicator>true</lexs:AttachmentViewableIndicator>
      <nc:BinaryDescriptionText>Mug shot for Bubba Johnson</nc:BinaryDescriptionText>
    </lexs:AttachmentLink>
  </lexs:DataItemPackage>

  <lexs:Attachment>
    <lexs:AttachmentURI>http://www.xyz.org/LEXS/examples/mugshot1.jpg</lexs:AttachmentURI>
    <nc:Binary>
      <nc:BinaryBase64Object/> <!-- empty to shorten example -->
       <nc:BinaryDescriptionText>Mug shot for Bubba Johnson</nc:BinaryDescriptionText>
       <nc:BinaryFormatID>image/jpg</nc:BinaryFormatID>
       <nc:BinaryFormatStandardName>MIME</nc:BinaryFormatStandardName>
       <nc:BinaryCategoryText>Mug Shot</nc:BinaryCategoryText>
    </nc:Binary>
  </lexs:Attachment>

</lexs:PublishMessage>
```

An AttachmentLink can also be part of a DataItemPackage in a DataItemResponseMessage or SearchResultPackage in a SearchResponseMessage. In both of these cases, the linkages are similar to the figure and example above with the exception that the referenced Attachment is not in the message. In the interest of limiting the size of these messages, AttachmentLinks are used to identify and describe Attachments, but the results do not include the Attachments themselves. A getAttachmentRequest operation retrieves an Attachment of interest using the AttachmentURI to identify the specific Attachment.

RenderingInstructions can also refer to an Attachment using an AttachmentLink as shown below. As discussed in section 5.1.14, the RenderingInstructions structure contains a RenderingReference element. If the RenderingReference points to an AttachmentLink, the referenced Attachment is either a stylesheet or a pre-rendered document. In the case of a stylesheet, the same stylesheet could be referenced by every package in the message.

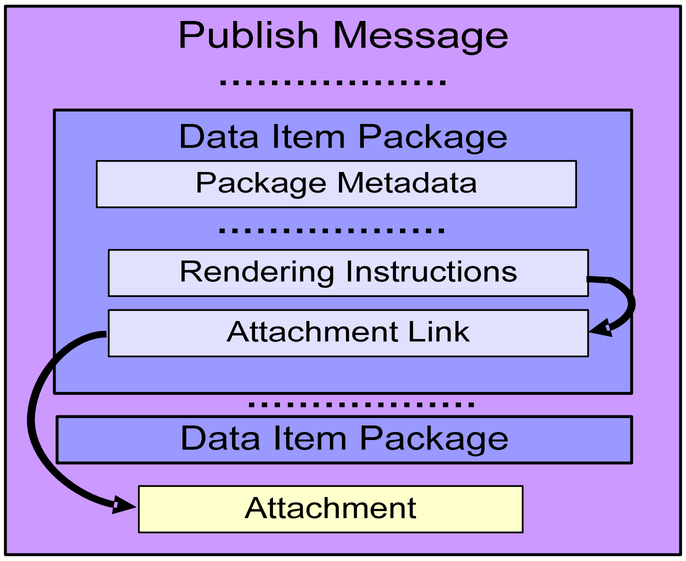


Figure 23. Rendering Attachment Linkage in Publish Message

The method of associating RenderingInstructions and an Attachment is also demonstrated in the XML fragment below.

```
<lexs:PublishMessage>
…
   <lexs:DataItemPackage>
…
      <lexs:AttachmentLink s:id="PAttachment2">
        <lexs:AttachmentURI>http://www.gtri.org/LEXS/example/XMLViewer.xsl</lexs:AttachmentURI>
        <lexs:AttachmentViewableIndicator>false</lexs:AttachmentViewableIndicator>
        <nc:BinaryDescriptionText>XML Stylesheet for XYZ packages</nc:BinaryDescriptionText>
      </lexs:AttachmentLink>

      <lexs:RenderingInstructions>
         <lexs:RenderingMethod>HTMLStyleSheet</lexs:RenderingMethod>
         <lexs:RenderingReference s:ref="PAttachment2"/>
         <lexs:RenderingDescription>Pretty prints a message</lexs:RenderingDescription>
      </lexs:RenderingInstructions>
   </lexs:DataItemPackage>

   <lexs:Attachment>
      <lexs:AttachmentURI>http://www.gtri.org/LEXS/examples/XMLViewer.xsl</lexs:AttachmentURI>
      <nc:Binary>
         <nc:BinaryBase64Object/> <!-- empty to shorten example -->
          <nc:BinaryDescriptionText>XML Stylesheet for XYZ packages</nc:BinaryDescriptionText>
          <nc:BinaryFormatID>application/xml+xslt</nc:BinaryFormatID>
          <nc:BinaryFormatStandardName>MIME</nc:BinaryFormatStandardName>
          <nc:BinaryCategoryText>Stylesheet</nc:BinaryCategoryText>
      </nc:Binary>
   </lexs:Attachment>
</lexs:PublishMessage>
```

5.3.2 SameAsDigestReference

As discussed previously, LEXS provides a layered architecture where the data contained in a Structured Payload builds upon the contents of the Digest. In order for applications to be able to connect XML elements in the Structured Payload with XML elements in the Digest, LEXS provides the SameAsDigestReference element to link elements.

As shown in the figure and example below, Structured Payload objects that build upon Digest contents are defined to include the SameAsDigestReference element. The SameAsDigestReference is a LEXS-specific version of a NIEM reference. A NIEM reference follows the definition of an XML *idref*, meaning the XML instance must include a matching *id* in order for the instance to be valid. Since the content of the Structure Payload is an instance based on a separate schema, any *idref* pointing to an *id* in the Digest will not validate. Therefore, LEXS defines the SameAsDigestReference as an XML NCName, essentially just a string, so that it can refer to an object outside XML scope of the Structured Payload.

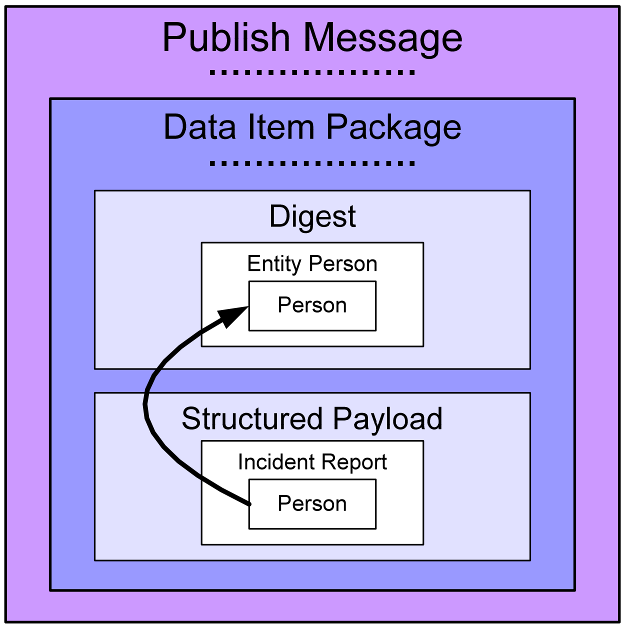


Figure 24. SameAsDigestReference in Publish Message

The following example shows a simple person in the Digest, with a Structured Payload that adds additional data for the person.

```
<lexs:PublishMessage>
…
  <lexs:DataItemPackage>
  …
    </lexs:Digest>
      <lexsdigest:EntityPerson>
        <lexsdigest:Person s:id="A">
          <nc:PersonName>
            <nc:PersonGivenName>John</nc:PersonGivenName>
            <nc:PersonSurName>Doe</nc:PersonSurName>
          </nc:PersonName>
        </lexsdigest:Person>
      </lexsdigest:EntityPerson>
    </lexs:Digest>
    <lexs:StructuredPayload>
      <lexs:StructuredPayloadMetadata>
        <lexs:CommunityURI>http://somewhere.gov/new-community</lexs:CommunityURI>
        <lexs:CommunityVersion>1.0</lexs:CommunityVersion>
      </lexs:StructuredPayloadMetadata>
      <new:IncidentReport>
        <new:Person>
          <nc:PersonHairStyleText>strawberry blond</nc:PersonHairStyleText>
          <lexslib:SameAsDigestReference lexslib:ref="A"/>
        </new:Person>
      </new:IncidentReport>
    </lexs:StructuredPayload>
  </lexs:DataItemPackage>
</lexs:PublishMessage>
```

As shown above, the lexsdigest:Person in the Digest has been assigned an *s:id* of “A”, and the new:Person in the Structured Payload builds upon the Digest person by adding hair style information and referring to the Digest person through use of the *lexslib:ref* of “A”.

There are cases where the SameAsDigestReference may be needed in other places in the Structured Payload. For example, the Digest allows multiple alternate names for a person. So if a Structured Payload adds content to an alternate name, such as a name prefix, the alternate name in the Structured Payload may require the SameAsDigestReference so that the name prefix can be unambiguously tied to the correct alternate name in the Digest; it is not required if the Digest only has one alternate name since in that case there is no ambiguity.. The example below shows a simple person in the Digest with two alternate names, with a Structured Payload that adds a name prefix to one of the alternate names.

```
<lexs:PublishMessage>
…
  <lexs:DataItemPackage>
  …
    </lexs:Digest>
      <lexsdigest:EntityPerson>
        <lexsdigest:Person s:id="Person1">
          <nc:PersonAlternateName s:id="Alt1">
            <nc:PersonGivenName>Billy</nc:PersonGivenName>
            <nc:PersonSurName>Smith</nc:PersonSurName>
          </nc:PersonAlternateName>
          <nc:PersonAlternateName s:id="Alt2">
            <nc:PersonGivenName>John</nc:PersonGivenName>
            <nc:PersonSurName>Doe</nc:PersonSurName>
          </nc:PersonAlternateName>
        </lexsdigest:Person>
      </lexsdigest:EntityPerson>
    </lexs:Digest>
    <lexs:StructuredPayload>
      <lexs:StructuredPayloadMetadata>
        <lexs:CommunityURI>http://somewhere.gov/new-community</lexs:CommunityURI>
        <lexs:CommunityVersion>1.0</lexs:CommunityVersion>
      </lexs:StructuredPayloadMetadata>
      <new:IncidentReport>
        < new:Person>
          < new:PersonAlternateName>
            <nc:PersonNamePrefixText>Sir</nc:PersonNamePrefixText>
            < new:PersonNameAugmentation>
              <lexslib:SameAsDigestReference lexslib:ref="Alt1"/>
            </ new:PersonNameAugmentation>
          </ new:PersonAlternateName>
          < new:PersonAugmentation>
            <lexslib:SameAsDigestReference lexslib:ref="Person1"/>
          </ new:PersonAugmentation>
        </ new:Person>
      </ new:IncidentReport>
    </lexs:StructuredPayload>
  </lexs:DataItemPackage>
</lexs:PublishMessage>
```

5.3.3 SameAsPayloadReference *(3.1.1 and later)*

As discussed previously, LEXS provides a layered architecture where the data contained in a Structured Payload can build upon the contents of another Structured Payload. In order for applications to be able to connect XML elements in one Structured Payload with XML elements in another Structured Payload, LEXS provides the SameAsPayloadReference element to link elements. The SameAsPayloadReference is very similar to the SameAsDigestReference described above, except that it contains two references: one to the Structured Payload and one to the element in the Structured Payload.

Note that the second Structured Payload may build upon an object in the Digest, in which case the second Structured Payload should use the SameAsDigestReference. So for example if the second Structured Payload builds upon a Person in the Digest to add a new identifier, the second Structured Payload should use the SameAsDigestReference to indicate that the new identifier builds upon the existing Digest Person. The SameAsPayloadReference is needed if the second Structured Payload builds on content that only exists in the first Structured Payload, or to unambiguously build on content in the first Structured Payload.

As shown in the figures and examples below, Structured Payload objects that build upon the contents of another Structured Payload are defined to include the SameAsPayloadReference element. The SameAsPayloadReference is a LEXS-specific version of a NIEM reference. A NIEM reference follows the definition of an XML *idref*, meaning the XML instance must include a matching *id* in order for the instance to be valid. Since the content of the Structure Payload is an instance based on a separate schema, any *idref* pointing to an *id* in another Structured Payload will not validate. Therefore, LEXS defines the SameAsPayloadReference attributes as XML NCName, essentially just a string, so that each can refer to an object outside XML scope of the Structured Payload.

The first example shows one Structured Payload building on another using the SameAsPayloadReference. In this example, the first Stuctured Payload adds an IRC Address object that does not correspond to any LEXS entity. The second Structured Payload builds upon the first Structured Payload's IRC Address to add the first use date.

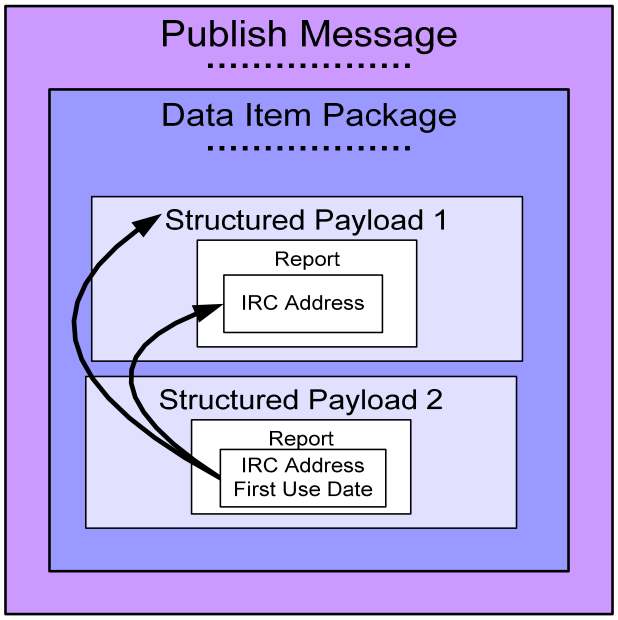


Figure 25. SameAsPayloadReference in Publish Message

```
<lexs:PublishMessage>
…
  <lexs:DataItemPackage>
  …

    <!--====================== Structured Payload #1 ======================-->
    <lexs:StructuredPayload s:id="SP1">
      <lexs:StructuredPayloadMetadata>
        <lexs:CommunityURI>http://somewhere.gov/new-community1</lexs:CommunityURI>
        <lexs:CommunityVersion>1.0</lexs:CommunityVersion>
      </lexs:StructuredPayloadMetadata>
      <new1:Report xmlns:new1="http://somewhere.gov/new-community1">
        <!-- IRC Address is an element with no corresponding Digest element, so no reference to Digest -->
        <new1:IRCAddress s:id="IRC1">
          <new1:UserName>xyz</new1:UserName>
          <new1:System>abc</new1:System>
        </new1:IRCAddress>
      </new1:Report>
    </lexs:StructuredPayload>
    <!--====================== Structured Payload #2 ======================-->
    <lexs:StructuredPayload>
      <lexs:StructuredPayloadMetadata>
        <lexs:CommunityURI>http://somewhere.gov/new-community2</lexs:CommunityURI>
        <lexs:CommunityVersion>1.0</lexs:CommunityVersion>
      </lexs:StructuredPayloadMetadata>
      <new2:Report xmlns:new2="http://somewhere.gov/new-community2">
        <new2:IRCAddress>
          <new2:FirstUseDate>2000-01-01</new2:FirstUseDate>
          <!-- Augmentation not used here assuming IRCAddress doesn't include any NIEM elements -->
          <lexslib:SameAsPayloadReference lexslib:ref="IRC1" lexslib:pref="SP1"/>
        </new2:IRCAddress>
      </new2:Report>
    </lexs:StructuredPayload>
  </lexs:DataItemPackage>
</lexs:PublishMessage>
```

The next example is more complicated, showing both cases where the SameAsPayloadReference may be necessary as well as a second Structured Payload building directly on the Digest. In this example, the Digest includes a person with multiple alternate names. The first Structured Payload builds upon the Digest AlternateName to add a name prefix. The second Structured Payload builds upon the name prefix in the first Structured Payload to indicate whether the name prefix is honorary and builds upon the person in the digest to add a hacker name.

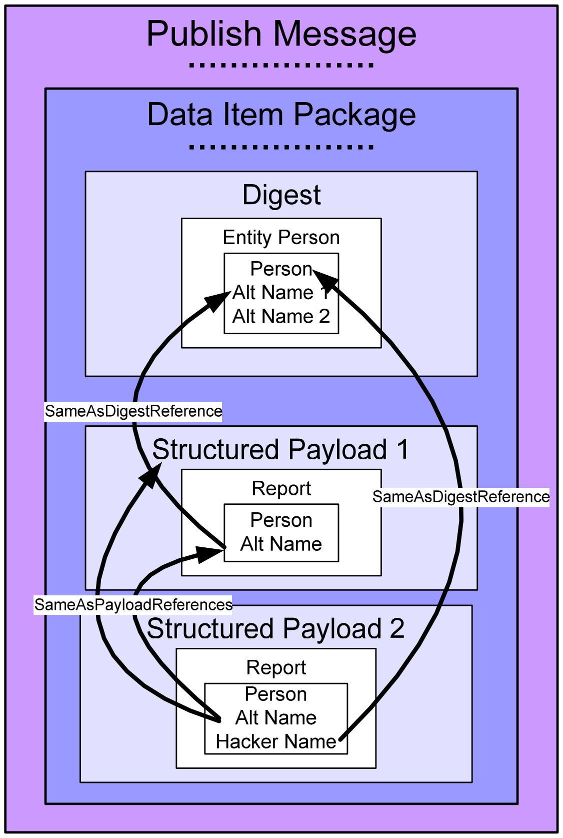


Figure 26. SameAsPayloadReference With SameAsDigestReference

