Logical Entity eXchange Specifications 5.0 (LEXS)

Building Structured Payloads Guide

Revision 1

Change History

Rev	Date	Author	Description of Revision
1	11/16/2016	GTRI	Initial version

Acknowledgments

This document was developed through a collaborative effort sponsored by the U.S. Department of Justice (DOJ) and written by the Georgia Tech Research Institute (GTRI).

The following individuals were responsible for the development of the LEXS 5.0 specifications and this documentation:

Jack Wallace, Editor	- GTRI
Eric Soto	- GTRI
Benjamin Shrom	- GTRI
Debby Park, Editor	- GTRI
Michael Haas	- DOJ

Table of Contents

1 Purpose

This document is intended for the reader who needs to build a Logical Entity eXchange Specifications (LEXS) 5.0 Structured Payload schema.

This document does not provide extensive background or detailed technical information on the LEXS specifications, which is provided in the LEXS 5.0 Specification Guide included as part of the LEXS 5.0 distribution. This document assumes the reader is familiar with basic LEXS concepts, or has at least reviewed the LEXS 5.0 Specification Guide.

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.

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. The focus is to document how to develop a complete set of information exchange schemas that define how information sharing will occur once a decision has been made that certain information can be shared.

2 Introduction

The Logical Entity eXchange Specifications (LEXS, pronounced "lex") were created to support the information sharing objectives of the Department of Justice (DOJ). LEXS aims to minimize the impact of the changing requirements and the varied demands of information sharing on the sources and consumers of data. Although 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 addresses two aspects of information sharing:

- Defines and consistently describes units of information to be shared.
- Defines interfaces and protocols for information distribution, including:
    - How data producers provide information and allow it to be shared.
    - How data consumers search and retrieve the information.
    - How data consumers subscribe to information and receive notification that the data of interest has been changed, searched, or accessed by someone else.

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 specifications shield both data sources and data recipients from the complexity of multiple interfaces and allow for the multipurpose use of information. A data item created by a source can be consumed by multiple recipients who can understand as much or as little of the data as necessary.

The LEXS 5.0 specification can be obtained from the LEXS community web site. Any updates to this document will also be posted at the same site.

3 LEXS Background Information

3.1 Terminology

The National Information Exchange Model (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.

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 logistics system, a logical unit of information may be details about a single piece of hardware, along with its maintenance history and location. 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.

LEXS defines a set of high-level, commonly understood, structured **base objects** that 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

3.2 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 report on a person, an incident, a shipping 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 are not required to develop large applications specific to each exchange, but rather build one that handles the basics and then expand upon it in order to produce or consume more complex exchanges.

The objective of the Digest is to present the most common characteristics of 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 Text portion of the package.

The LEXS Digest defines a high-level set of LEXS entities, each containing a basic set of NIEM elements, and in some cases extensions. LEXS 5.0 defines the following high-level- entities:

Activity
Aircraft
Arrest
Booking
Case
Citation
Court Activity
Credit Card
Document
Drug
Email
Explosive
Firearm
Hash
Incident
Instant Messenger
Intangible Item
Location
Network Address
Notification
Offense
Organization
Pawn Transaction
Person
Program
Prosecution
Sentence
Service Call
Substance
Supervision
Tangible Item
Telephone Number
Vehicle
Vessel
Warrant
Watchlist

3.3 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, but there are two approaches to constructing a Structured Payload.

The first approach is referred to as a **split data model.** With this approach, the Digest is a set of data that is the foundation of the Structured Payload. The independent structured payload schemas are built to be aware of the LEXS Digest, that is, Structured Payload entities build on and link to LEXS Digest entities. An exchange based on the split data model may also define a new structured payload entity where the LEXS Digest does not provide a foundation. For exchanges that use the split data model, LEXS includes a mechanism to connect elements in the Structured Payload to elements in the Digest or in another Structured Payload.

In the other approach, referred to as a **self-contained** Structured Payload, the Structured Payload contains all data needed by the exchange and the Digest contains a summary of the contents of the Structured Payload. With this approach, if a consumer understands the Structured Payload, they would ignore the Digest; if they do not understand the Structured Payload, they would consume the Digest. XML instance developers must populate all appropriate Digest data elements that are equivalent to a Structured Payload counterpart.