```
<lexs:PublishMessage>
…
  <lexs:DataItemPackage>
  …
    <lexs:Digest>
    <!--======================  Person  ======================-->
      <lexsdigest:EntityPerson>
        <lexsdigest:Person s:id="Person1">
          <nc:PersonAlternateName s:id="AltName1">
            <nc:PersonGivenName>Bill</nc:PersonGivenName>
            <nc:PersonSurName>Smith</nc:PersonSurName>
          </nc:PersonAlternateName>
          <nc:PersonAlternateName s:id="AltName2">
            <nc:PersonGivenName>John</nc:PersonGivenName>
            <nc:PersonSurName>Doe</nc:PersonSurName>
          </nc:PersonAlternateName>
        </lexsdigest:Person>
      </lexsdigest:EntityPerson>
    </lexs:Digest>
    <!--====================== Structured Payload #1 ======================-->
    <lexs:StructuredPayload s:id="SP1">
      <lexs:StructuredPayloadMetadata>
        <lexs:CommunityURI>http://somewhere.gov/new-community1</lexs:CommunityURI>
        <lexs:CommunityVersion>1.0</lexs:CommunityVersion>
      </lexs:StructuredPayloadMetadata>
      <new1:Report xmlns:new1="http://somewhere.gov/new-community1">
        <new1:Person>
          <new1:PersonAlternateName s:id="SP1AltName1">
            <nc:PersonPrefixNameText>Dr.</nc:PersonPrefixNameText>
            <!-- Augmentation element used here since augmenting a NIEM Person element -->
            <new1:PersonNameAugmentation>
              <lexslib:SameAsDigestReference lexslib:ref="AltName1"/>
            </new1:PersonNameAugmentation>
          </new1:PersonAlternateName>
        </new1:Person>
      </new1:Report>
    </lexs:StructuredPayload>
    <!--====================== Structured Payload #2 ======================-->
    <lexs:StructuredPayload>
      <lexs:StructuredPayloadMetadata>
        <lexs:CommunityURI>http://somewhere.gov/new-community2</lexs:CommunityURI>
        <lexs:CommunityVersion>1.0</lexs:CommunityVersion>
      </lexs:StructuredPayloadMetadata>
      <new2:Report xmlns:new2="http://somewhere.gov/new-community2">
        <new2:Person>
          <new2:PersonAlternateName>
            <new2:PersonPrefixNameHonoraryIndicator>true</new2:PersonPrefixNameHonoraryIndicator>
            <!-- Augmentation not used here since PersonAlternateName doesn't include any NIEM elements -->
            <lexslib:SameAsPayloadReference lexslib:ref="SP1AltName1" lexslib:pref="SP1"/>
          </new2:PersonAlternateName>
          <new2:HackerName>The Doctor</new2:HackerName>
          <!-- Augmentation element used here assuming the new2:Person includes NIEM elements -->
          <new2:PersonAugmentation>
            <lexslib:SameAsDigestReference lexslib:ref="Person1"/>
          </new2:PersonAugmentation>
        </new2:Person>
      </new2:Report>
    </lexs:StructuredPayload>
  </lexs:DataItemPackage>
</lexs:PublishMessage>
```

5.3.4 SameAsPayloadAssociation

There may be cases where a community wants to use LEXS, but has already defined their own schema for data representation without utilizing LEXS concepts. Since Structured Payload contents are based on separate schemas, such a community can use their existing schemas to define the contents of the Structured Payload. In this case the Structured Payload will not build upon the contents of the Digest. However, the Digest can be populated with data elements that map to data elements in the Structured Payload so that LEXS consumers can still gain the benefits of the common understanding provided by the Digest.

To support this use case, LEXS provides the SameAsPayloadAssociation so that Digest contents can be linked to the corresponding Structured Payload contents, for example to indicate that this particular entity in the Digest is the same as this entity in the Structured Payload.

This association is included in the Digest, and is similar to any other association in that it links separate objects. As shown in the figure and example below, the association references an object in the Digest, a Structured Payload (since a package can have more than one), and an object in the Structured Payload.

However, in this case, the SameAsPayloadAssociation exists in the Digest and refers to an object in a Structured Payload. Since the Structured Payload data contents are contained in an element defined as an “xsd:any” in the LEXS schemas, XML validation will fail if the SameAsPayloadAssociation uses an XML *idref* to refer to an XML *id* in the Structured Payload. Therefore, LEXS defines each reference in the SameAsPayloadAssociation as an XML NCName, essentially just a string, so that it can refer to an object outside the XML scope of the Digest. The SameAsPayloadAssociation is defined in the LEXS library schema so that it can be used in Structured Payloads if desired, for example if a Structured Payload needs to point into another Structured Payload.

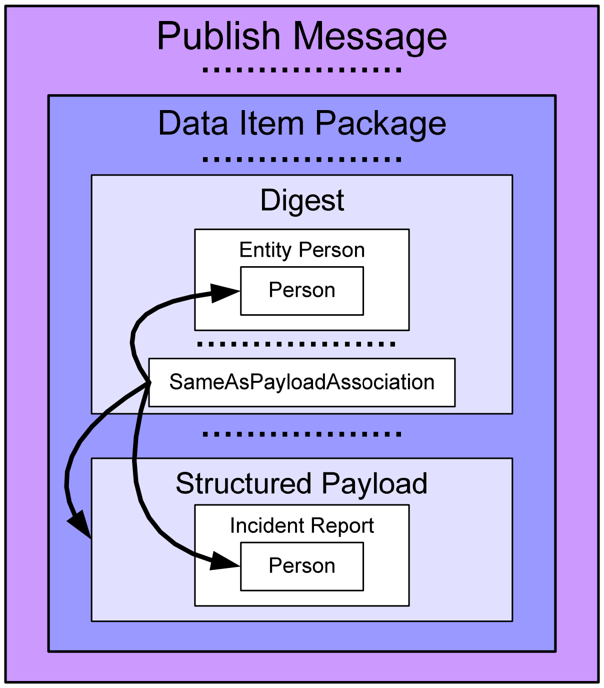


Figure 27. SameAsPayloadAssociation in Publish Message

The following example shows a simple person in the Digest, with a Structured Payload person that includes the Digest content plus some additional content.

```
<lexs:PublishMessage>
…
   <lexs:DataItemPackage>
   …
      </lexs:Digest>
         <lexsdigest:EntityPerson s:id="EPerson1">
            <lexsdigest:Person s:id="A">
               <nc:PersonName>
                  <nc:PersonGivenName>John</nc:PersonGivenName>
                  <nc:PersonSurName>Doe</nc:PersonSurName>
               </nc:PersonName>
            </lexsdigest:Person>
         </lexsdigest:EntityPerson>

         <lexsdigest:Associations>
            <lexslib:SameAsPayloadAssociation>
               <lexslib:ObjectReference s:ref="A"/>
               <lexslib:PayloadObjectReference lexslib:ref="SPA"/>
               <lexslib:PayloadReference s:ref="SP1"/>
            </lexslib:SameAsPayloadAssociation>
         </lexsdigest:Associations>
      </lexs:Digest>

      <lexs:StructuredPayload s:id="SP1">
         <lexs:StructuredPayloadMetadata>
            <lexs:CommunityURI>http://somewhere.gov/new-community</lexs:CommunityURI>
            <lexs:CommunityVersion>1.0</lexs:CommunityVersion>
         </lexs:StructuredPayloadMetadata>
```

```
            <new:IncidentReport>
              <new:Person s:id="SPA">
                <nc:PersonName>
                  <nc:PersonGivenName>John</nc:PersonGivenName>
                  <nc:PersonSurName>Doe</nc:PersonSurName>
                </nc:PersonName>
                <nc:PersonHairStyleText>strawberry blond</nc:PersonHairStyleText>
              </new:Person>
            </new:IncidentReport>
          </lexs:StructuredPayload>
        </lexs:DataItemPackage>
      </lexs:PublishMessage>
```

As shown above, the lexsdigest:Person in the Digest has been assigned an *s:id* of "A", and the new:Person in the Structured Payload includes the same content plus hair style information. The new:Person element has been assigned an s:id of "SPA", and the Structured Payload itself has been assigned the s:id of "SP1". The SameAsPayloadAssociation links the Digest and Structured Payload by referring to the object in the Digest with the *lexslib:ref* of "A", the Structured Payload with the *lexslib:ref* of "SP1", and the object in the Structured Payload with *lexslib:ref* of "SPA".

5.3.5 AttachmentLinkReference *(3.1.3 and later)*

The LEXS Digest includes associations to link LEXS Entities to Attachments, via an AttachmentLink. However, there may be cases where a community wishes to provide a more granular linkage between an Attachment and a Structured Payload object, or when there are objects in the Structured Payload that have no corresponding Entity in the Digest.. For example, a Structured Payload might include a person's physical features such as scars, marks and tattoos (SMT). In this case, a person might have many SMTs described in the Structured Payload with Attachments that are each an image of a single SMT, but since the LEXS EntityPersonSMTImageAssociation links the Attachment to the Entity, there is no way to determine which image goes with which SMT.

LEXS includes the capability to link individual AttachmentLink elements to objects in the Structured Payload. This is accomplished through the use of the lexslib:AttachmentLinkReference, which is incorporated into Structured Payload schemas in the same fashion as lexslib:SameAsDigestReference elements. (Note that the lexslib:AttachmentLinkReference discussed here is distinct from the lexsdigest:AttachmentLinkReference utilized by the Digest associations. The lexslib:AttachmentLinkReference is of type "lexlib:ReferenceType" which allows an element in a Structured Payload to refer to an element outside the Structured Payload, while the lexsdigest:AttachmentLinkReference cannot do that since it is of type "s:ReferenceType".)

Attachments should be associated with LEXS Entities using the various Entity to AttachmentLink associations (such as EntityPersonSMTImageAsscoation or EntitySupportingDocumentationAssociation) in order to provide consistent context for Attachments. lexslib:AttachmentLinkReference provides additional granularity but should not be used as the only linkage for an Attachment.

The AttachmentLinkReference mechanism is of use in Publish messages so that Structured Payloads can provide detailed context for attachments, as well as in SR exchanges so that users can better determine which attachments they want to see in order to avoid problems of forcing a user to review all attachments or to determine what attachments to retrieve based on description alone.

A Structured Payload object may refer to more than one AttachmentLink, for example if there are multiple images of a specific SMT. Multiple Structured Payload objects may also refer to a single AttachmentLink, for example if one image includes two SMTs.

The diagram below shows a Digest element with an EntityPerson who has two SMT images, linked to the Entity via two EntityPersonSMTImageAssociations. The Structured Payload includes a person that builds upon the Digest Person, incorporating two PhysicalFeature elements. The Structured Payload PhysicalFeature elements are in turn linked to specific AttachmentLink elements in order to indicate which image is for which PhysicalFeature.

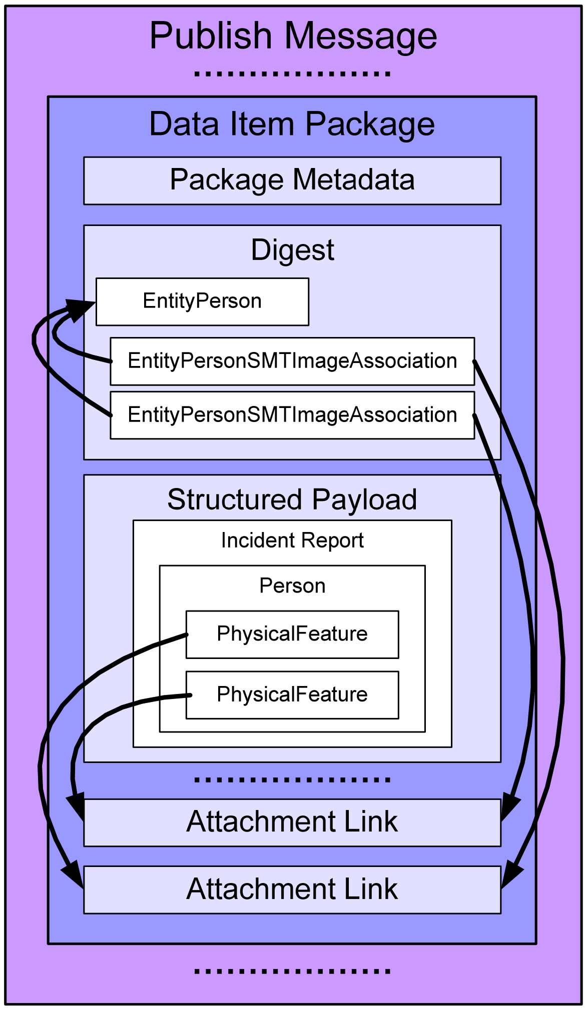


Figure 28. AttachmentLinkReference in Publish Message

The following example shows a simple person in the Digest with two EntityPersonSMTImageAssociations connecting the person with two AttachmentLink elements (the Attachments have been left out to simplify the diagram). The Structured Payload includes a Person with two physical features, each linked to a specific AttachmentLink.

```
<lexs:PublishMessage>
…
  <lexs:DataItemPackage>
…
    <lexs:Digest>
      <lexsdigest:EntityPerson s:id="EArr1">
        <lexsdigest:Person s:id="Person1">
          <nc:PersonName>
            <nc:PersonGivenName>Bubba</nc:PersonGivenName>
            <nc:PersonMiddleName>Ray</nc:PersonMiddleName>
            <nc:PersonSurName>Johnson</nc:PersonSurName>
          </nc:PersonName>
        </lexsdigest:Person>
      </lexsdigest:EntityPerson>

      <lexsdigest:Associations>
        <lexsdigest:EntityPersonSMTImageAssociation>
          <lexsdigest:EntityReference s:ref="EArr1"/>
          <lexsdigest:AttachmentLinkReference s:ref="PAttachment1"/>
        </lexsdigest:EntityPersonSMTImageAssociation>
        <lexsdigest:EntityPersonSMTImageAssociation>
          <lexsdigest:EntityReference s:ref="EArr1"/>
          <lexsdigest:AttachmentLinkReference s:ref="PAttachment2"/>
        </lexsdigest:EntityPersonSMTImageAssociation>
      </lexsdigest:Associations>

    </lexs:Digest>

    <lexs:StructuredPayload>
      <lexs:StructuredPayloadMetadata>
        <lexs:CommunityURI>http://somewhere.gov/new-community</lexs:CommunityURI>
        <lexs:CommunityVersion>1.0</lexs:CommunityVersion>
      </lexs:StructuredPayloadMetadata>
      <new:IncidentReport>
        <new:Person>
          <new:PersonPhysicalFeature>
            <nc:PhysicalFeatureDescriptionText>lightning tattoo, right arm</nc:PhysicalFeatureDescriptionText>
            <new:PhysicalFeatureAugmentation>
              <lexslib:AttachmentLinkReference lexslib:ref="PAttachment1"/>
            </new:PhysicalFeatureAugmentation>
          </new:PersonPhysicalFeature>
          <new:PersonPhysicalFeature>
            <nc:PhysicalFeatureDescriptionText>cross tattoo, left arm</nc:PhysicalFeatureDescriptionText>
            <new:PhysicalFeatureAugmentation>
              <lexslib:AttachmentLinkReference lexslib:ref="PAttachment2"/>
            </new:PhysicalFeatureAugmentation>
          </new:PersonPhysicalFeature>
          <new:PersonAugmentation>
            <lexslib:SameAsDigestReference lexslib:ref="Person1"/>
          </new:PersonAugmentation>
        </new:Person>
      </new:IncidentReport>
    </lexs:StructuredPayload>

    <lexs:AttachmentLink s:id="PAttachment1">
```

```
      <lexs:AttachmentURI>http://www.xyz.org/LEXS/examples/smt1.jpg</lexs:AttachmentURI>
      <lexs:AttachmentViewableIndicator>true</lexs:AttachmentViewableIndicator>
      <nc:BinaryDescriptionText>Lightning tattoo image for Bubba Johnson</nc:BinaryDescriptionText>
    </lexs:AttachmentLink>
    <lexs:AttachmentLink s:id="PAttachment2">
      <lexs:AttachmentURI>http://www.xyz.org/LEXS/examples/smt2.jpg</lexs:AttachmentURI>
      <lexs:AttachmentViewableIndicator>true</lexs:AttachmentViewableIndicator>
      <nc:BinaryDescriptionText>Cross tattoo image for Bubba Johnson</nc:BinaryDescriptionText>
    </lexs:AttachmentLink>
  </lexs:DataItemPackage>

  <!-- Attachments left out to simplify example -->

</lexs:PublishMessage>
```

5.3.6 Indicating Entities Are or May Be the Same

If two different packages each contain an EntityPerson with an identical value for Social Security Number, these two entities might represent the same real-world person. When entities have attributes that match, these entities (and thus data items) are only linked implicitly. Business rules can be used to determine when entities from separate data items can be determined to be referring to the same real-world object—for use in a link analysis chart, etc.

To preclude consuming systems from having to re-discover (or fail to discover) these links, LEXS provides explicit linking mechanisms that allow data sources to make this local information available globally. In addition, many source systems store data in resolved/normalized tables, where the creation of packages involves de-normalizing (duplicating) entity data across multiple packages.

An issue with the explicit linking of entities between packages can be seen in the publication model. There may be ambiguity as to whether the source system intends for the linked entities to be treated as distinct for display and update (within their data items), or as the *same* entity (i.e. if updated in one package, must be updated everywhere by a consuming system). Thus there are two distinct kinds of explicit linking:

- Where the source system effectively stores a single record for an entity referenced in multiple packages, wishes to have the entity data be shown identically in those multiple data items by the receiver, and wishes to be able to update the entity in all data items without having to send updates for each and every one in multiple packages. Some of the DOJ component data sources are good examples of this.
- Where the source system effectively stores multiple records for an entity (or multiple variants of the information for that entity) and needs/wishes to preserve distinct views of the entity in distinct data items, but somehow knows/believes/thinks that these multiple entity records all refer to the same real-world object and would like to make this apparent to the receiving system and its users. In addition, the source system is willing/wants to update the different entity records individually. Systems that perform entity resolution on data contributed from multiple sources are good examples of this.

As a result, two separate linking schemes are provided in LEXS: one to handle the case where entity records are normalized into a single master table and an update to that entity means an update to every data item that references that entity (hard-linked entities), and one to handle the case where the entities are linked together somehow, but entity information is stored separately for each data item and may be different in terms of detail or actually conflicting and thus must be updated independently (soft-linked entities).

The first (hard linking) uses the concept of a "source id." All entities with the same source id should be (effectively) collapsed into a single entity record in the receiving system, such that any update of the entity within any single package updates it for each and every data item in which it is referenced. As a corollary, all representations of the entity within a single submission (snapshot) should be identical; otherwise, the results are not deterministic.

The second (soft linking) uses the concept of a "logical id." All entity representations with the same logical id—but distinct or omitted source ids—will be preserved as separate records in the receiving system, and thus the receiving system will be able to show different views of the entity for different data items. Updates are interpreted individually—the entity is only updated in association with the updated data item—no other soft-linked entities are modified.

Note that in both cases, the entities have to be of the same kind, for example both people, and the information must be from the same data source. In other words, consumers must not treat matching ids from different data sources as explicit links, nor should they treat matching ids for different kinds of entities (such as a person and vehicle) as explicit links.

The table below represents four person entities, each in different data items provided by the same data source. Based on the name and social security number alone these items could describe the same person or different people, however, the data source provides explicit links through the use of source id and logical id

	A	**B**	**C**	**D**
Name	John Doe	John Doe	John Dione	John Doe
SSN	087-67-8945	087-67-8945	087-67-8935	unknown
Source ID	ATF-SID-765	ATF-SID-765	ATF-SID-789	ATF-SID-888
Logical ID	ATF-LID-700	ATF-LID-700	ATF-LID-700	ATF-LID-901

The Source ID and Logical ID values above indicate the following:

- Person A and person B are considered to be the same person by the data source since the SourceIDText value is the same for both.
- Person A and person C are believed or suspected to be the same person since the LogicalIDText values match. The data source does not know they are the same since the SourceIDText values differ.
- Person A and person D are different people, at least as far as this data source is concerned since neither the SourceIDText nor LogicalIDText values match.

Hard links, soft links, and implicit links are all between entities in ***distinct*** packages, and these packages may even be in different messages and/or message containers. Using the person entities from the table above, the links tie the John Doe entity in package A to the John Doe entity in package B. When these John Doe entities are hard-linked together, then they are by definition always identical—updated in sync and displayed identically for all data items. But the entities in the package do not exist separately from the packages and they are not published or updated separately. Operationally, data sources still update a data item by providing a complete updated data item; it is just that other data items may be implicitly updated as well if they contain a hard-linked entity.

LEXS 3.1 uses NIEM's Metadata construct to implement linking using the elements SourceIDText and LogicalIDText. Metadata elements are contained in LEXS Entity elements, such as a lexsdigest:EntityPerson. A specific element, such as a lexsdigest:Person, states what metadata applies to it using the s:metadata attribute.

The example below shows the case where a single data source provides three different packages that include John Doe entities. The John Does represented by Person A and Person B are hard-linked to show that the data source considers these people to be the same record. The John Does represented by Person A and Person C are soft-linked to show that the data source treats these people as separate records but suspects or believes them to be the person.