Additional information about these approaches to Structured Payloads can be found in Appendix A of the LEXS 5.0 Specification Guide that is included as part of the LEXS distribution.

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 a split data model Structured Payload may not make logical sense by itself without the contents of the Digest, all Structured Payload instances 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.

4 Structured Payload Concepts

This chapter is intended to provide details on Structured Payloads in general and is not specific to split or self-contained Structured Payloads except where specifically noted.

Note that LEXS is based on the structures, standards, and usage guidelines of NIEM. While Structured Payloads are not required to be based on NIEM, it is expected that the general use case will be NIEM-based Structured Payloads and this document is geared toward that use case. This does not mean that this document can only be used by those utilizing NIEM; in fact, much of the information presented here has nothing to do with NIEM. However, some terminology is based on NIEM concepts and some sections specifically address NIEM constructs. Portions of this document refer to NIEM Model Package Documentation (MPD) and NIEM conformance as documented in the NIEM Naming and Design Rules. Detailed discussion of these concepts is beyond the scope of this guide, but documentation is available on the NIEM web site.

4.1 Structured Payload Schemas versus LEXS Schemas

As discussed previously, each Structured Payload is based on a schema or schemas created by a community outside of LEXS. Regardless of whether a Structured Payload utilizes the split data model or the self-contained model, the Structured Payload schemas must co-exist peacefully with the LEXS schemas.

All information exchanges based on LEXS that include one or more Structured Payloads must include a copy of the appropriate LEXS schemas. LEXS 5.0 includes four top-level exchange schemas plus a number of supporting schemas. The four top-level exchange schemas are for publication and discovery (lexs-publish-discover.xsd), search and retrieval (lexs-search-retrieve.xsd), subscription and notification (lexs-subscribe-notify.xsd), and domain exchange (lexs-domain-exchange.xsd). What is appropriate depends on the use case. For example, those who are implementing a publish capability require the lexs-publish-discover.xsd, but the other three top-level schemas can be removed.

LEXS-based exchange specifications must incorporate the LEXS schema files as-is; they may not be modified in any way except where noted in this document. Note that some XML editors make subtle changes to schema files when the files are opened, such as converting comments to annotations. However, the LEXS schemas must be included unchanged, so any changes introduced by any XML editor must be reverted back to the original file included with the LEXS specification.

Since the LEXS and Structured Payload schemas are separate and independent schemas, caution is required to ensure that the schemas are kept segregated so that there are no inheritance issues between them, especially with regard to NIEM subset schemas. The Digest includes NIEM subset schemas, and NIEM-based Structured Payloads include their own independent NIEM subset schemas. Structured Payloads must not reference the LEXS NIEM subset schemas or any LEXS extension schemas. Note that the Structured Payload schemas must also be kept segregated from any other Structured Payloads, in the case where a community is building a Structured Payload that incorporates an existing LEXS exchange that already includes one or more Structured Payloads.

If a Structured Payload schema needs elements that are already defined in the Digest schema, then the Structured Payload schema set must include a copy, either original or a subset, of the Digest schema. For example, a Structured Payload schema may need to reference a Watchlist which is defined by LEXS rather than NIEM. The LEXS Digest defines one for use in the Digest, so rather than redefining the Watchlist in the Structured Payload, the payload schema set can include a subset copy of the Digest schema that includes just the Watchlist element. The Structured Payload schema must NOT reference the original Digest schema since that would result in a conflict between the Digest's NIEM subset and the Structured Payload's NIEM subset.

4.2 Split Data Model Relationships among Digest and Structured Payloads

Data in the Digest and Structured Payload must be linked when both represent the same real world object under the split data model approach. Data in such a Structured Payload may also be linked to data in another Structured Payload when they represent the same real world object. The following sections discuss this data linking when using the split data model.

4.2.1 Structured Payload Building upon the Digest

In the case of the split data model, Structured Payload objects build upon Digest objects when both represent the same real world object. In some cases, the Structured Payload object name and Digest object name may be the same, while in other cases, the Structured Payload object may represent a more specific or detailed object than the Digest element. There may even be cases where the Structured Payload object is less specific than the Digest object. Regardless of whether the Structured Payload object is more or less specific than the Digest object, the Structured Payload object must be linked to the corresponding Digest object.

There may be cases where an object can be represented completely using the Digest, in which case there is no corresponding Structured Payload object. There may also be cases where an object cannot be represented in the Digest at all, and so only appears in the Structured Payload.

Structured Payload Objects Building upon the Same Digest Object
A Structured Payload object may build upon the same Digest object, such as a Structured Payload Person object building upon a Digest Person object.

For example, an exchange may need to represent a Person with a name and hair style. The LEXS Digest includes person name as part of Person, but does not include any representation for hair style. However, NIEM does include a hair style text element in Person. Therefore, the NIEM subset schema for the Structured Payload would include the PersonHairStyleText element as part of PersonType. The Person element in the Digest would be linked to the Person element in the Structured Payload in an XML instance through the use of a SameAsConnection element in the Linkages block.

Structured Payload Objects Building upon a Less Specific Digest Object
A Structured Payload object may build upon a less specific Digest object, such as a Structured Payload Jewelry object building upon a Digest Item object.

For example, an exchange may need to represent Jewelry with a jewelry description and a jewelry category code. While the LEXS Digest includes Item, it does not include Jewelry. However, in NIEM, JewelryType extends ItemType, and so provides a basis for a Structured Payload Jewelry element. Therefore, the NIEM subset for the Structured Payload would include the JewelryCategoryCode element as part of JewelryType, but would not include the ItemDescriptionText element since that is already available in the LEXS Digest as part of the Item element. The Item element in the Digest would be linked to the Jewelry element in the Structured Payload in an XML instance through the use of a SameAsConnection element in the Linkages block.

Digest Objects with no Corresponding Structured Payload Object
Even though an exchange may define a Structured Payload that builds upon the Digest, there may be cases where an object required by the exchange can be represented completely in the Digest. In this case, the Structured Payload does not need a corresponding Structured Payload object.

For example, an exchange may require Firearm description and caliber code, which are both available in the LEXS Digest Firearm. Therefore, the Structured Payload does not require a Firearm element.

Structured Payload Objects with no Corresponding Digest Object
While the LEXS Digest includes an extensive set of high-level data objects, there may be cases when an exchange requires an object for which the LEXS Digest cannot provide a reasonable basis.

Duplication of Data between Digest and Structured Payload
In general, with the split data model approach, Structured Payload schemas should not duplicate Digest data elements so there is a consistent representation for consumers to deal with. Having multiple places where the same information can be provided makes it more complicated for consumers since they would need to pay attention to multiple places in order to provide complete understanding of the instance.

There may be cases where this cannot be avoided. For example, the LEXS Digest includes value as part of item. LEXS defines item value as a singly occurring element since in general an item can only have one value. However, an exchange may require multiple values based on other criteria, such as reported value versus actual value, or stolen value versus recovered value. In this case, a Structured Payload would need to include item value as a multiply occurring element, possibly with additional data elements to define what "kind" of value is being provided. As above, the Item element in the Digest would be linked to the Item element in the Structured Payload through the SameAsConnection element.

4.2.2 Structured Payloads Building upon other Structured Payloads

Structured Payloads may build upon other Structured Payloads. In some cases, the object names in both Structured Payloads may be the same, while in other cases one Structured Payload object may represent a more specific or detailed object than the other Structured Payload object. Regardless, objects are linked between Structured Payloads in XML instances through the use of a SameAsConnection element in the Linkages block.

4.3 Considerations for the use of Self-contained Structured Payloads

In the case of self-contained Structured Payloads, Structured Payload objects do not build upon Digest objects but are complete by themselves. In other words, the Structured Payload contains all data needed by the exchange and the Digest contains a summary of the contents of the Structured Payload. With this approach, if a consumer understands the Structured Payload, they would ignore the Digest; if they do not understand the Structured Payload, they would consume the Digest. In some cases, a Structured Payload object name and Digest object name may be the same, while in other cases, the Structured Payload object may represent a more specific or detailed object than the Digest element, such as a "Truck" versus a "Vehicle".