```
<!-- Person A-->
<lexsdigest:EntityPerson>
    <lexsdigest:Metadata s:id="MA">
        <nc:SourceIDText>ATF-SID-765</nc:SourceIDText>
        <lexsdigest:LogicalIDText>ATF-LID-700</lexsdigest:LogicalIDText>
    </lexsdigest:Metadata>
    <lexsdigest:Person s:id="A" s:metadata="MA">
        <nc:PersonName>
            <nc:PersonGivenName>John</nc:PersonGivenName>
            <nc:PersonSurName>Doe</nc:PersonSurName>
        </nc:PersonName>
        <nc:PersonSSNIdentification>
             <nc:IdentificationID>087-67-8945</nc:IdentificationID>
        </nc:PersonSSNIdentification>
    </lexsdigest:Person>
</lexsdigest:EntityPerson>

<!-- Person B-->
<lexsdigest:EntityPerson>
    <lexsdigest:Metadata s:id="MB">
        <nc:SourceIDText>ATF-SID-765</nc:SourceIDText>
        <lexsdigest:LogicalIDText>ATF-LID-700</lexsdigest:LogicalIDText>
    </lexsdigest:Metadata>
    <lexsdigest:Person s:id="B" s:metadata="MB">
        <nc:PersonName>
            <nc:PersonGivenName>John</nc:PersonGivenName>
            <nc:PersonSurName>Doe</nc:PersonSurName>
        </nc:PersonName>
        <nc:PersonSSNIdentification>
            <nc:IdentificationID>087-67-8945</nc:IdentificationID>
        </nc:PersonSSNIdentification>
    </lexsdigest:Person>
</lexsdigest:EntityPerson>
```

```
<!-- Person C-->
<lexsdigest:EntityPerson>
    <lexsdigest:Metadata s:id="MC">
        <nc:SourceIDText>ATF-SID-789</nc:SourceIDText>
        <lexsdigest:LogicalIDText>ATF-LID-700</lexsdigest:LogicalIDText>
    </lexsdigest:Metadata>
    <lexsdigest:Person s:id="C" s:metadata="MC">
        <nc:PersonName>
            <nc:PersonGivenName>John</nc:PersonGivenName>
            <nc:PersonSurName>Doe</nc:PersonSurName>
        </nc:PersonName>
        <nc:PersonSSNIdentification>
            <nc:IdentificationID>087-67-8935</nc:IdentificationID>
        </nc:PersonSSNIdentification>
    </lexsdigest:Person>
</lexsdigest:EntityPerson>
```

6 Building Queries

LEXS supports both structured and unstructured searches. An unstructured search, or text search, is used to request a text value in any context. A text search does not constrain the search to a particular element. A structured search request specifies both the value and the element with which the value should be associated. LEXS provides a broad range of query capabilities that are described below.

6.1 Text Query

LEXS SR supports an unstructured search capability, including matching all terms, any terms, or exact phrase. Logical operators AND as well as OR may be specified in the query.

A search utilizing AND between all terms is equivalent to a search on “all terms”. A search utilizing OR between all terms is equivalent to a search on “any terms”. The ability to mix and match AND with OR, for example to perform a search on “(Tom OR Steve) AND Corvette”, provides great flexibility.

Some service providers may support the concept of “all terms” and/or “any terms”, but not allow mixing and matching of AND with OR. In this case, the service provider can respond if the search criteria included nothing but AND or nothing but OR, and would return an error if the search criteria includes both.

If AND is mixed with OR, parentheses must be included in order to state precedence clearly. This is a business rule and is not enforced by the LEXS schemas.

AND and OR do not have to be capitalized as shown here, but this capitalization does help with readability and is encouraged.

Unstructured search strings may request exact phrases by including the phrases in quotes, for example: “give me the money”. Phrases may be mingled with single words, such as a search on
Smith AND “give me the money”.

If a ‘*’ is included in an exact phrase, the ‘*’ is interpreted literally. In other words, the exact phrase “Tom Smith*” would search for Tom Smith with a literal ‘*’ after the ‘h’.

Note that stop words may affect the results depending on the technology and techniques used by the service provider. For example, the word “the” may be ignored by some implementations, meaning that a search on the phrase “give me the money” may also match occurrences of “give me a money”. LEXS cannot control the underlying implementations, so users will need to be made aware of this issue.

Wildcards may be used when specifying elements or attributes to search for, using the ‘*’ character to indicate 0-n characters. The wildcard character may appear at the beginning, end, or middle of a string. Wildcards may appear in more than one place, such as at the beginning and end, or beginning and middle, or even beginning, middle, and end.

The search criteria for unstructured searches are supplied in a string. No validation of the contents of the string is enforced by the LEXS schemas. The following provides a sample of acceptable unstructured search criteria:

Tom AND Smith AND Corvette
(Tom AND Smith AND Corvette)
Tom OR Smith OR Corvette
(Tom OR Steve) AND Corvette
Smith*
Tom AND Smith*
“Tom Smith”
Smith AND “give me the money”

6.2 Structured Query

LEXS SR supports a structured search capability, including matching partial, exact, range, and multiple values. Note that there is an implicit AND between different entities and between different fields of an entity. There is no support for an OR between entities or fields.

Below an overview of structured search capabilities and principles is followed by a discussion of how the structured query mechanism works and examples for how each capability is handled.

LEXS SR Structured Search Capabilities

1. Query can be against LEXS Digest and Structured Payload elements and attributes.
   - Query may be stated against objects and roles listed in the LEXS Digest, including attributes.
   - Query may be stated against data listed in the LEXS Structured Payload, including attributes.
   - Service provider does not have to support search on every query field listed in the query; service provider makes “best effort” to answer queries. For example, a search specifies Person First Name, Person Last Name, and Person SSN; however, service provider does not support search by SSN. Therefore, the service provider should execute search by First Name and Last Name, dropping SSN field.
   - Service providers should ignore unknown fields within a Structured Payload query, as well as entire Structured Payloads that are not understood. Therefore, LEXS SR query statement metadata provides information on how many query terms there are for each Structured Payload in order for the service provider to be able to calculate an accurate match score.
   - Service provider should be able to provide information on what query fields were answered, or ignored, during search. If a requested search criteria for the field was not supported, but the field was still searched using different criteria, the service provider should report this information back to query requestor. (For example query specified “fuzzy” search for PersonSurName, however, exact search was performed for PersonSurName).
2. Query supports partial searches on text fields through the use of the wildcard character “*”.

3. Query supports intelligent search (or “fuzzy” search) functions on individual search fields.
4. Query supports exact dates and date ranges; includes support for “open-ended” ranges, such as a date before 01-01-1970.
5. Query supports exact numbers and numeric value ranges; includes support for “open-ended” ranges, such as a weight larger than 200.
6. Query supports multiple value matches for single field, for example last name Smith or Jones.
7. Query allows specifying multiple entities and uses implicit “AND” between different entities and between different fields of an entity. For example, a person with last name Smith and a vehicle with model Corvette. Note that queries are geared towards finding data items that match the criteria. So when multiple entities are included, this means the search is for data items that contain ALL the indicated entities. If the query includes two people, that means the search is for data items that include BOTH people.

Structured Query Statements
LEXS SR structured queries are stated using the LEXS Digest and Structured Payload entities by supplying field values for selected elements. This allows implementers to leverage tools and constructs developed for processing XML instances that utilize the well-defined Digest and Structured Payload constructs, and provides a straightforward means of validating the contents of a structured query. Simple queries look like responses or publish requests.

An example of a structured query is shown below; where the query is looking for a person with the first name Jane, a fuzzy match on last name Doe, and strawberry blond hair.

```
<lexs:StructuredQuery>
    <lexs:DigestQueryStatement>
        <lexs:DigestQueryField>
            <lexsdigest:EntityPerson>
                <lexsdigest:Person>
                    <nc:PersonName>
                        <nc:PersonGivenName>Jane</nc:PersonGivenName>
                    </nc:PersonName>
                </lexsdigest:Person>
            </lexsdigest:EntityPerson>
        </lexs:DigestQueryField>
        <lexs:QueryMatch>exact</lexs:QueryMatch>
    </lexs:DigestQueryStatement>
    <lexs:DigestQueryStatement>
        <lexs:DigestQueryField>
            <lexsdigest:EntityPerson>
                <lexsdigest:Person>
                    <nc:PersonName>
                        <nc:PersonSurName>Doe</nc:PersonSurName>
                    </nc:PersonName>
                </lexsdigest:Person>
            </lexsdigest:EntityPerson>
        </lexs:DigestQueryField>
        <lexs:QueryMatch>fuzzy</lexs:QueryMatch>
    </lexs:DigestQueryStatement>
    <lexs:StucturedPayloadQueryStatement>
        <lexs:StructuredPayloadMetadata>
            <lexs:CommunityURI>http://somewhere.gov/new-community</lexs:CommunityURI>
            <lexs:CommunityVersion>1.0</lexs:CommunityVersion>
        </lexs:StructuredPayloadMetadata>
        <lexs:StructuredPayloadQueryField>
            <new:Person>
                <nc:PersonHairStyleText>strawberry blond</nc:PersonHairStyleText>
            </new:Person>
        </lexs:StructuredPayloadQueryField>
        <lexs:QueryMatch>exact</lexs:QueryMatch>
    </lexs:StucturedPayloadQueryStatement>
<lexs:StructuredQuery>
```

Only one query field is supplied per DigestQueryStatement or StructuredPayloadQueryStatement element. So for example, if the query contains two query fields, such as PersonSurName and PersonGivenName, there must be two DigestQueryStatement elements in the query.

Note that the DigestQueryField and StructuredPayloadQueryField contents can be validated using the Digest and Structured Payload schemas.

The mandatory element “lexsdigest:QueryMatch” indicates the type of match to perform, and includes the values “fuzzy”, “exact”, “lt” (less than), “le” (less than or equal to), “gt” (greater than), “ge” (greater than or equal to), and “wildcard” (value contains one or more wildcard characters).

The remainder of this section provides examples of how queries are stated based on the various capabilities.

1) *Query can be against LEXS Digest and Structured Payload elements and attributes.* Query statements may include Digest and Structured Payload sections exactly as with responses or publish requests. A query could search on just Digest elements, or just Structured Payload elements, or a combination of both. Attributes may be included as well.

   A service provider can identify the supported search fields by examining each DigestQueryStatement and StructuredPayloadStatement block. The service provider can easily make a decision whether or not to support the requested match function by checking for a QueryMatch element.

   The query below searches for a person with a last name Doe from the Digest and hair style "strawberry blond" from a Structured Payload.

```
<lexs:StructuredQuery>
    <lexs:DigestQueryStatement>
        <lexs:DigestQueryField>
            <lexsdigest:EntityPerson>
                <lexsdigest:Person>
                    <nc:PersonName>
                        <nc:PersonSurName>Doe</nc:PersonSurName>
                    </nc:PersonName>
                </lexsdigest:Person>
            </lexsdigest:EntityPerson>
        </lexs:DigestQueryField>
        <lexs:QueryMatch>exact</lexs:QueryMatch>
    </lexs:DigestQueryStatement>
    <lexs:StucturedPayloadQueryStatement>
        <lexs:StructuredPayloadMetadata>
            <lexs:CommunityURI>http://somewhere.gov/new-community</lexs:CommunityURI>
            <lexs:CommunityVersion>1.0</lexs:CommunityVersion>
        </lexs:StructuredPayloadMetadata>
        <lexs:StructuredPayloadQueryField>
            <new:Person>
                <nc:PersonHairStyleText>strawberry blond</nc:PersonHairStyleText>
            </new:Person>
        </lexs:StructuredPayloadQueryField>
        <lexs:QueryMatch>exact</lexs:QueryMatch>
    </lexs:StucturedPayloadQueryStatement>
</lexs:StructuredQuery>
```

   The query below searches for an item with a value of $100. The currency of U.S. dollars is provided as an attribute in NIEM. The item value element and the currency attribute are supplied in separate DigestQueryStatement elements. Note the use of "xsi:nil" attribute for the DigestQueryStatement that specifies the currency code attribute.

```
<lexs:StructuredQuery>
   <lexs:DigestQueryStatement>
      <lexs:DigestQueryField>
         <lexsdigest:EntityTangibleItem>
            <nc:TangibleItem>
               <nc:ItemValue>
                  <nc:ItemValueAmount nc:currencyCode="USD" xsi:nil="true"/>
               </nc:ItemValue>
            </nc:TangibleItem>
         </lexsdigest:EntityTangibleItem>
      </lexs:DigestQueryField>
      <lexs:QueryMatch>exact</lexs:QueryMatch>
   </lexs:DigestQueryStatement>
   <lexs:DigestQueryStatement>
      <lexs:DigestQueryField>
         <lexsdigest:EntityTangibleItem>
            <nc:TangibleItem>
               <nc:ItemValue>
                  <nc:ItemValueAmount>100</nc:ItemValueAmount>
               </nc:ItemValue>
            </nc:TangibleItem>
         </lexsdigest:EntityTangibleItem>
      </lexs:DigestQueryField>
      <lexs:QueryMatch>lt</lexs:QueryMatch>
   </lexs:DigestQueryStatement>
</lexs:StructuredQuery>
```

2) *Query supports partial searches on text fields through the use of the wildcard character "*".*
Wildcards are supplied in the value for the element itself, and the QueryMatch element has the value "wildcard". When the QueryMatch element is populated with "wildcard", service providers know that the value has one or more wildcard characters in it and to do whatever processing is necessary to handle wildcards.

The query below searches for a person with a first name that matches the "Jan*" pattern, such as Jane, Janice, Janet

```
<lexs:StructuredQuery>
    <lexs:DigestQueryStatement>
        <lexs:DigestQueryField>
            <lexsdigest:EntityPerson>
                <lexsdigest:Person>
                    <nc:PersonName>
                        <nc:PersonGivenName>Jan*</nc:PersonGivenName>
                    </nc:PersonName>
                </lexsdigest:Person>
            </lexsdigest:EntityPerson>
        </lexs:DigestQueryField>
        <lexs:QueryMatch>wildcard</lexs:QueryMatch>
    </lexs:DigestQueryStatement>
</lexs:StructuredQuery>
```

3) *Query supports intelligent search (or "fuzzy" search) functions.*
Fuzzy searches are stated using the QueryMatch element with the value "fuzzy".

The query below searches for a person using a fuzzy match on the last name Doe.

```
<lexs:StructuredQuery>
    <lexs:DigestQueryStatement>
        <lexs:DigestQueryField>
            <lexsdigest:EntityPerson>
                <lexsdigest:Person>
                    <nc:PersonName>
                        <nc:PersonSurName>Doe</nc:PersonSurName>
                    </nc:PersonName>
                </lexsdigest:Person>
            </lexsdigest:EntityPerson>
        </lexs:DigestQueryField>
        <lexs:QueryMatch>fuzzy</lexs:QueryMatch>
    </lexs:DigestQueryStatement>
</lexs:StructuredQuery>
```

4) *Query supports exact dates and date ranges.*
Ranges on dates are stated using the QueryMatch element with values “lt” (less than), “le” (less than or equal to), “gt” (greater than), or “ge” (greater than or equal to). Open ended ranges, such as a date later than 1-1-1970, are stated by supplying a single value and the appropriate QueryMatch value.

The query below searches for a person whose birthday is between 1-1-1974 and 1-1-1980.

```
<lexs:StructuredQuery>
    <lexs:DigestQueryStatement>
        <lexs:DigestQueryField>
            <lexsdigest:EntityPerson>
                <lexsdigest:Person>
                    <nc:PersonBirthDate>1974-01-01</nc:PersonBirthDate>
                </lexsdigest:Person>
            </lexsdigest:EntityPerson>
        </lexs:DigestQueryField>
        <lexs:QMatch>ge</lexs:QMatch>
    </lexs:DigestQueryStatement>
    <lexs:DigestQueryStatement>
        <lexs:DigestQueryField>
            <lexsdigest:EntityPerson>
                <lexsdigest:Person>
                    <nc:PersonBirthDate>1980-01-01</nc:PersonBirthDate>
                </lexsdigest:Person>
            </lexsdigest:EntityPerson>
        </lexs:DigestQueryField>
        <lexs:QueryMatch>le</lexs:QueryMatch>
    </lexs:DigestQueryStatement>
</lexs:StructuredQuery>
```

5) *Query supports exact numbers and numeric value ranges.*
NIEM 2.0 defines a MeasureType that includes both a “point” value as well as a range. This allows most numeric ranges to be stated using NIEM terms rather than having to state them using a LEXS construct. There are a few numeric fields that do not include a range such as

engine quantity. Even though these fields are unlikely to be searched as a range, the “lt”, “le”, “gt”, and “ge” terms can be used if necessary. However, the NIEM range values must be used to specify ranges if available, with the “lt”, “le”, “gt”, and “ge” terms reserved for cases where the range cannot be stated using NIEM elements. Note that both the NIEM range as well as the “lt”, “le”, “gt”, and “ge” allow the specification of an open-ended range, such as a person who weighs over 200 pounds, by just specifying one “end” of the range.

The example below shows a query for a person between 180 and 200 pounds.

```
<lexs:StructuredQuery>
    <lexs:DigestQueryStatement>
        <lexs:DigestQueryField>
            <lexsdigest:EntityPerson>
                <lexsdigest:Person>
                    <nc:PersonWeightMeasure>
                        <nc:MeasureRangeValue>
                            <nc:RangeMinimumValue>180</nc:RangeMinimumValue>
                        </nc:MeasureRangeValue>
                    </nc:PersonWeightMeasure>
                </lexsdigest:Person>
            </lexsdigest:EntityPerson>
        </lexs:DigestQueryField>
        <lexs:QueryMatch>exact</lexs:QueryMatch>
    </lexs:DigestQueryStatement>
    <lexs:DigestQueryStatement>
         <lexs:DigestQueryField>
            <lexsdigest:EntityPerson>
                <lexsdigest:Person>
                    <nc:PersonWeightMeasure>
                        <nc:MeasureRangeValue>
                            <nc:RangeMaximumValue>200</nc:RangeMaximumValue>
                        </nc:MeasureRangeValue>
                    </nc:PersonWeightMeasure>
                </lexsdigest:Person>
            </lexsdigest:EntityPerson>
        </lexs:DigestQueryField>
        <lexs:QueryMatch>exact</lexs:QueryMatch>
    </lexs:DigestQueryStatement>
    <lexs:DigestQueryStatement>
        <lexs:DigestQueryField>
            <lexsdigest:EntityPerson>
                <lexsdigest:Person>
                    <nc:PersonWeightMeasure>
                        <nc:WeightUnitCode>LBR</nc:WeightUnitCode>
                    </nc:PersonWeightMeasure>
                </lexsdigest:Person>
            </lexsdigest:EntityPerson>
        </lexs:DigestQueryField>
        <lexs:QueryMatch>exact</lexs:QueryMatch>
    </lexs:DigestQueryStatement>
</lexs:StructuredQuery>
```

The example below shows a query for a vessel with more than one engine using an open-ended range.

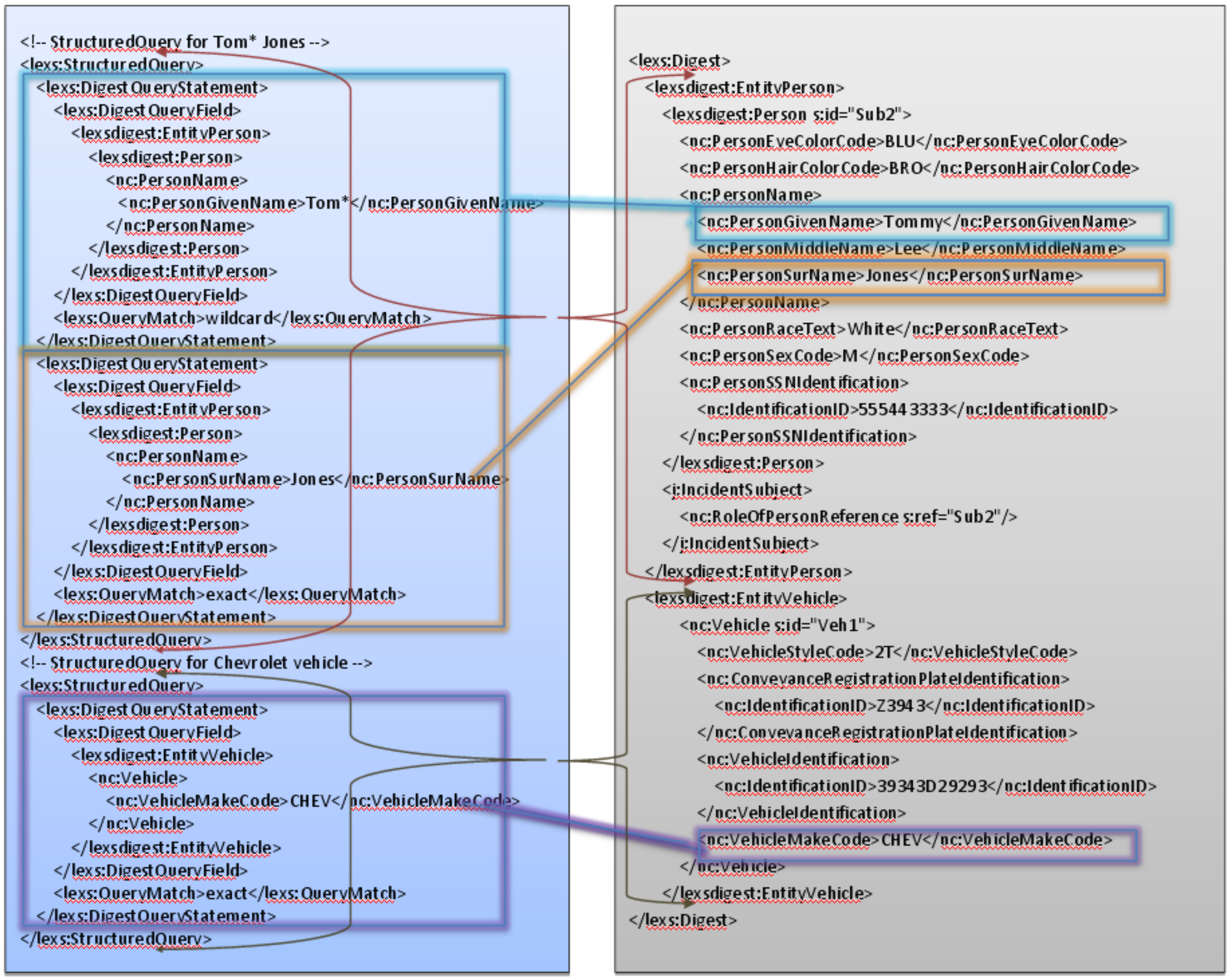
```
<lexs:StructuredQuery>
    <lexs:DigestQueryStatement>
        <lexs:DigestQueryField>
            <lexsdigest:EntityVessel>
                <nc:Vessel>
                    <nc:ConveyanceEngineQuantity>1</nc:ConveyanceEngineQuantity>
                </nc:Vessel>
            </lexsdigest:EntityVessel>
        </lexs:DigestQueryField>
        <lexs:QMatch>gt</lexs:QMatch>
    </lexs:DigestQueryStatement>
</lexs:StructuredQuery>
```

6) *Query supports multiple value matches for single field.*
According to the LEXS Digest NIEM restriction schema, lexsdigest:Person can have only one nc:PersonSurName; therefore additional values for the same query field are put into separate DigestQueryStatement blocks. The example below shows a query for a Person with first name Jennifer, Janice, or Jane.

```
<lexs:StructuredQuery>
    <lexs:DigestQueryStatement>
        <lexs:DigestQueryField>
            <lexsdigest:EntityPerson>
                <lexsdigest:Person>
                     <nc:PersonName>
                         <nc:PersonGivenName>Jennifer</nc:PersonGivenName>
                      </nc:PersonName>
                  </lexsdigest:Person>
            </lexsdigest:EntityPerson>
        </lexs:DigestQueryField>
        <lexs:QueryMatch>exact</lexs:QueryMatch>
    </lexs:DigestQueryStatement>
    <lexs:DigestQueryStatement>
        <lexs:DigestQueryField>
            <lexsdigest:EntityPerson>
                <lexsdigest:Person>
                    <nc:PersonName>
                        <nc:PersonGivenName>Janice</nc:PersonGivenName>
                    </nc:PersonName>
                </lexsdigest:Person>
            </lexsdigest:EntityPerson>
        </lexs:DigestQueryField>
        <lexs:QueryMatch>exact</lexs:QueryMatch>
    </lexs:DigestQueryStatement>
    <lexs:DigestQueryStatement>
        <lexs:DigestQueryField>
            <lexsdigest:EntityPerson>
                <lexsdigest:Person>
                    <nc:PersonName>
                        <nc:PersonGivenName>Jane</nc:PersonGivenName>
                    </nc:PersonName>
                </lexsdigest:Person>
            </lexsdigest:EntityPerson>
        </lexs:DigestQueryField>
        <lexs:QueryMatch>exact</lexs:QueryMatch>
    </lexs:DigestQueryStatement>
</lexs:StructuredQuery>
```

7) *Query allows specifying multiple entity objects and uses implicit "AND" between different entities and between different fields of an entity*.
Queries assume AND between all fields and values. Each StructuredQuery element groups query fields for a single entity. (Note: Prior to LEXS 3.1.1 only one StructuredQuery element was allowed.) So if the query is for data items that include one specific person, all the person query fields would be contained in a single StructuredQuery element. However, if the query is for data items that include two different people (such as a search for data items that include both John Smith and Mary Doe), then there would be two StructuredQuery elements; one for all the query fields for the first person (John Smith) and one for all the query fields for the second person (Mary Doe).

The diagram below shows a query for data items that include two different kinds of entities, a person entity matching the criteria of a first name starting with "Tom" and last name "Jones" plus a vehicle entity made by Chevrolet. The diagram show how the query fields correspond to fields in a possible match where the left side of the diagram shows the query, while the right side shows a Digest fragment from possible match. The query contains two StructuredQuery elements since the query is against two different entities, and the first StructuredQuery element contains two DigestQueryStatement elements since the query is against a first and last name for the person.

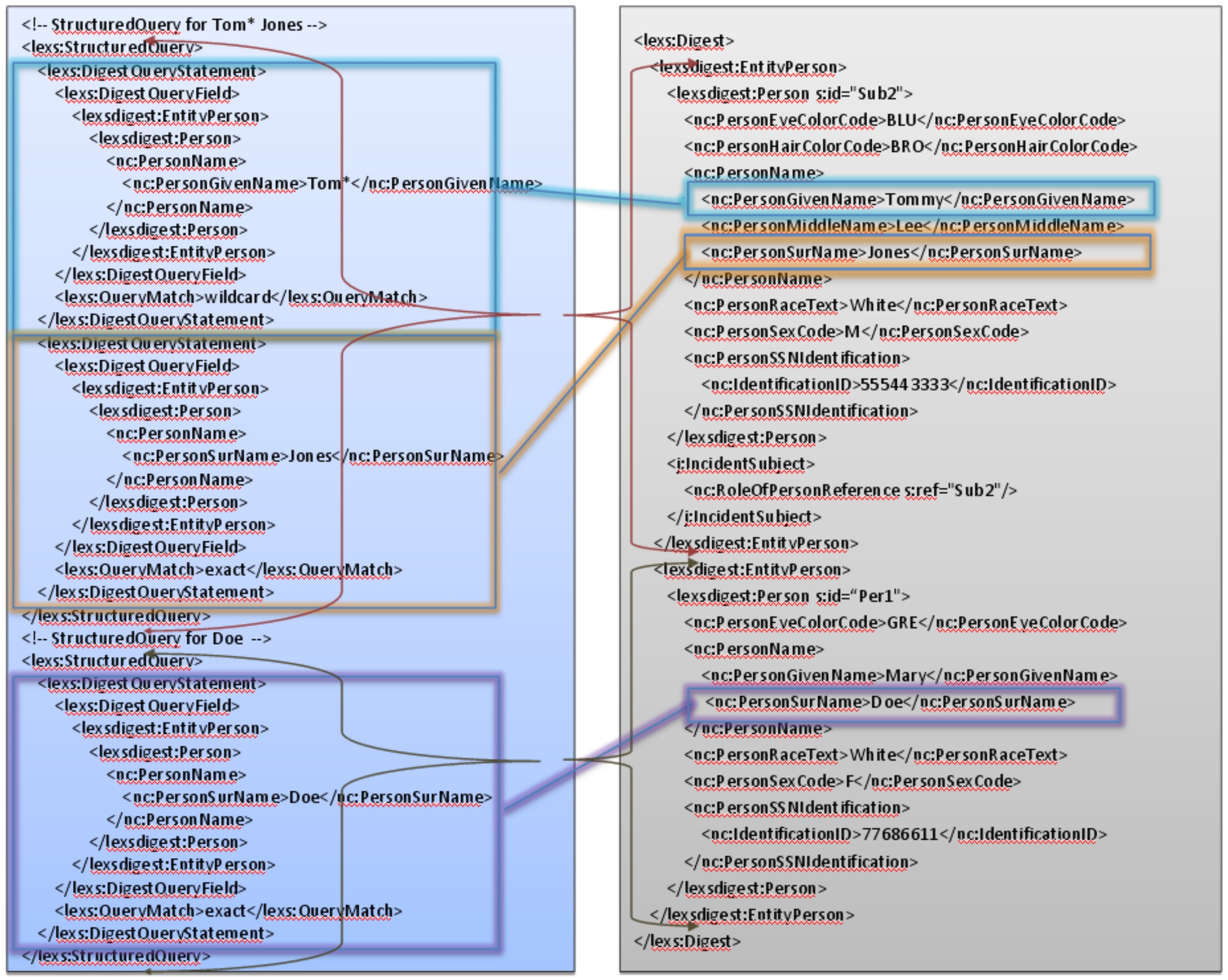


```
<!-- StructuredQuery for Tom* Jones -->
<lexs:StructuredQuery>
  <lexs:DigestQueryStatement>
    <lexs:DigestQueryField>
      <lexsdigest:EntityPerson>
        <lexsdigest:Person>
          <nc:PersonName>
            <nc:PersonGivenName>Tom*</nc:PersonGivenName>
          </nc:PersonName>
        </lexsdigest:Person>
      </lexsdigest:EntityPerson>
    </lexs:DigestQueryField>
    <lexs:QueryMatch>wildcard</lexs:QueryMatch>
  </lexs:DigestQueryStatement>
  <lexs:DigestQueryStatement>
    <lexs:DigestQueryField>
      <lexsdigest:EntityPerson>
        <lexsdigest:Person>
          <nc:PersonName>
            <nc:PersonSurName>Jones</nc:PersonSurName>
          </nc:PersonName>
        </lexsdigest:Person>
      </lexsdigest:EntityPerson>
    </lexs:DigestQueryField>
    <lexs:QueryMatch>exact</lexs:QueryMatch>
  </lexs:DigestQueryStatement>
</lexs:StructuredQuery>
<!-- StructuredQuery for Chevrolet vehicle -->
<lexs:StructuredQuery>
  <lexs:DigestQueryStatement>
    <lexs:DigestQueryField>
      <lexsdigest:EntityVehicle>
        <nc:Vehicle>
          <nc:VehicleMakeCode>CHEV</nc:VehicleMakeCode>
        </nc:Vehicle>
      </lexsdigest:EntityVehicle>
    </lexs:DigestQueryField>
    <lexs:QueryMatch>exact</lexs:QueryMatch>
  </lexs:DigestQueryStatement>
</lexs:StructuredQuery>
```

```
<lexs:Digest>
  <lexsdigest:EntityPerson>
    <lexsdigest:Person s:id="Sub2">
      <nc:PersonEyeColorCode>BLU</nc:PersonEyeColorCode>
      <nc:PersonHairColorCode>BRO</nc:PersonHairColorCode>
      <nc:PersonName>
        <nc:PersonGivenName>Tommy</nc:PersonGivenName>
        <nc:PersonMiddleName>Lee</nc:PersonMiddleName>
        <nc:PersonSurName>Jones</nc:PersonSurName>
      </nc:PersonName>
      <nc:PersonRaceText>White</nc:PersonRaceText>
      <nc:PersonSexCode>M</nc:PersonSexCode>
      <nc:PersonSSNIdentification>
        <nc:IdentificationID>555443333</nc:IdentificationID>
      </nc:PersonSSNIdentification>
    </lexsdigest:Person>
    <j:IncidentSubject>
      <nc:RoleOfPersonReference s:ref="Sub2"/>
    </j:IncidentSubject>
  </lexsdigest:EntityPerson>
  <lexsdigest:EntityVehicle>
    <nc:Vehicle s:id="Veh1">
      <nc:VehicleStyleCode>2T</nc:VehicleStyleCode>
      <nc:ConveyanceRegistrationPlateIdentification>
        <nc:IdentificationID>Z3943</nc:IdentificationID>
      </nc:ConveyanceRegistrationPlateIdentification>
      <nc:VehicleIdentification>
        <nc:IdentificationID>39343D29293</nc:IdentificationID>
      </nc:VehicleIdentification>
      <nc:VehicleMakeCode>CHEV</nc:VehicleMakeCode>
    </nc:Vehicle>
  </lexsdigest:EntityVehicle>
</lexs:Digest>
```

Figure 29. Query Against Elements of Two Different Entity Types

The diagram below shows a query for data items that include two of the same kind of entity, a person entity matching the criteria of a first name starting with “Tom” and last name “Jones” plus a second person entity matching the criteria of last name “Doe”. The diagram show how the query fields correspond to fields in a possible match where the left side of the diagram shows the query, while the right side shows a Digest fragment from possible match. The query contains two StructuredQuery elements since the query is against two entities, where each StructuredQuery element contains two DigestQueryStatement elements since the query is against a first and last name for the person.