In order to provide interoperability with consumers who may only understand the Digest, if the LEXS self-contained structured payload data model is used, the Digest must be populated with all appropriate Digest data elements that correspond to elements in the Structured Payload. For example, if a Structured Payload includes a person's driver license number, the Digest must also include the person driver license number. Note that there may be cases where it is not "appropriate" for data to be copied if the more sparse data in the Digest would give a consumer an incorrect idea of what the data represents. For example, if the Structured Payload includes an element to indicate that some of the information is believed possible rather than known, the Digest representation of the "possible" information may not be considered an appropriate representation since it may mislead consumers, and can therefore be left out.

Those developing self-contained Structured Payloads may want to consider the use of the Digest schemas as the starting point for the Structured Payload. Code developed by an implementer to process a Digest could be expanded to process additional data included in a Digest-based Structured Payload, which may save significant time compared to developing all-new code for Structured Payloads created from scratch. In this fashion, an implementer could build up their Structured Payload processor over time as they work with additional exchanges. In addition, the use of the Digest as the starting point for a Structured Payload would allow the creation of a tool that could convert any such Structured Payload instance to a Digest instance. For the purposes of this document, it is assumed that self-contained Structured Payloads use the Digest schemas as the starting point for the Structured Payload schemas.

4.4 Structured Payloads and NIEM

Structured Payloads are not required to be NIEM-based, although the general use case for LEXS is for NIEM-based exchanges.

NIEM-based Structured Payloads must be NIEM conformant, meaning they must follow the NIEM Naming and Design Rules (NDR), which is available from the NIEM web site.

Since Structured Payloads are separate schemas, they include their own NIEM subset schemas as well as extension schemas. Communities developing Structured Payloads are anticipated to use NIEM tools, such as the Subset Generation Tool.

4.5 Structured Payloads in LEXS Messages

The LEXS schemas define the Structured Payload as an element that consists of Structured Payload metadata, followed by an xsd:any element.

Structured Payload metadata is required for each Structured Payload, and is used to uniquely identify and describe each Structured Payload. It is used to define the Structured Payload schema, not an individual instance of a Structured Payload.

xsd:any is a XML Schema construct that allows any well-formed content to be provided in an instance in place of the xsd:any element. This allows each community to define their own schemas and possible root elements, and to plug instances conforming to those schemas in place of the xsd:any element. The “processContents” attribute is defined as “skip”, meaning that no validation will be performed; the element must simply be well formed. This ensures that all LEXS instances are valid to the LEXS schemas regardless of the contents of the Structured Payload.

Note that the use of the xsd:any element means that LEXS messages that include a Structured Payload essentially consist of an XML instance encapsulated in another XML instance. Each instance must be validated separately. This doesn’t mean that it is impossible to build, consume, or validate a complete LEXS message in a single pass, just that most tools currently require these “separate” instances be treated separately in a two-pass process.

5 Building Structured Payloads

Communities that are building a Structured Payload must address a number of issues. The implementer needs to decide whether to use the split data model or the self-contained Structured Payload. The community needs to determine what data is required in the Structured Payload based on what is available in the Digest. The community should also evaluate existing information exchanges to determine if an exchange could be re-used. Re-use is preferable to building from scratch in order to lower costs and increase interoperability. An exchange may already exist that doesn’t include all data necessary, in which case the community could re-use the existing exchange and build a second Structured Payload either to supplement the existing Structured Payload (split model) or incorporate the existing Structured Payload (self-contained model).

If a community determines that a new Structured Payload must be constructed, they must determine what objects in the Digest have corresponding objects in the Structured Payload, what objects can exist solely in the Digest, and what objects will exist solely in the Structured Payload.

The basic steps required for building either a split data or self-contained Structured Payload are summarized below. Note that there is significant overlap in the steps required.