```
<!-- StructuredQuery for Tom* Jones -->
<lexs:StructuredQuery>
  <lexs:DigestQueryStatement>
    <lexs:DigestQueryField>
      <lexsdigest:EntityPerson>
        <lexsdigest:Person>
          <nc:PersonName>
            <nc:PersonGivenName>Tom*</nc:PersonGivenName>
          </nc:PersonName>
        </lexsdigest:Person>
      </lexsdigest:EntityPerson>
    </lexs:DigestQueryField>
    <lexs:QueryMatch>wildcard</lexs:QueryMatch>
  </lexs:DigestQueryStatement>
  <lexs:DigestQueryStatement>
    <lexs:DigestQueryField>
      <lexsdigest:EntityPerson>
        <lexsdigest:Person>
          <nc:PersonName>
            <nc:PersonSurName>Jones</nc:PersonSurName>
          </nc:PersonName>
        </lexsdigest:Person>
      </lexsdigest:EntityPerson>
    </lexs:DigestQueryField>
    <lexs:QueryMatch>exact</lexs:QueryMatch>
  </lexs:DigestQueryStatement>
</lexs:StructuredQuery>
<!-- StructuredQuery for Doe -->
<lexs:StructuredQuery>
  <lexs:DigestQueryStatement>
    <lexs:DigestQueryField>
      <lexsdigest:EntityPerson>
        <lexsdigest:Person>
          <nc:PersonName>
            <nc:PersonSurName>Doe</nc:PersonSurName>
          </nc:PersonName>
        </lexsdigest:Person>
      </lexsdigest:EntityPerson>
    </lexs:DigestQueryField>
    <lexs:QueryMatch>exact</lexs:QueryMatch>
  </lexs:DigestQueryStatement>
</lexs:StructuredQuery>
```

```
<lexs:Digest>
  <lexsdigest:EntityPerson>
    <lexsdigest:Person s:id="Sub2">
      <nc:PersonEyeColorCode>BLU</nc:PersonEyeColorCode>
      <nc:PersonHairColorCode>BRO</nc:PersonHairColorCode>
      <nc:PersonName>
        <nc:PersonGivenName>Tommy</nc:PersonGivenName>
        <nc:PersonMiddleName>Lee</nc:PersonMiddleName>
        <nc:PersonSurName>Jones</nc:PersonSurName>
      </nc:PersonName>
      <nc:PersonRaceText>White</nc:PersonRaceText>
      <nc:PersonSexCode>M</nc:PersonSexCode>
      <nc:PersonSSNIdentification>
        <nc:IdentificationID>55544 3333</nc:IdentificationID>
      </nc:PersonSSNIdentification>
    </lexsdigest:Person>
    <j:IncidentSubject>
      <nc:RoleOfPersonReference s:ref="Sub2"/>
    </j:IncidentSubject>
  </lexsdigest:EntityPerson>
  <lexsdigest:EntityPerson>
    <lexsdigest:Person s:id="Per1">
      <nc:PersonEyeColorCode>GRE</nc:PersonEyeColorCode>
      <nc:PersonName>
        <nc:PersonGivenName>Mary</nc:PersonGivenName>
        <nc:PersonSurName>Doe</nc:PersonSurName>
      </nc:PersonName>
      <nc:PersonRaceText>White</nc:PersonRaceText>
      <nc:PersonSexCode>F</nc:PersonSexCode>
      <nc:PersonSSNIdentification>
        <nc:IdentificationID>77686611</nc:IdentificationID>
      </nc:PersonSSNIdentification>
    </lexsdigest:Person>
  </lexsdigest:EntityPerson>
</lexs:Digest>
```

Figure 30. Query Again Elements of the Same Entity Type

Processing and Composition Notes

- If multiple values are supplied for a single field, the range operators “ge”, “gt”, “lt”, and “le” must not be mixed with “exact”, “fuzzy” and “wildcard”.
- If multiple values are supplied for a single field, the operators “exact”, “fuzzy” and “wildcard” can be mixed, for example searching on a person with last name “Jones” using “exact”, “Smith” using “fuzzy”, and “Williams*” using “wildcard”.
- If multiple values are supplied for a single field, the range operators imply an “AND” (e.g. greater than 50 AND less than 100), while the “exact”, “fuzzy” and “wildcard” operators imply an “OR” among the values (e.g. last name exactly Smith OR exactly Jones).

7 Working with Attachments and Rendering Instructions

7.1 Scenarios for Interacting with a LEXS SR Server

This section describes three possible scenarios for interacting with a LEXS SR Server. It provides a recommended choreography when using the LEXS SR operations described in detail earlier in this document. It is not intended to be exclusive; additional scenarios may be supported by LEXS SR clients.

The scenario descriptions pay particular attention to attachments, which require coordination to ensure that they are managed and rendered appropriately.

7.1.1 Scenario 1: Retrieving entire data items

Scenario 1, depicted in Figure 31, describes a typical interaction between a user performing a search, receiving a list of results, and selecting individual data items of interest. These data items may have attachments such as images and rendering instructions.

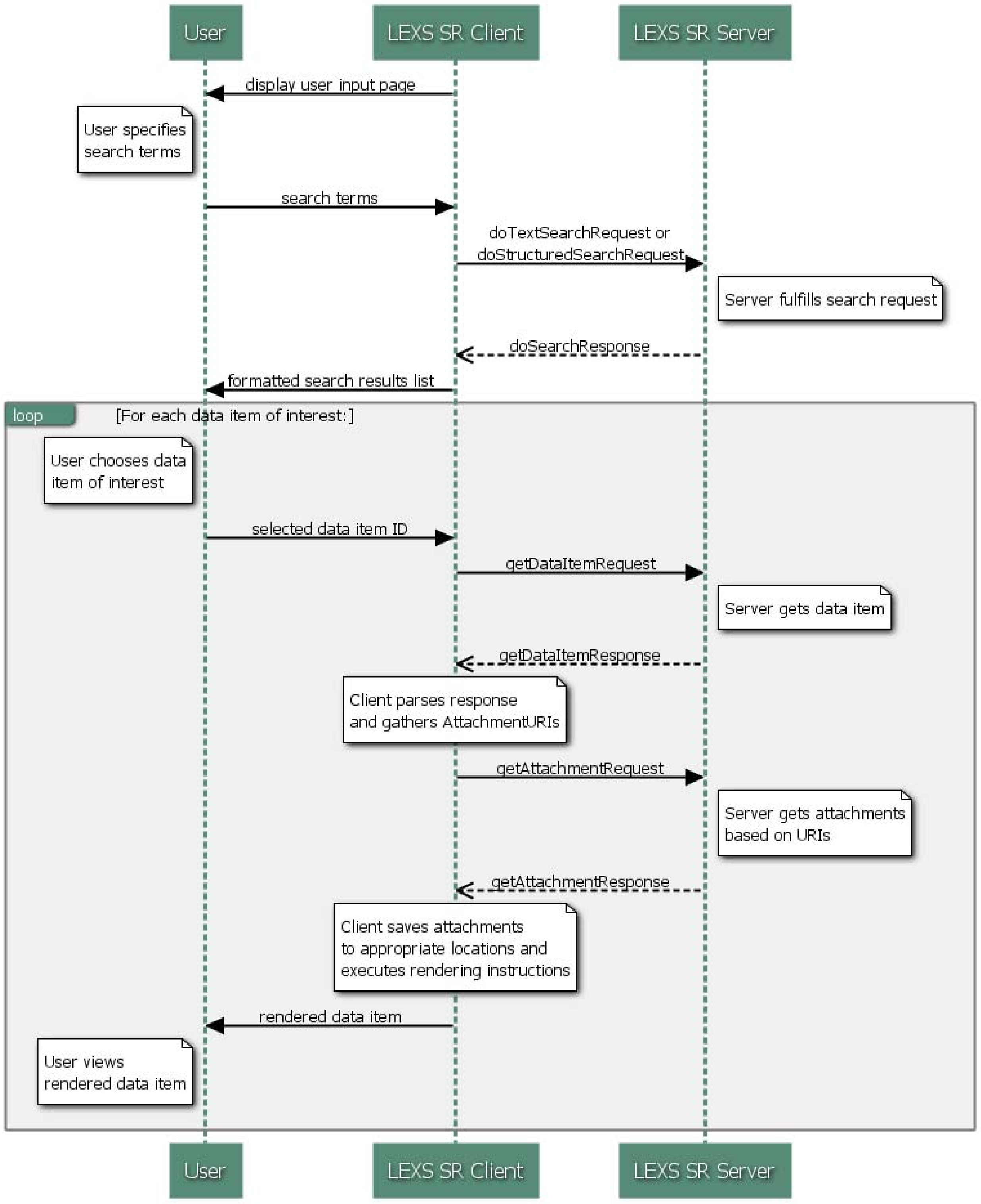

User
LEXS SR Client
LEXS SR Server
display user input page
User specifies search terms
search terms
doTextSearchRequest or doStructuredSearchRequest
Server fulfills search request
doSearchResponse
formatted search results list
loop
[For each data item of interest:]
User chooses data item of interest
selected data item ID
getDataItemRequest
Server gets data item
getDataItemResponse
Client parses response and gathers AttachmentURIs
getAttachmentRequest
Server gets attachments based on URIs
getAttachmentResponse
Client saves attachments to appropriate locations and executes rendering instructions
rendered data item
User views rendered data item
User
LEXS SR Client
LEXS SR Server

Figure 31: Scenario 1: Retrieving entire data items

Scenario 1 consists of the following steps:

1. The LEXS SR Client collects the search terms from the user and assembles an XML document that is passed to either the doTextSearch or doStructuredSearch operation.
2. The LEXS SR Server fulfills the search request and returns a doSearchResponse message that contains summaries of data items that match the search terms.
3. The LEXS SR Client presents these search results to the user and allows them to select one or more items to see more detail.
4. For each item that the user selects, the LEXS SR Client assembles an XML document that is passed to the getDataItem operation.
5. The LEXS SR Server fulfills the request and returns a getDataItemResponse message with the full details of that data item. The message includes AttachmentLink elements that specify the URIs of the attachments that apply to that data item.
6. The LEXS SR Client parses the getDataItemResponse message(s) to retrieve a list of attachment URIs for the selected data item(s). It may not need to retrieve all attachments for a data item if not all of them are intended to be displayed. For example, the Client may choose only show certain kinds of images (such as mug shots), and provide links to other attachments (such as tattoos) that are downloaded separately when the link is clicked.

   It may not need to retrieve some attachments if they have previously been cached, as described in section 7.3. It then assembles an XML document that is passed to the getAttachment operation. It lists all of the necessary attachment URIs by specifying multiple "lexs:AttachmentURI" children of "lexssr:getAttachmentRequest".
7. The LEXS SR Server fulfills the request and returns a getAttachmentResponse message that includes the attachments embedded in "lexs:Attachment" elements in base-64 binary format.
8. The LEXS SR Client parses the getAttachmentResponse messages and stores the attachment files in a designated location. Special attention must be paid to the location and identifiers used to retrieve these attachments, as described in section 7.2.
9. The LEXS SR Client identifies which attachment constitutes the rendering instructions and executes it to generate the rendered data item (including attachments) to the user.

7.1.2 Scenario 2: Retrieving attachments based on search results

Scenario 2, depicted in Figure 32, involves retrieving attachments based on information in the search results, rather than downloading the entire detail of a data item. A typical example is when a search returns a list of people, and the user wants to take a quick look at the mug shot of a person rather than download the entire data item.

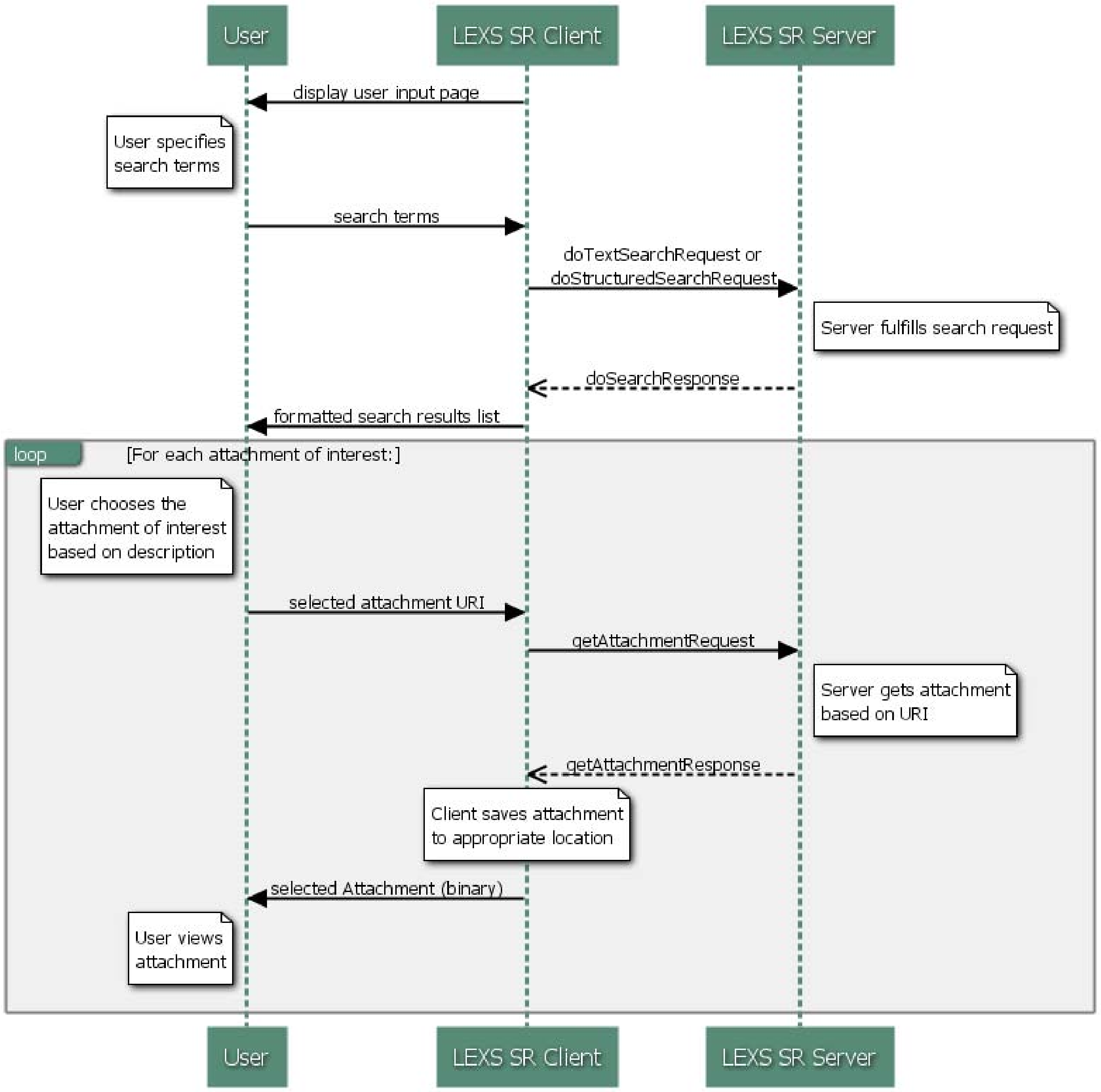

User
LEXS SR Client
LEXS SR Server
display user input page
User specifies search terms
search terms
doTextSearchRequest or doStructuredSearchRequest
Server fulfills search request
doSearchResponse
formatted search results list
loop
[For each attachment of interest:]
User chooses the attachment of interest based on description
selected attachment URI
getAttachmentRequest
Server gets attachment based on URI
getAttachmentResponse
Client saves attachment to appropriate location
selected Attachment (binary)
User views attachment
User
LEXS SR Client
LEXS SR Server

Figure 32: Scenario 2: Retrieving attachments based on search results

Scenario 2 consists of the following steps:

1. The LEXS SR Client collects the search terms from the user and assembles an XML document that is passed to either the doTextSearch or doStructuredSearch operation.
2. The LEXS SR Server fulfills the search request and returns a doSearchResponse message contain summaries of data items that match the search terms.
3. The LEXS SR Client presents these search results to the user and provides them links to select one or more attachments to view for each data item. Viewable attachments are those whose "lexs:AttachmentLink/lexs:AttachmentViewableIndicator" element contains the value "true". In the case that there is more than one attachment for a data item, the

value of the "lexs:AttachmentLink/lexs:BinaryDescriptionText" element can be presented to the user to distinguish between them.

Alternatively, the link(s) can be limited to a specific attachment type by parsing the digest (if available) in the doSearchResponse message. For example, if the Client should only display links to mug shots (and not other kinds of images), it can parse the doSearchResponse message to find the attachments referenced from "lexs:EntityPersonFacialImageAssociation" elements for a particular data item.

4. For each attachment that the user selects, the LEXS SR Client determines whether it needs to retrieve that attachment from the server. It may not need to retrieve the attachment if it has previously been cached, as described in section 7.3. If it needs to be retrieved, the LEXS SR Client assembles an XML document containing the appropriate attachment URI, that is passed to the getAttachment operation.
5. The LEXS SR Server fulfills the request and returns a getAttachmentResponse message that includes the attachment embedded in "lexs:Attachment" element in base-64 binary format.
6. The LEXS SR Client parses the getAttachmentResponse messages and stores the attachment file in a designated location.
7. The LEXS SR Client presents the attachment to the user.

7.1.3 Scenario 3: Retrieving attachments with search results

Scenario 3, depicted in Figure 33, involves displaying selected attachments alongside the data item information in the search results. For example, the Client might display a list of people, and show the mug shots alongside the name and other summary information about that person. This is similar to Scenario 2, except that the user is not required to click on a link before viewing the attachment. This requires the Client to retrieve the attachments before displaying the results to the user.

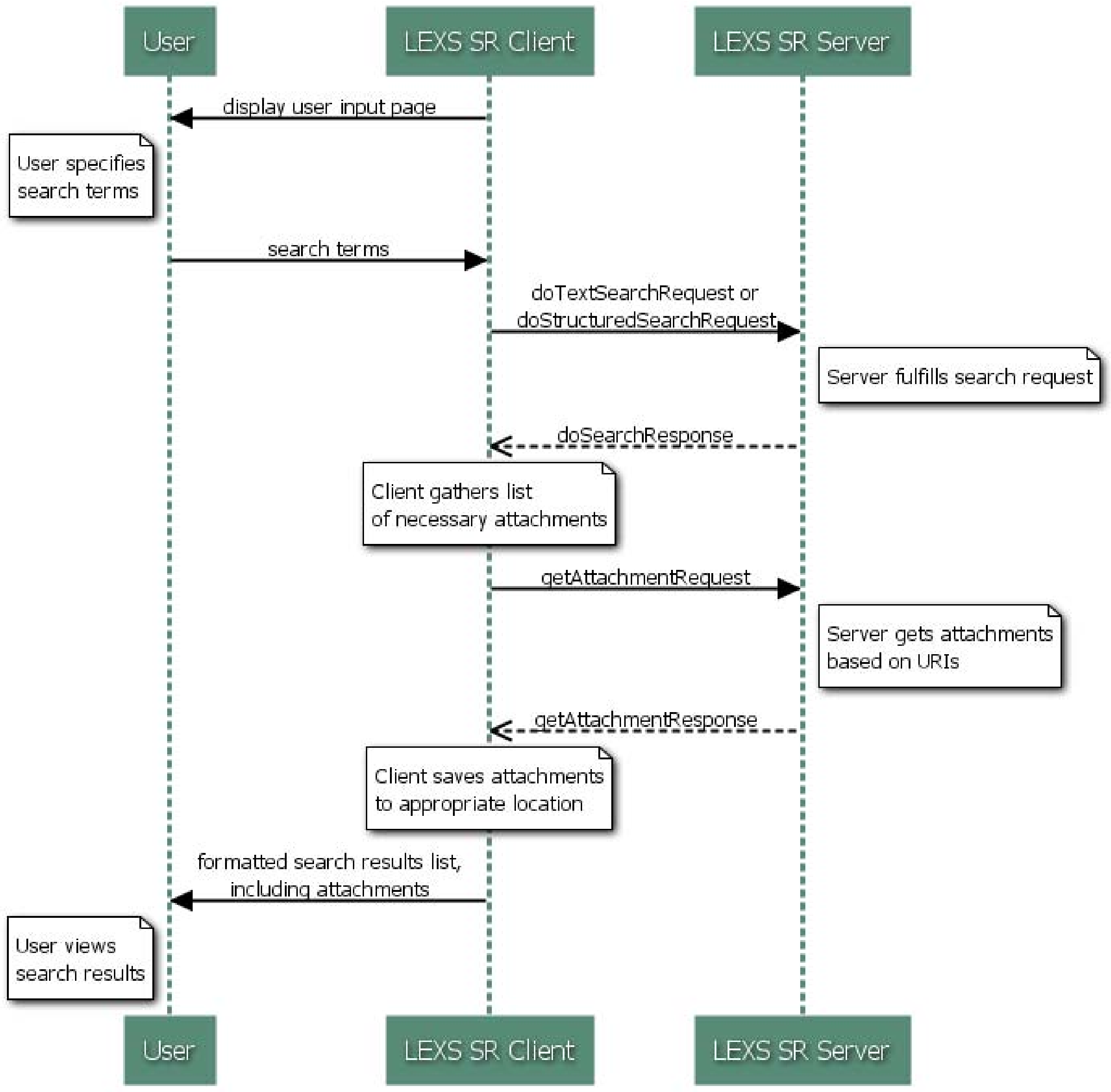

User
LEXS SR Client
LEXS SR Server
display user input page
User specifies search terms
search terms
doTextSearchRequest or doStructuredSearchRequest
Server fulfills search request
doSearchResponse
Client gathers list of necessary attachments
getAttachmentRequest
Server gets attachments based on URIs
getAttachmentResponse
Client saves attachments to appropriate location
formatted search results list, including attachments
User views search results
User
LEXS SR Client
LEXS SR Server

Figure 33: Scenario 3: Retrieving attachments with search results

Scenario 3 consists of the following steps:

1. The LEXS SR Client collects the search terms from the user and assembles an XML document that is passed to either the doTextSearch or doStructuredSearch operation.
2. The LEXS SR Server fulfills the search request and returns a doSearchResponse message contain summaries of data items that match the search terms.
3. The LEXS SR Client parses the doSearchResponse message(s) to retrieve a list of appropriate attachment URIs for the data item(s) in the search results. The Client may not need to retrieve all attachments for the data item if some have previously been cached, as described in section 7.3. It then assembles an XML document that is passed to the getAttachment operation. It lists all of the necessary attachment URIs by specifying multiple "lexs:AttachmentURI" children of "lexssr:getAttachmentRequest".

In most cases, the Client will retrieve attachments of a particular type. For example, if the Client will only display mug shots (and not other kinds of images) next to the person search results, it can parse the doSearchResponse message to find the attachments referenced from "lexs:EntityPersonFacialImageAssociation" elements.

4. The LEXS SR Server fulfills the request and returns a getAttachmentResponse message that includes the attachment embedded in "lexs:Attachment" element in base-64 binary format.
5. The LEXS SR Client parses the getAttachmentResponse messages and stores the attachment file in a designated location.
6. The LEXS SR Client presents the search results, including the relevant attachments, to the user.

7.2 Attachment URLs

Attachments in LEXS are identified by URIs, which are simply unique strings. The URI does not indicate the actual location (URL) of the attachment. When a LEXS SR Client that supports rendering instructions (hereafter referred to as a "Rendering Client") performs a search, it will retrieve the attachments associated with a data item and make a copy of them locally (whether it is in a database, file system or other mechanism). Therefore, the actual location (URL) of the attachments is determined by the Rendering Client when it retrieves it.

However, some XSLT rendering instructions need to be able to generate HTML that refers to attachments (most commonly, images) in order to display them or provide links to them. The XSLT stylesheet has no inherent knowledge of where the Rendering Client will place the retrieved attachments.

Therefore, a common approach is needed for how Rendering Client applications should turn the URIs into URLs, so that:

- the authors of rendering instructions know how to refer to the images in the HTML generated by the XSLT stylesheets, and
- the implementers of LEXS search systems know where to put the attachments so that they are referenced correctly.

7.2.1 Escaping the Attachment URIs

Because attachment URIs are unique within a data source, the URI itself can serve as an identifier for the attachment. However, the URI must be escaped in such a way that it can be included in the URL used to retrieve the attachment.

For example, the attachment URI:

```
http://www.usdoj.gov/DOJ/JABS/LEXS/PHOTO/Per12346/001.jpg
```

can uniquely identify the attachment, but it cannot be used as a parameter within a URL, as in:

```
http://www.clientapp.net/getAttachment?partner=dhs.icepick&uri=http://www.usd
oj.gov/DOJ/JABS/LEXS/PHOTO/Per12346/001.jpg
```

because the colon and slashes in the attachment URI have special meaning that will affect the interpretation of the entire URL.

Therefore, attachment URIs must be escaped using the standard URI escaping rules defined in section 2 of *RFC 3968: Uniform Resource Identifier (URI): Generic Syntax*.[3] For example, the attachment URI:

```
http://www.usdoj.gov/DOJ/JABS/LEXS/PHOTO/Per12346/001.jpg
```

becomes:

```
http%3A%2F%2Fwww.usdoj.gov%2FDOJ%2FJABS%2FLEXS%2FPHOTO%2FPer12346%2F001.jpg
```

An example of a URL that a Rendering Client might use to access this attachment is:

```
http://www.clientapp.net/getAttachment?partner=dhs.icepick&uri=http%3A%2F%2Fw
ww.usdoj.gov%2FDOJ%2FJABS%2FLEXS%2FPHOTO%2FPer12346%2F001.jpg
```

This approach has the benefit of being a standardized, reproducible mapping from the URI to a unique identifier that can be used as a parameter in a URL. The escaping mechanism will be used both by the Rendering Client in generating IDs to retrieve specific attachments, as well as in the rendering instructions (in XSLT). The XSLT will generate an HTML document that contains, in the case of images, an "img" element that points to the correct location, as in:

```
<img
src="http://www.clientapp.net/getAttachment?partner=dhs.icepick&uri=http%3A%2
F%2Fwww.usdoj.gov%2FDOJ%2FJABS%2FLEXS%2FPHOTO%2FPer12346%2F001.jpg"/>
```

Although the more common example is images, the escaping mechanism applies to all kinds of attachments. For example, XSLT stylesheets may include other XSLT stylesheets as separate attachments, or refer to external CSS stylesheets. These files will also be identified by attachment URIs that need to be escaped.

The table below lists the specific characters, identified as reserved characters in RFC 3986, that must be escaped, along with their escape value. The escape values are case-sensitive, and should always be in upper case.

Character	Character Description	Escape Value
	space	%20
:	colon	%3A
/	forward slash	%2F
?	question mark	%3F
#	hash mark/pound sign	%23

[3] RFC 3986 can be found at http://www.ietf.org/rfc/rfc3986.txt

Character	Character Description	Escape Value
[	left square bracket	%5B
]	right square bracket	%5D
@	at sign	%40
!	exclamation mark	%21
"	quote	%22
&	ampersand	%24
$	dollar sign	%26
'	apostrophe	%27
(	left parenthesis	%28
)	right parenthesis	%29
*	asterisk	%2A
+	plus sign	%2B
,	comma	%2C
;	semicolon	%3B
=	equals sign	%3D
<	less than	%3C
>	greater than	%3E
\	backward slash	%5C
|	vertical bar	%7C

Note: Other characters, such as non-ASCII accented letters, also have escape values. These special characters will already be escaped in the attachment URIs and do not need to be further escaped by this process.

7.2.2 Formulating the URL

The beginning of the URL (before the escaped attachment URI) is determined by the Rendering Client. For example, the text in blue in this example is specific to the Rendering Client:

```
http://www.clientapp.net/getAttachment?partner=dhs.icepick&uri=http%3A%2F%2Fw
ww.usdoj.gov%2FDOJ%2FJABS%2FLEXS%2FPHOTO%2FPer12346%2F001.jpg
```

Although attachments URIs must be unique within a partner system, their uniqueness is not guaranteed across all partner systems. Therefore, the full URLs of the attachments should include some designation of the partner system.

The example above shows an additional parameter being passed (`partner=dhs.icepick`) with the attachment location that specifies the organization name and system ID. Another example is:

```
http://www.clientapp.net/dhs.icepick/getAttachment?uri=http%3A%2F%2Fwww.usdoj
.gov%2FDOJ%2FJABS%2FLEXS%2FPHOTO%2FPer12346%2F001.jpg
```

The syntax for specifying the partner system in the beginning part of the URL can be decided by the Rendering Client.

Because the author of the rendering XSLT has no prior knowledge of the beginning of the URL, it must be passed to the XSLT as a parameter. Each rendering instructions stylesheet that needs to display attachments will have a parameter named "attachmentLocation". The stylesheet will concatenate the value of the attachmentLocation parameter to the beginning of each escaped attachment URI, resulting in correct URLs in the HTML.

If the attachmentLocation parameter is not passed to the XSLT, or it is a blank value, the rendering instructions will not display the images.

In the XSLT rendering instructions example below, the attachmentLocation parameter is defined. It is then used at the beginning of the value of the "src" attribute of the "img" element, which contains the URL of the image.

```
<xsl:stylesheet ...>
  <xsl:param name="attachmentLocation"/>
  <xsl:template match="/">
    <html>
      <!-- mug shot(s)-->
      <xsl:for-each select=".//lexsdigest:EntityPersonFacialImageAssociation">
        <xsl:variable name="attLinkID" select="lexsdigest:AttachmentLinkReference/@s:ref"/>
        <xsl:variable name="AttachmentURI"
                      select="//lexs:AttachmentLink[@s:id = $attLinkID]/lexs:AttachmentURI"/>
        <img>
          <xsl:attribute name="src">
            <xsl:value-of select="$attachmentLocation"/>
            <xsl:call-template name="escape-uri">
              <xsl:with-param name="uri" select="$AttachmentURI"/>
            </xsl:call-template>
          </xsl:attribute>
        </img>
      </xsl:for-each>
      <!--...-->
    </html>
  </xsl:template>
</xsl:stylesheet>
```

7.3 Caching of Attachments

Rendering Clients are encouraged to cache attachments in order to improve performance and reduce bandwidth, memory and disk space requirements.

Because the attachment URIs are only guaranteed to be unique within a partner system, the caching strategy should take into account that the attachment URI alone is not sufficient to uniquely identify and retrieve a cached attachment.

Images and pre-rendered representations of data items should be treated like data. They should not be cached for more than twenty-four hours. Additionally, they should be considered transient; they should not be permanently archived or backed up.

XSLT rendering instructions and CSS stylesheets, like other attachments, can be cached. Because rendering instructions change very infrequently, they can be cached less frequently. Retrieving the rendering instructions once per day is sufficient to ensure that up-to-date stylesheets are being used.

Appendix A Entity Examples

A very simple but technically complete example of a person entity is shown below.

```
</lexsdigest:EntityPerson>
    <lexsdigest:Person>
        <nc:PersonName>
            <nc:PersonGivenName>John</nc:PersonGivenName>
            <nc:PersonSurName>Jacobs</nc:PersonSurName>
        </nc:PersonName>
    </lexsdigest:Person>
</lexsdigest:EntityPerson>
```

Obviously, the example above (XML example for Person “John Jacobs”) is a very simple example and both LEXS and the underlying NIEM structures provide a very rich set of elements that can contain a considerable amount of information relating to a person. Thus, a more detailed person sample may be represented by the following XML fragment:

```
<lexsdigest:EntityPerson>
    <lexsdigest:Person >
        <nc:PersonAgeMeasure>
            <nc:MeasureRangeValue>
                <nc:RangeMinimumValue>25</nc:RangeMinimumValue>
                <nc:RangeMaximumValue>30</nc:RangeMaximumValue>
            </nc:MeasureRangeValue>
        </nc:PersonAgeMeasure>
        <nc:PersonAlternateName>
            <nc:PersonFullName>Taz</nc:PersonFullName>
        </nc:PersonAlternateName>
        <nc:PersonBirthDate>
          <nc:Date>1972-05-09</nc:Date>
        </nc:PersonBirthDate>
        <nc:PersonEyeColorCode>BRO</nc:PersonEyeColorCode>
        <nc:PersonHairColorCode>BRO</nc:PersonHairColorCode>
        <nc:PersonHeightMeasure>
            <nc:MeasureRangeValue>
                <nc:RangeMinimumValue>66</nc:RangeMinimumValue>
                <nc:RangeMaximumValue>72</nc:RangeMaximumValue>
            </nc:MeasureRangeValue>
            <nc:LengthUnitCode>INH</nc:LengthUnitCode>
        </nc:PersonHeightMeasure>
        <nc:PersonName>
            <nc:PersonGivenName>Billy</nc:PersonGivenName>
            <nc:PersonMiddleName>Bob</nc:PersonMiddleName>
            <nc:PersonSurName>Guy</nc:PersonSurName>
        </nc:PersonName>
        <nc:PersonRaceText>White</nc:PersonRaceText>
        <nc:PersonSexCode>M</nc:PersonSexCode>
        <nc:PersonSSNIdentification>
            <nc:IdentificationID>987654321</nc:IdentificationID>
        </nc:PersonSSNIdentification>
        <nc:PersonWeightMeasure>
            <nc:MeasureRangeValue>
```

```
                <nc:RangeMinimumValue>200</nc:RangeMinimumValue>
                <nc:RangeMaximumValue>250</nc:RangeMaximumValue>
            </nc:MeasureRangeValue>
            <nc:WeightUnitCode>LBR</nc:WeightUnitCode>
        </nc:PersonWeightMeasure>
    </lexsdigest:Person >
</lexsdigest:EntityPerson>
```

Below is an example for EntityLocation.