Step 1 - Determine Data Contents
Either Split Data Model or Self-contained Structured Payload

a) Digest high-level objects and data elements in each object
b) New high-level objects required
c) New data elements required for existing high-level objects

Step 2 - Create Structured Payload
If Split Data Model Structured Payload

d) Create directory structure
e) Copy Digest schemas into separate directory branch
f) Modify NIEM subset for Digest to add and remove data elements as necessary
g) Modify and/or remove LEXS-defined data contents as necessary
h) Add Structured Payload-specific data objects and data elements as necessary

If Self-contained Structured Payload

d) Create directory structure
e) Create new NIEM subset for Structured Payload
f) Add Structured Payload-specific data objects and data elements as necessary

5.1 General Data Content of Structured Payload

Communities utilizing LEXS may have a well-defined data model, or may be developing the data model as part of the effort of producing a LEXS-based information exchange. Regardless of the situation, the community must review the high-level objects available as part of the LEXS Digest in order to determine what objects can be leveraged from the Digest, and what objects do not map to anything from the Digest. There may be exchanges where a Structured Payload is required to add to another Structured Payload. If a community determines that an object is required for which there is a match in the Digest, the community then needs to determine whether the Digest includes everything necessary, or if additional data elements are required.

For example, the community may need a person object. The Digest includes Person, with a relatively rich set of person data. If the community only needs a person’s first name, last name, social security number, driver license number, and date of birth, the community can use the Digest Person without additional Structured Payload data elements. However, if the information exchange also requires information about a person’s physical features, such as scars or tattoos, that data would need to be supplied in a Structured Payload.

In some cases, the Digest may be able to provide the basis of a required high-level object, but without the required specifics. For example, the Digest includes Item. However, the community may require details about Jewelry. Since Jewelry is an item, the community can use the Digest’s Item for things like description and size, and put jewelry-specific data elements in the Structured Payload.

While the Digest includes a broad range of objects, there are frequently cases where an information exchange requires some object for which the Digest provides no foundation. For example, prior to LEXS 3.1.4 there was nothing in the Digest that could be used to represent a network address, such as an IP or web site address. Therefore, prior to LEXS 3.1.4 any information exchange that required a network address had to provide it in the Structured Payload, with no corresponding data elements in the Digest.

The process of mapping the community data model to the LEXS Digest data model is not limited to objects, but includes associations as well, which specify relationships between objects. The same evaluations are necessary for associations as for objects. Digest associations may not be semantically specific enough, or may not contain all the information the exchange needs to represent in the association. There may also be cases where the exchange needs to indicate a relationship involving one or more objects that do not exist in the Digest.

Once it has been determined what objects and associations will be leveraged from the Digest, and which will exist solely in the Structured Payload, it is time to map at the data element-level. This provides information about which data elements will be used in the Digest objects and associations, and what data elements will be needed in the Structured Payload. The most common mapping artifact is a spreadsheet that lists the data elements needed for the information exchange, along with mapping information that indicates what elements map to LEXS Digest data elements and which map to data elements in the Structured Payload.

Data mappers may have to do some searching to find appropriate matches between their data model and LEXS due to terminology and naming differences. For example, an exchange may need to include information on a police officer, which is represented in NIEM and LEXS through the use of the data element EnforcementOfficial as well as more specific element names such as ArrestOfficial.

An information exchange may include an object that requires the use of multiple objects from LEXS/NIEM. For example, an information exchange may contain information on a police officer, including their name and age and badge number. In NIEM an EnforcementOfficial is a role of a person. So while the data source that is being mapped may include name and age and badge number in a single object, LEXS includes badge number in the EnforcementOfficial role with name and age in Person. Therefore the data mapping must include both the EnforcementOfficial role and the Person base object in order to represent all required information.

It is important to note that mapping should be done between objects and data elements that are semantically equivalent. Mappers should be careful not to fall into the trap of “close enough” or “I’m not using that element for anything else, so I can use it for this” since that leads to information exchanges where it is harder for others to understand what is intended, and may in turn lead to data integrity problems later.

When the mapping is done, work can start on creating the Structured Payload schemas. A subset schema will be needed for any Structured Payload data elements that map to NIEM data elements, plus one or more extension schemas for data elements that are not in the Digest or in NIEM. For the self-contained approach, the extension schema(s) will incorporate the desired Digest objects as well as additional elements not included in the Digest. For the split data model, the extension schema(s) will just contain data elements that are not in the Digest and/or not in NIEM.