```
</lexsdigest:EntityLocation>
    <nc:Location>
        <nc:LocationAddress>
            <nc:StructuredAddress>
                <nc:LocationStreet>
                    <nc:StreetFullText>12 Main Street</nc:StreetFullText>
                </nc:LocationStreet>
                <nc:AddressSecondaryUnitText>Apt. 2234</nc:AddressSecondaryUnitText>
                <nc:LocationCityName>Smallville</nc:LocationCityName>
                <nc:LocationStateName>WV</nc:LocationStateName>
                <nc:LocationPostalCode>33445</nc:LocationPostalCode>
            </nc:StructuredAddress>
        </nc:LocationAddress>
        <nc:LocationTwoDimensionalGeographicCoordinate>
            <nc:GeographicCoordinateLatitude>
                <nc:LatitudeDegreeValue>42</nc:LatitudeDegreeValue>
                <nc:LatitudeMinuteValue>12</nc:LatitudeMinuteValue>
                <nc:LatitudeSecondValue>20</nc:LatitudeSecondValue>
            </nc:GeographicCoordinateLatitude>
            <nc:GeographicCoordinateLongitude>
                <nc:LongitudeDegreeValue>42</nc:LongitudeDegreeValue>
                <nc:LongitudeMinuteValue>12</nc:LongitudeMinuteValue>
                <nc:LongitudeSecondValue>20</nc:LongitudeSecondValue>
            </nc:GeographicCoordinateLongitude>
        </nc:LocationTwoDimensionalGeographicCoordinate>
    </nc:Location>
</lexsdigest:EntityLocation>
```

Appendix B LEXS 3.1 Associations

The following sections group the various associations in LEXS based on LEXS entities. The LEXS associations include associations from NIEM core and NIEM domains, as well as LEXS-specific associations. Not all NIEM associations are imported into LEXS, only those believed to be of greatest utility to LEXS and which associate objects that are included in LEXS.

Associations Involving Persons

j:AccompliceAssociation
j:ActivityAssistingPersonAssociation
j:ActivityClearerPersonAssociation
j:ActivityDispatcherAssociation
j:ActivityEnforcementOfficialAssociation
j:ActivityInformationAbstracterPersonAssociation
j:ActivityInformationApproverAssociation
j:ActivityInformationOwnerAssociation
j:ActivityInformationReleaserAssociation
j:ActivityInformationReporterAssociation
j:ActivityJudicialOfficialAssociation
j:ActivityUnknownAffiliateAssociation
j:IncidentInformantAssociation
j:IncidentInvestigatorAssociation
j:IncidentItemObtainerAssociation
j:IncidentUnknownAssociationPersonAssociation
j:SubjectInvolvedPersonAssociation
lexsdigest:ActivityPrimaryPersonAssociation *(3.1.4 and later)*
lexsdigest:ActivityResponsiblePersonAssociation *(3.1.4 and later)*
lexsdigest:ArrestOfficerAssociation
lexsdigest:ArrestSubjectAssociation
lexsdigest:BailBondAssociation *(3.1.1 and later)*
lexsdigest:BoyfriendGirlfriendAssociation
lexsdigest:CaregiverAssociation
lexsdigest:CellmateAssociation *(3.1.1 and later)*
lexsdigest:ClientAssociation
lexsdigest:DeclarationPersonAssociation *(3.1.1 and later)*
lexsdigest:DeliveryAssociation
lexsdigest:DocumentAuthorAssociation *(3.1.4 and later)*
lexsdigest:EmailMessageAssociation
lexsdigest:EntityEmailAssociation
lexsdigest:EntityInstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:EntityNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:EntityTelephoneNumberAssociation
lexsdigest:HomosexualAssociation
lexsdigest:IncidentReportingOfficialAssociation
lexsdigest:IncidentSubjectPersonAssociation
lexsdigest:IncidentVictimPersonAssociation
lexsdigest:IncidentWitnessAssociation
lexsdigest:InmateApprovedTelephoneListAssociation
lexsdigest:InmateTelephoneCallLogAssociation
lexsdigest:InstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:ItemLienHolderPersonAssociation
lexsdigest:NetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:OffenseSubjectPersonAssociation
lexsdigest:OffenseVictimPersonAssociation
lexsdigest:OffenseWitnessAssociation
lexsdigest:PatientAssociation
lexsdigest:PersonArrestLocationAssociation
lexsdigest:PersonBirthLocationAssociation
lexsdigest:PersonContactAssociation *(3.1.4 and later)*
lexsdigest:PersonFoundLocationAssociation *(3.1.1 and later)*
lexsdigest:PersonLastSeenWitnessAssociation *(3.1.1 and later)*
lexsdigest:PersonOfInterestAssociation
lexsdigest:PhysicalMailAssociation
lexsdigest:ServiceCallCallerAssociation
lexsdigest:ServiceCallDispatcherAssociation
lexsdigest:ServiceCallOperatorAssociation
lexsdigest:StudentAssociation
lexsdigest:SubjectVictimAssociation
lexsdigest:SubjectWitnessAssociation
lexsdigest:TelephoneCallAssociation
lexsdigest:VictimWitnessAssociation
lexsdigest:VisitorAssociation *(3.1.1 and later)*
nc:AcquaintanceAssociation
nc:ActivityInvolvedPersonAssociation
nc:ActivitySupervisorPersonAssociation
nc:AuthorityFigureAssociation
nc:BabysitterAssociation
nc:CohabitantAssociation
nc:CoworkerAssociation
nc:DomesticPartnershipAssociation
nc:FamilyAssociation
nc:FriendshipAssociation
nc:GuardianAssociation
nc:ImmediateFamilyAssociation
nc:LocationNeighboringPersonAssociation
nc:MarriageAssociation
nc:NeighborAssociation

nc:OrganizationPrincipalOfficialAssociation
nc:PersonActivityInvolvementAssociation
nc:PersonAssignedUnitAssociation
nc:PersonAssociation
nc:PersonConveyanceAssociation
nc:PersonCriminalOrganizationAssociation
nc:PersonCurrentEmploymentAssociation
nc:PersonCurrentLocationAssociation
nc:PersonDetainmentLocationAssociation
nc:PersonEmploymentAssociation
nc:PersonEmploymentLocationAssociation
nc:PersonFormerEmploymentAssociation
nc:PersonGangAssociation
nc:PersonInvolvedInDrivingIncidentAssociation
nc:PersonItemAssociation
nc:PersonKnownPreviousLocationAssociation
nc:PersonLastSeenLocationAssociation
nc:PersonLocationAssociation
nc:PersonOrganizationAffiliationAssociation
nc:PersonOrganizationAssociation
nc:PersonOwnsItemAssociation
nc:PersonPossessesItemAssociation
nc:PersonPrimaryWorkerAssociation
nc:PersonReferralWorkerAssociation
nc:PersonTemporaryAssignedUnitAssociation
nc:PersonWorkerAssociation
nc:ResidenceAssociation
nc:StrangerAssociation
nc:TransportationAssociation
scr:AdoptedChildAssociation

Associations Involving Organizations

lexsdigest:BailBondAssociation *(3.1.1 and later)*
lexsdigest:CaregiverAssociation
lexsdigest:ClientAssociation
lexsdigest:DeliveryAssociation
lexsdigest:EmailMessageAssociation
lexsdigest:EntityEmailAssociation
lexsdigest:EntityInstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:EntityNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:EntityTelephoneNumberAssociation
lexsdigest:IncidentSubjectOrganizationAssociation
lexsdigest:IncidentVictimOrganizationAssociation
lexsdigest:InmateTelephoneCallLogAssociation
lexsdigest:InstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:ItemLienHolderOrganizationAssociation
lexsdigest:NetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:OffenseSubjectOrganizationAssociation
lexsdigest:OffenseVictimOrganizationAssociation
lexsdigest:PatientAssociation
lexsdigest:PhysicalMailAssociation
lexsdigest:StudentAssociation
lexsdigest:SubjectVictimAssociation
lexsdigest:SubjectWitnessAssociation
lexsdigest:TelephoneCallAssociation
lexsdigest:VictimWitnessAssociation
lexsdigest:VisitorAssociation *(3.1.1 and later)*
nc:ActivityInformationAbstracterOrganizationAssociation
nc:ActivityInformationClearerOrganizationAssociation
nc:ActivityInvolvedOrganizationAssociation
nc:ActivityPrimaryOrganizationAssociation
nc:ActivityReportingOrganizationAssociation
nc:ActivityResponsibleOrganizationAssociation
nc:ActivitySupervisorOrganizationAssociation
nc:ItemHolderAssociation
nc:ItemMoverAssociation
nc:LocationContainsOrganizationAssociation
nc:LocationEmergencyServicesAssociation
nc:LocationOrganizationAssociation
nc:LocationPoliceDepartmentAssociation
nc:OrganizationAssociation
nc:OrganizationGangAssociation
nc:OrganizationItemAssociation
nc:OrganizationOwnsItemAssociation
nc:OrganizationParentAssociation
nc:OrganizationPossessesItemAssociation
nc:OrganizationPrincipalOfficialAssociation
nc:OrganizationSubsidiaryAssociation
nc:PersonAssignedUnitAssociation
nc:PersonCriminalOrganizationAssociation
nc:PersonCurrentEmploymentAssociation
nc:PersonEmploymentAssociation
nc:PersonFormerEmploymentAssociation
nc:PersonGangAssociation
nc:PersonOrganizationAffiliationAssociation
nc:PersonOrganizationAssociation
nc:PersonTemporaryAssignedUnitAssociation
nc:VehicleTowerAssociation

Associations Involving Locations

j:ActivityLocationAssociation
lexsdigest:CreditCardBillingLocationAssociation *(3.1.4 and later)*
lexsdigest:IncidentLocationAssociation
lexsdigest:OffenseLocationAssociation
lexsdigest:PersonArrestLocationAssociation
lexsdigest:PersonBirthLocationAssociation
lexsdigest:PersonContactAssociation *(3.1.4 and later)*
lexsdigest:PersonFoundLocationAssociation
lexsdigest:PhysicalMailAssociation *(3.1.1 and later)*
lexsdigest:PropertyRecoveryLocationAssociation
lexsdigest:PropertySeizedLocationAssociation
lexsdigest:ServiceCallDispatchLocationAssociation
lexsdigest:ServiceCallLocationAssociation
nc:ItemLocationAssociation
nc:LocationContainsOrganizationAssociation
nc:LocationEmergencyServicesAssociation
nc:LocationNeighboringPersonAssociation
nc:LocationOrganizationAssociation
nc:LocationPoliceDepartmentAssociation
nc:PersonCurrentLocationAssociation
nc:PersonDetainmentLocationAssociation
nc:PersonEmploymentLocationAssociation
nc:PersonKnownPreviousLocationAssociation
nc:PersonLastSeenLocationAssociation
nc:PersonLocationAssociation
nc:PropertyCurrentLocationAssociation
nc:PropertyDispositionLocationAssociation
nc:ResidenceAssociation
nc:VehicleGarageLocationAssociation
scr:LocationAssociation

Associations Involving Items, including CreditCard *(3.1.4 and later)*, Drug, Explosive, Firearm, Substance, TangibleItem and IntangibleItem *(3.1.1 and later)*

lexsdigest:ArrestInvolvedWeaponAssociation
lexsdigest:CreditCardBillingLocationAssociation *(3.1.4 and later)*
lexsdigest:IncidentEvidenceAssociation *(3.1.1 and later)*
lexsdigest:IncidentInvolvedItemAssociation
lexsdigest:IncidentWeaponAssociation
lexsdigest:ItemAssignedNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:ItemAssignedTelephoneNumberAssociation *(3.1.1 and later)*
lexsdigest:ItemEmailAddressAssociation *(3.1.1 and later)*
lexsdigest:ItemInstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:ItemLienHolderOrganizationAssociation
lexsdigest:ItemLienHolderPersonAssociation
lexsdigest:ItemNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:ItemTelephoneNumberAssociation *(3.1.1 and later)*
lexsdigest:OffenseInvolvedItemAssociation
lexsdigest:OffenseWeaponAssociation
lexsdigest:PropertyRecoveryLocationAssociation
lexsdigest:PropertySeizedLocationAssociation
nc:ActivityItemAssociation *(3.1.1 and later)*
nc:ItemContainerAssociation *(3.1.1 and later)*
nc:ItemHolderAssociation
nc:ItemLocationAssociation
nc:ItemMoverAssociation
nc:OrganizationItemAssociation
nc:OrganizationOwnsItemAssociation
nc:OrganizationPossessesItemAssociation
nc:PersonItemAssociation
nc:PersonOwnsItemAssociation
nc:PersonPossessesItemAssociation
nc:PropertyCurrentLocationAssociation
nc:PropertyDispositionLocationAssociation

Associations Involving Conveyances, including Aircraft, Vehicle, and Vessel

lexsdigest:ArrestInvolvedWeaponAssociation
lexsdigest:IncidentEvidenceAssociation *(3.1.1 and later)*
lexsdigest:IncidentInvolvedItemAssociation
lexsdigest:IncidentWeaponAssociation
lexsdigest:ItemLienHolderOrganizationAssociation
lexsdigest:ItemLienHolderPersonAssociation
lexsdigest:OffenseInvolvedItemAssociation
lexsdigest:OffenseWeaponAssociation
lexsdigest:PropertyRecoveryLocationAssociation
lexsdigest:PropertySeizedLocationAssociation
nc:ActivityItemAssociation *(3.1.1 and later)*
nc:ItemHolderAssociation
nc:ItemLocationAssociation
nc:ItemMoverAssociation
nc:OrganizationItemAssociation
nc:OrganizationOwnsItemAssociation
nc:OrganizationPossessesItemAssociation
nc:PersonConveyanceAssociation
nc:PersonItemAssociation
nc:PersonOwnsItemAssociation
nc:PersonPossessesItemAssociation
nc:PropertyCurrentLocationAssociation
nc:PropertyDispositionLocationAssociation
nc:TransportationAssociation
nc:VehicleGarageLocationAssociation (Vehicle only)
nc:VehicleTowerAssociation (Vehicle only)

Associations Involving Activity

j:ActivityAssistingPersonAssociation
j:ActivityClearerPersonAssociation
j:ActivityDispatcherAssociation
j:ActivityEnforcementOfficialAssociation
j:ActivityInformationAbstracterPersonAssociation
j:ActivityInformationApproverAssociation
j:ActivityInformationOwnerAssociation
j:ActivityInformationReleaserAssociation
j:ActivityInformationReporterAssociation
j:ActivityJudicialOfficialAssociation
j:ActivityLocationAssociation
j:ActivityUnknownAffiliateAssociation
j:IncidentInformantAssociation
j:IncidentInvestigatorAssociation
j:IncidentItemObtainerAssociation
j:IncidentUnknownAssociationPersonAssociation
lexsdigest:ActivityEmailAddressAssociation *(3.1.4 and later)*
lexsdigest:ActivityHashAssociation *(3.1.4 and later)*
lexsdigest:ActivityInstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:ActivityNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:ActivityPrimaryPersonAssociation *(3.1.4 and later)*
lexsdigest:ActivityResponsiblePersonAssociation *(3.1.4 and later)*
lexsdigest:ActivityTelephoneNumberAssociation *(3.1.4 and later)*
lexsdigest:ArrestInvolvedWeaponAssociation
lexsdigest:ArrestOffenseAssociation
lexsdigest:ArrestOfficerAssociation
lexsdigest:ArrestSubjectAssociation
lexsdigest:IncidentArrestAssociation
lexsdigest:IncidentEmailAddressAssociation *(3.1.1 and later)*
lexsdigest:IncidentEvidenceAssociation *(3.1.1 and later)*
lexsdigest:IncidentInvolvedItemAssociation
lexsdigest:IncidentLocationAssociation
lexsdigest:IncidentInstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:IncidentNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:IncidentOffenseAssociation
lexsdigest:IncidentReportingOfficialAssociation
lexsdigest:IncidentServiceCallAssociation
lexsdigest:IncidentSubjectOrganizationAssociation
lexsdigest:IncidentSubjectPersonAssociation
lexsdigest:IncidentTelephoneNumberAssociation *(3.1.1 and later)*
lexsdigest:IncidentVictimOrganizationAssociation
lexsdigest:IncidentVictimPersonAssociation
lexsdigest:IncidentWeaponAssociation
lexsdigest:IncidentWitnessAssociation
lexsdigest:OffenseEmailAddressAssociation *(3.1.1 and later)*
lexsdigest:OffenseInstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:OffenseInvolvedItemAssociation
lexsdigest:OffenseLocationAssociation
lexsdigest:OffenseNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:OffenseSubjectOrganizationAssociation
lexsdigest:OffenseSubjectPersonAssociation
lexsdigest:OffenseTelephoneNumberAssociation *(3.1.1 and later)*
lexsdigest:OffenseVictimOrganizationAssociation
lexsdigest:OffenseVictimPersonAssociation
lexsdigest:OffenseWeaponAssociation
lexsdigest:OffenseWitnessAssociation
lexsdigest:PersonOfInterestAssociation
lexsdigest:ServiceCallCallerAssociation
lexsdigest:ServiceCallDispatcherAssociation
lexsdigest:ServiceCallDispatchLocationAssociation
lexsdigest:ServiceCallEmailAddressAssociation
lexsdigest:ServiceCallInstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:ServiceCallLocationAssociation
lexsdigest:ServiceCallNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:ServiceCallOperatorAssociation
lexsdigest:ServiceCallTelephoneNumberAssociation
nc:ActivityInformationAbstracterOrganizationAssociation
nc:ActivityInformationClearerOrganizationAssociation
nc:ActivityInvolvedOrganizationAssociation
nc:ActivityInvolvedPersonAssociation
nc:ActivityItemAssociation *(3.1.1 and later)*
nc:ActivityPrimaryOrganizationAssociation
nc:ActivityReportingOrganizationAssociation
nc:ActivityResponsibleOrganizationAssociation
nc:ActivitySupervisorOrganizationAssociation
nc:ActivitySupervisorPersonAssociation
nc:PersonActivityInvolvementAssociation
nc:PersonInvolvedInDrivingIncidentAssociation
nc:PersonPrimaryWorkerAssociation
nc:PersonReferralWorkerAssociation
nc:PersonWorkerAssociation
nc:PreviousActivityAssociation
nc:RelatedActivityAssociation
nc:RelatedCaseAssociation

Associations Involving Email

lexsdigest:ActivityEmailAddressAssociation *(3.1.4 and later)*
lexsdigest:EmailMessageAssociation
lexsdigest:EntityEmailAssociation
lexsdigest:IncidentEmailAddressAssociation *(3.1.1 and later)*
lexsdigest:ItemEmailAddressAssociation *(3.1.1 and later)*
lexsdigest:OffenseEmailAddressAssociation *(3.1.1 and later)*
lexsdigest:PersonContactAssociation *(3.1.4 and later)*
lexsdigest:ServiceCallEmailAddressAssociation

Associations Involving InstantMessenger *(3.1.4 and later)*

lexsdigest:ActivityInstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:EntityInstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:IncidentInstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:InstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:ItemInstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:OffenseInstantMessengerAssociation *(3.1.4 and later)*
lexsdigest:ServiceCallInstantMessengerAssociation *(3.1.4 and later)*

Associations Involving NetworkAddress *(3.1.4 and later)*

lexsdigest:ActivityNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:EntityNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:IncidentNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:ItemAssignedNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:ItemNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:NetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:OffenseNetworkAddressAssociation *(3.1.4 and later)*
lexsdigest:ServiceCallNetworkAddressAssociation *(3.1.4 and later)*

Associations Involving TelephoneNumber

lexsdigest:ActivityTelephoneNumberAssociation *(3.1.4 and later)*
lexsdigest:EntityTelephoneNumberAssociation
lexsdigest:IncidentTelephoneNumberAssociation *(3.1.1 and later)*
lexsdigest:InmateApprovedTelephoneListAssociation
lexsdigest:InmateTelephoneCallLogAssociation
lexsdigest:ItemAssignedTelephoneNumberAssociation
lexsdigest:ItemTelephoneNumberAssociation
lexsdigest:OffenseTelephoneNumberAssociation *(3.1.1 and later)*
lexsdigest:PersonContactAssociation *(3.1.4 and later)*
lexsdigest:ServiceCallTelephoneNumberAssociation
lexsdigest:TelephoneCallAssociation

Associations Involving Hash *(3.1.4 and later)*

lexsdigest:ActivityHashAssociation *(3.1.4 and later)*

Associations Involving Document *(3.1.4 and later)*

lexsdigest:DocumentAuthorAssociation *(3.1.4 and later)*

Associations Involving Package Structure

The following associations are related to linking elements to other elements or Attachments which maybe inside or outside a package.

lexsdigest:EntityAttachmentLinkAssociation
lexsdigest:EntityCrimeSceneImageAssociation
lexsdigest:EntityItemImageAssociation
lexsdigest:EntityPersonFacialImageAssociation
lexsdigest:EntityPersonImageAssociation
lexsdigest:EntityPersonSMTImageAssociation
lexsdigest:EntitySupportingDocumentationAssociation
lexslib:SameAsPayloadAssociation *(3.1.1 and later)*

Appendix C LEXS 3.1 Roles

The following sections group the various roles in LEXS based on LEXS entities. The LEXS roles include roles from NIEM core and NIEM domains, as well as LEXS-specific roles. Not all NIEM roles are imported into LEXS, only those believed to be of greatest utility to LEXS.

Roles of Person

em:Resource
j:AppellateCaseDecisionIssuingJudge
j:AppellateCaseNoticeProsecutingAttorney
j:ArrestOfficial
j:ArrestSubject
j:Attorney
j:BailSubject
j:BookingEmployee
j:BookingReleaseCorrectionsAnalyst
j:BookingSearchOfficial
j:BookingSubject
j:BookingTelephoneCallSupervisingOfficial
j:BookingTransportOfficial
j:CaseJuror
j:CaseWitness
j:ChargeSubject
j:ChargeVictim
j:CitationIssuingOfficial
j:CitationSubject
j:ConvictionSubject
j:CourtEventJudge
j:CourtOrderDesignatedSubject
j:CourtOrderIssuingJudicialOfficial
j:CourtOrderServiceOfficialEnforcement
j:CourtOrderServiceOfficialJudicial
j:CustodyTransferReceivingEnforcementOfficial
j:CustodyTransferReleasingEnforcementOfficial
j:CustodyTransferSubject
j:EnforcementOfficial
j:ForceSubject
j:ForceVictim
j:IncidentAssistingOfficial
j:IncidentReportingOfficial
j:IncidentResponseOfficial
j:IncidentSubject
j:IncidentSupervisingOfficial
j:IncidentVictim
j:IncidentWitness
j:Judge
j:JudicialOfficial
j:Juror
j:MissingPerson
j:MissingPersonLastSeenWitness
j:PropertySeizureSeizingEnforcementOfficial
j:RegisteredOffender
j:RegisteredSexOffender
j:SentenceSubject
j:ServiceCallDispatchedOfficial
j:SeverityLevelAssignedJudge
j:Subject
j:Suspect
j:VerdictIssuingJudge
j:VerdictSubject
j:Victim
j:VisitationSupervisingOfficialEnforcement
j:VisitationSupervisingOfficialJudicial
j:Witness
lexdigest:Informant
lexdigest:Inmate
lexdigest:OtherInvolvedPerson
lexdigest:Parolee
lexdigest:Prisoner
lexdigest:Probationer
lexdigest:ProtectedParty
lexdigest:SupervisionSubject
nc:Lessee *(3.1.1 and later)*
nc:Lessor *(3.1.1 and later)*

Roles of Organization

j:ChargeSubject
j:ChargeVictim
j:CitationSubject
j:ConvictionSubject
j:CourtOrderDesignatedSubject
j:CriminalOrganization
j:ForceSubject
j:ForceVictim
j:IncidentSubject
j:IncidentVictim
j:SentenceSubject
j:Subject
j:Suspect
j:VerdictSubject
j:Victim
lexdigest:SupervisionSubject
nc:Lessee *(3.1.1 and later)*
nc:Lessor *(3.1.1 and later)*
nc:VehicleBrander

Roles of Item

em:Resource
j:ChargeVictim
j:IncidentVictim
j:ForceVictim
j:Victim
nc:Weapon

Appendix D Search Result Paging Support

There may be cases where a search query results in more hits than can be returned, either because the user requested fewer or because the service provider limits the number of hits that can be returned. Users, or applications acting on behalf of a user, must be able to specify a maximum number of hits that should be returned.

Limiting the number of hits
doTextSearch and doStructuredSearch requests include a mandatory element called MaxItemMatchesRequested. This element allows the query to specify the maximum number of hits that should be returned in the response. Note that the service provider may return less than this either because of fewer hits or because the service provider has a lower limit for the number of hits that can be returned.

If a service provider has more hits than can be returned in a single response, the service provider limits responses based on the type of search performed. Responses to structured searches are limited based on dates, with the most recent being returned for the initial query. Responses to unstructured searches are limited based on whatever relevancy score is used by the service provider, with the most relevant being returned for the initial query.

For all responses, the service provider populates three elements.

- ServerLimitFlag, a mandatory Boolean element to indicate whether the number of hits was limited due to a service provider limit. True means the number returned was limited due to a server restriction.

- MaxItemMatchesRequested, a mandatory numeric element, which contains the value supplied in MaxItemMatches in the doStructuredSearch or doTextSearch. This makes the quantity originally requested readily available to any client application consuming the response without having to refer to the original query.

- NumberItemMatches, a mandatory text element to indicate how many matches there really were. This is a text field so that the service provider does not necessarily have to determine an exact number, which may require retrieving a large number of hits. For example, if 20 hits were requested and there are really 50 matches, the service provider can count the actual number of matches and populate NumberItemMatches with the value 50, or the service provider can skip counting all the matches and return the string 20+ to indicate that there were more than 20.

Service providers may support the capability to return additional hits when all hits cannot be returned in the initial response. Since some service providers may not want to retain any state information about searches and responses that have already been processed, LEXS SR provides a mechanism to support the request for additional hits without requiring service providers to maintain state information. However, since other service provider may retain state information, LEXS also supports an identifier that can be used to facilitate requests for additional hits.

If supported by the service provider, when there are more hits than can be returned in the response, the service provider populates an optional text element called MatchEndPoint. This can be utilized in a follow-up query to get the next set of hits. If the response is from a “next” query, the service provider populates an optional text element called MatchBeginPoint. These point elements are used by a service provider on a “next” or “previous” query to determine what set of data to return, and could be timestamps, or relevancy scores, or record numbers; but are only intended to be meaningful to the service provider.

Service providers that retain information about what hits were returned in response to queries can populate an optional text element called ServiceProviderSearchID with a value that is meaningful to the service provider for returning additional hits. This allows the service provider to utilize any state information it may retain so that it does not have to rerun the query again to get the next or previous set of results. Service providers define this value; requesters merely copy the value back into the follow-up query.

Requesting additional hits (next/previous)

If supported by the service provider and the user’s client application, additional hits may be requested if there are more hits than can be returned in the initial response. So for example, if the maximum number of hits that can be returned is 20, and there are 50 hits in total, the next 20 can be requested (hits 21-40). Once hits 21-40 are received, the previous 20 can also be requested (hits 1-20) or the next 20 (hits 41-50 in this example).

In order to get the next set of hits, the client application submits the same search query that was used to generate this set of results, but also populates an optional text element called MatchBeginAfter to indicate to the service provider where the next set of hits should start. MatchBeginAfter is populated with the value of MatchEndPoint in the response the user is following up.

In order to get the previous set of hits, the client application submits the same search query, but also populates an optional text element called MatchEndBefore to indicate to the service provider where the previous set of hits should end. MatchEndBefore is populated with the value of MatchBeginPoint in the response the user is following up.

In either case, if the response that is being followed-up included the element ServiceProviderSearchID, then the follow-up query must populate the ServiceProviderSearchID with the value provided in that response. Service providers define this value; requesters merely copy the value back into the follow-up query. Note that a service provider may provide different values for this element for each set of hits; for example, the service provider may supply one value that indicates to it that the set returned was hits 1-20, and a different value to indicate that another set includes hits 21-40.

Examples
Original query requests no more than 20 hits, and there are 50 matches. Namespace aliases are left out to simplify examples. MatchEndPoint, MatchBeginPoint, MatchBeginAfter, and MatchEndBefore are shown here as timestamps, but could be anything meaningful to the service provider. Sequence shown may change in the actual schemas.

Example 1
Service provider does not retain any state information about previous searches and responses, and supports next and previous queries.

Initial search result
```
<ServerLimitFlag>false</ServerLimitFlag>
<MatchItemMatchesRequested>20</MatchItemMatchesRequested>
<NumberItemMatches>50</NumberItemMatches>   <!-- could be populated with 20+ -->
<MatchEndPoint> 2006-12-01T09:30:47.0Z</MatchEndPoint>
```

Follow-up search for the next 20 hits
(includes same search criteria as initial search)
```
<MaxItemMatches>20</MaxItemMatches>
<MatchBeginAfter>2006-12-01T09:30:47.0Z</MatchBeginAfter>
```

Second set of results
```
<ServerLimitFlag>false</ServerLimitFlag>
<MatchItemMatchesRequested>20</MatchItemMatchesRequested>
<NumberItemMatches>50</NumberItemMatches>   <!-- could be populated with 20+ -->
<MatchBeginPoint>2006-11-30T09:30:47.0Z</MatchBeginPoint>
<MatchEndPoint>2005-12-01T09:30:47.0Z</MatchEndPoint>
```

Follow-up search for the previous 20 hits, which is actually the original 20
(includes same search criteria as initial search)
```
<MaxItemMatches>20</MaxItemMatches>
<MatchEndBefore>2006-11-30T09:30:47.0Z</MatchEndBefore>
```

Example 2
Service provider does retain state information about previous searches and responses, and does support next and previous queries.

Initial search result
```
<ServerLimitFlag>false</ServerLimitFlag>
<MatchItemMatchesRequested>20</MatchItemMatchesRequested>
<NumberItemMatches>50</NumberItemMatches>   <!-- could be populated with 20+ -->
<MatchEndPoint> 2006-12-01T09:30:47.0Z</MatchEndPoint>
<ServiceProviderSearchID>ABC123</ServiceProviderSearchID>
```

Follow-up search for the next 20 hits
(includes same search criteria as initial search)