5.2 NIEM Subset Schemas

Once is has been determined what data elements must exist in the Structured Payload, the first step in building a NIEM-based Structured Payload is to build a NIEM subset. The NIEM subset consists of data types and data elements from NIEM that are needed for the information exchange. Keep in mind that for a self-contained Structured Payload, data elements that exist in the Digest are duplicated in the Structured Payload subset schema, and for the split data model they are not duplicated.

NIEM provides the Subset Generation Tool (SSGT) to build NIEM subset schemas. While a detailed tutorial on the use of the SSGT is beyond the scope of this document, a few tips may be useful. Note that the information in this document pertains to the SSGT as it existed at the time this document was written; the SSGT will evolve over time based on user feedback. For the split data model, the SSGT can be used to build subset schemas from scratch, while for the self-contained approach the SSGT is used to modify the Structured Payload copy of the Digest subset schemas.

The SSGT allows searching for data elements by type, element name, description, etc. Searches can also be performed on code values from code tables, so for example if the exchange needs to represent an incident or activity that is “closed”, the SSGT can search for code lists that include the word “closed”, allowing the mapper to find a code list with appropriate values and semantics, and then work backwards to find out the data element that uses that code list. Note that semantics are important in addition to code values, so if the model needs to represent whether a door is open or closed, the use of a code list intended to indicate whether an investigation is open or closed would not be appropriate.

Structural pieces need to be included as well to provide a complete data model for the Structured Payload. So for example, if the Structured Payload needs a high-level person object, the subset should include the NIEM Person data element; the SSGT is intelligent enough to include required types and parent types, so in the case of Person, the SSGT will include PersonType. However, it will include it without any data elements. Data elements must be explicitly added to the type. For instance, if the information exchange needs information about a person’s physical feature, such as a tattoo, the PersonPhysicalFeature data element can be included using the SSGT. Since the exchange probably considers a person’s physical feature to be part of Person, subset builders need to make sure that PersonPhysicalFeature is added to the PersonType. If the SSGT is used to just select the PersonPhysicalFeature data element, it will be added to the subset as an “orphan” rather than as part of the parent PersonType. Whether elements are included as part of a type or as an orphan generally depends on the order in which additions are made and how the SSGT is used to arrive at a particular data element. For example, if PersonType is added, and PersonEthnicityCode is added by finding it in PersonType, it will be part of PersonType. Conversely, if PersonType is added, and then PersonEthnicityCode is added by finding it through a search on the word “ethnicity”, it will not be part of PersonType.

If the exchange requires a code list data element, such as a person’s ethnicity, the data element as well as the underlying code types and enumerations must be included. Adding the PersonEthnicityCode element to the subset via the SSGT will result in the inclusion of the underlying code types and values to the subset. The SSGT allows a developer to select all code values or just some of them, so the subset may be built to include only certain values of interest to the community.

Note that NIEM does not provide for extending a code list, only making a subset of it. So if an exchange requires a code list that is an expanded version of a NIEM code list, the existing code list cannot be expanded. Implementers have the choice between putting the new values in a separate list or building a complete list from scratch. Either can be done without violating any NIEM or LEXS rules, although in general having a single complete list is easier from both mapping and code development perspectives.

The SSGT allows developers to set the cardinality of data elements. By default, all elements are optional and unbounded. However, the SSGT allows editing of cardinality to reflect any desired minimum or maximum.

There may be cases where an essentially empty subset of a type is needed, primarily when the split data model is utilized, and the only contents of the NIEM subset is an Augmentation Point element so the Structured Payload augmentation can be incorporated. This empty subset is useful when the Structured Payload schema needs to extend some NIEM type, but where the complete contents will be defined in an extension schema. For example, the Digest includes PersonName, and the Structured Payload may need to include PersonName as well. But if there is no person name data element available in NIEM for the necessary concept, the additional person name element will have to be defined in an extension schema. If the only content of the Structured Payload person name is one or more new extension elements, an empty PersonNameType should be included to contain the Structured Payload augmentation to PersonNameType. The first diagram below shows a subset of PersonNameType that includes only the NIEM PersonNameAugmentationPoint element, and the second diagram shows the PersonName element with the Structured Payload augmentation for an additional name component, such as person hacker name.