```
<MaxItemMatches>20</MaxItemMatches>
<MatchBeginAfter>2006-12-01T09:30:47.0Z</MatchBeginAfter>
<ServiceProviderSearchID>ABC123</ServiceProviderSearchID>
```

Second set of results

```
<ServerLimitFlag>false</ServerLimitFlag>
<MatchItemMatchesRequested>20</MatchItemMatchesRequested>
<NumberItemMatches>50</NumberItemMatches>   <!-- could be populated with 20+ -->
<MatchBeginPoint>2006-11-30T09:30:47.0Z</MatchBeginPoint>
<MatchEndPoint>2005-12-01T09:30:47.0Z</MatchEndPoint>
<ServiceProviderSearchID>ABC124</ServiceProviderSearchID>
<!-- doesn’t have to be different ID, up to service provider to determine how to track -->
```

Follow-up search for the previous 20 hits, which is actually the original 20
(includes same search criteria as initial search)

```
<MaxItemMatches>20</MaxItemMatches>
<MatchEndBefore>2006-11-30T09:30:47.0Z</MatchEndBefore>
<ServiceProviderSearchID>ABC124</ServiceProviderSearchID>
```

Example 3
Service provider does not support next and previous queries.

Initial search result

```
<ServerLimitFlag>false</ServerLimitFlag>
<MatchItemMatchesRequested>20</MatchItemMatchesRequested>
<NumberItemMatches>50</NumberItemMatches>   <!-- could be populated with 20+ -->
```

Appendix E Error/Warning Categories

The following warning and error categories are provided in an enumerated list for the AdvisoryCategory element. The list below breaks them out separately for warning and error for organizational purposes; the enumerated list in schema is not separated into warning versus error lists or sections. Note that some warnings might become errors. For example, if an unsupported element is provided in a search query and that is the only element provided, the query cannot be processed and the result should be an error; versus an unsupported element that can be ignored with the result based on remaining elements.

For Errors:

- Network failure (e.g.. cannot contact service provider)
- Invalid Response (response from service provider cannot be processed)
- Invalid Request (service provider could not process incoming request)
- Timeout (service provider did not respond in allotted time)
- Business Rules Not Met (e.g. service provider requires SSN in query, but query didn’t provide)
- Next/Previous Not Supported
- Structured Search Not Supported (structured search submitted, but service provider or data owner only support unstructured searches)
- Unstructured Search Not Supported (unstructured search submitted, but service provider or data owner only support structured searches)
- Fuzzy Match Not Supported
- Wildcards Not Supported (service provider does not support wildcard queries in unstructured searches)
- Logical Operators Not Supported (service provider does not support logical AND/OR queries in unstructured searches)
- Phrases Not Supported (service provider does not support exact phrase queries in unstructured searches)
- Invalid Attachment Requested (service provider received request for invalid Attachment)
- Invalid Data Item Requested (service provider received getDataItem request for data item that does not exist)
- Other Error (for errors that don’t fall into categories above)

For Warnings:

- Query Object Not Supported (e.g. query was on vehicle and service provider doesn’t handle vehicles)

- Query Field Not Supported (e.g. query was on SSN and service provider doesn't include SSN)
- Query Operator Not Supported (e.g. query included a wildcard and service provider only does exact)
- Quantity Mismatch (user asked for maximum # of hits, service provider supplied different #)
- Pointer Information Only (service provider can only return contact info for follow-up, but no data)
- Multiple Values Not Supported (search query included multiple values for a search field, but service provider only supports single values)
- Other Warning (other issues that don't fall into above categories)

Appendix F Data Owner/Submitter and Message Origin/Destination

LEXS 3.0, which was limited to PD, only provided a mechanism to indicate where a message came from, which was generally assumed to be the data owner. However, there are cases where an owner may publish data to another system, and the second system may then publish the data to a third system; so owner versus submitter can become murky. Therefore, LEXS 3.1 PD explicitly labels the data so that there is no confusion about who the owner is versus a submitter that may not own the data. LEXS SR search responses also explicitly indicate the owner, versus the responding system, versus the system that submitted the data to the responding system. The different terms and their definitions and usage are discussed below.

Organization and Placement
Data Owner is for an individual data item for both PD and SR.

Data Submitter is for an entire message for PD since all submissions in a single message are from the same submitter. Data Submitter is for an individual data item in SR. For SR, service providers may not retain information about the submitting system, so in this case Data Submitter is optional.

Message Origin and Data Submitter for PD are synonymous, so Message Origin is not included in PD. For SR requests, the Message Origin indicates the system making the request. However, for SR responses, the Message Origin indicates the responding system, which may be returning data from multiple submitters and/or multiple data owners.

Message Destination applies to SR and specifies the service provider the message is intended for.

Scenarios and Usage
This section provides examples of how the Owner, Submitter, Origin, and Destination are used and populated.

For PD, assume that SysA owns a record, and submits to SysB which then submits to SysC, which in turn submits to SysE. In this case, SysA must be specified as the Owner in all cases. The Submitter is the “last” submitter in the chain; so for the example above, when SysA submits to SysB, SysA is the Owner and Submitter, but when SysB submits to SysC, SysA is still the Owner, but SysB is the Submitter. This is illustrated in top portion of Figure 34 below.

For SR, assume that SysF performs a search that matches the data submitted by SysA above, and that SysD directly submitted another record to SysE that also matches the search criteria. As is shown in Figure 34 below, the query will have SysF as the Origin and SysE as the Destination; while the response will have two hits, with SysA and SysD, respectively, shown as Owners. Since the SysA record was submitted to SysE by SysC, the response shows SysC as the Submitter. SysE is shown as the response’s Origin since SysE is the one actually providing the hits to SysF, and SysF is listed as the Destination in the response.

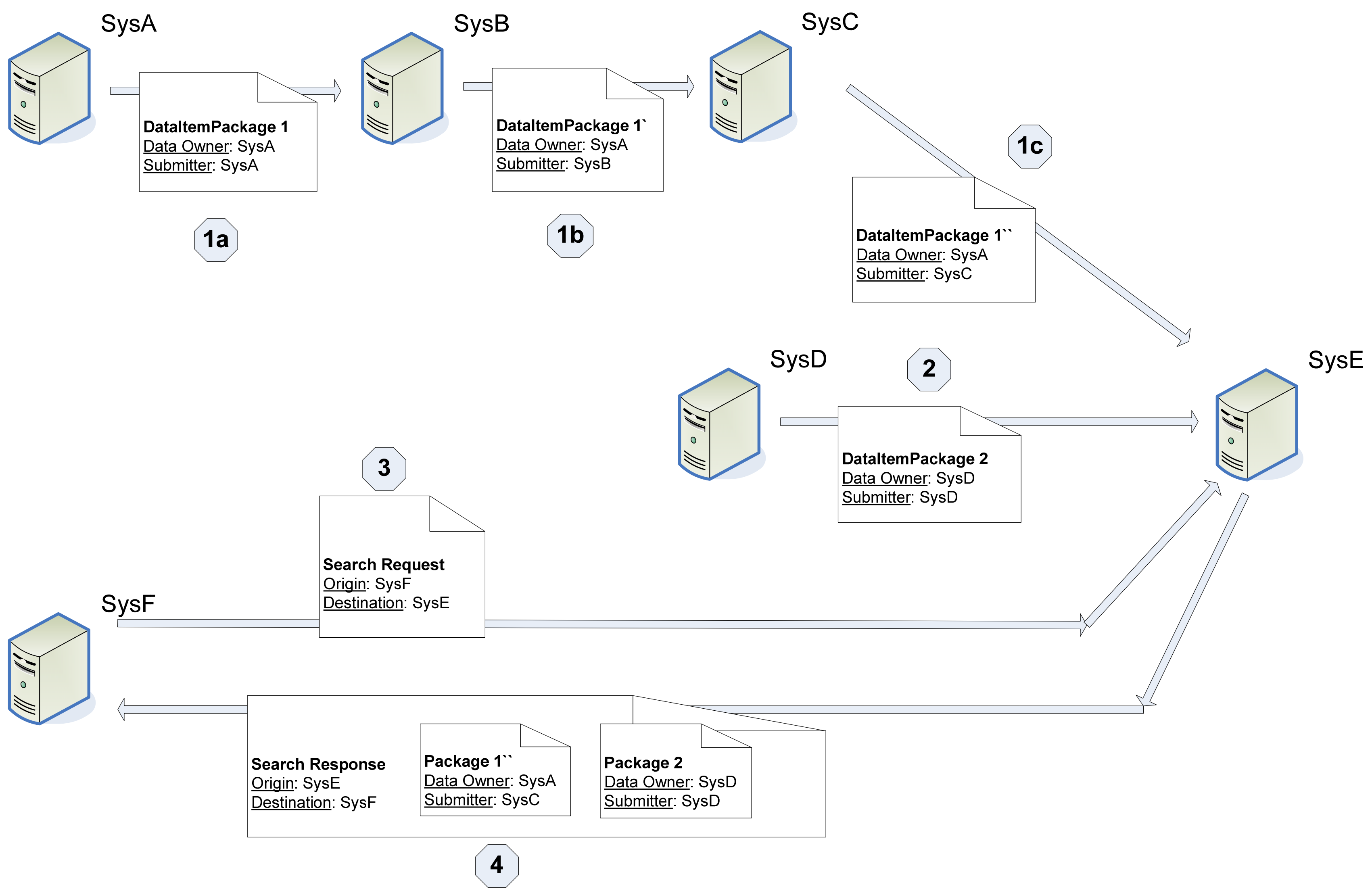


Figure 34 - Example of Data Owner, Data Submitter, Message Origin, and Message Destination

Step 1) SysA data submitted to SysE through a chain of submissions

- Step 1a) SysA submits to SysB
  - SysA as owner (in DataItemPackage/PackageMetadata/DataOwnerMetadata)
  - SysA as submitter (in DataSubmitterMetadata)
- Step 1b) SysB submits that data to SysC
  - SysA as owner (in DataItemPackage/PackageMetadata/DataOwnerMetadata)
  - SysB as submitter (in DataSubmitterMetadata)
- Step 1c) SysC submits that data to SysE
  - SysA as owner (in DataItemPackage/PackageMetadata/DataOwnerMetadata)
  - SysC as submitter (in DataSubmitterMetadata)

Step 2) SysD data submitted directly to SysE

- SysD as owner (in DataItemPackage/PackageMetadata/DataOwnerMetadata)
- SysD as submitter (in DataSubmitterMetadata)

Step 3) SysF queries SysE
Request message has SysF as message origin, SysE as message destination

Step 4) SysE returns hits to SysF, one from SysA and one from SysD
Response message has SysE as message origin, SysF as message destination
Search response includes 2 packages

- Package 1
  - SysA as owner (in SearchResultPackage/PackageMetadata/DataOwnerMetadata)
  - SysC as submitter (in SearchResultPackage/DataSubmitterMetadata)
- Package 2
  - SysD as owner (in SearchResultPackage/PackageMetadata/DataOwnerMetadata