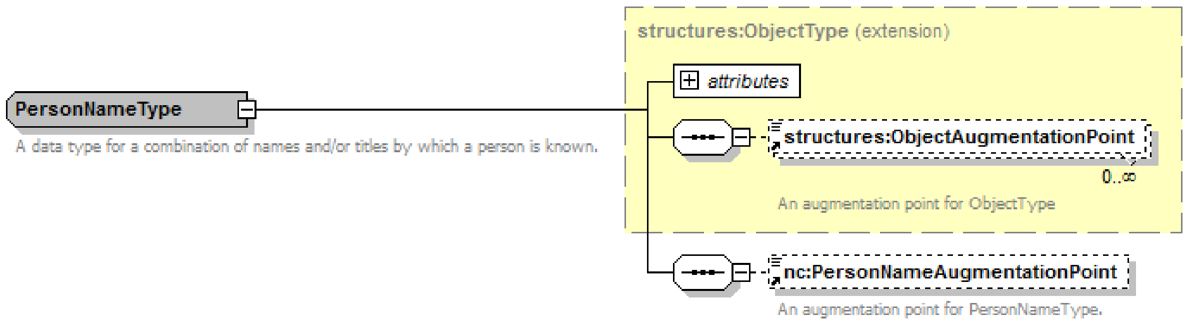

structures:ObjectType (extension)
attributes
PersonNameType
A data type for a combination of names and/or titles by which a person is known.
structures:ObjectAugmentationPoint
0..∞
An augmentation point for ObjectType
nc:PersonNameAugmentationPoint
An augmentation point for PersonNameType.

Figure 1 - NIEM Subset with Only Augmentation Point Element

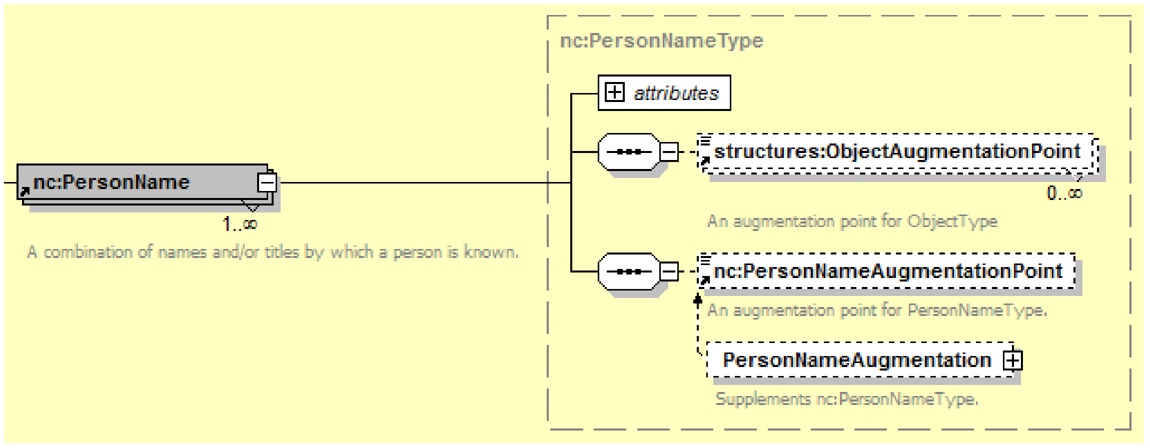

nc:PersonNameType
attributes
nc:PersonName
1..∞
A combination of names and/or titles by which a person is known.
structures:ObjectAugmentationPoint
0..∞
An augmentation point for ObjectType
nc:PersonNameAugmentationPoint
An augmentation point for PersonNameType.
PersonNameAugmentation
Supplements nc:PersonNameType.

Figure 2 - PersonName Element Augmented with Structured Payload Element

5.3 Code List Extension Schemas

Structured Payload schemas may need to define code list elements that include well-defined enumerations. NIEM provides a code list generation tool that converts an Excel spreadsheet into a code list schema. The tool generates a separate code list type for each tab in the spreadsheet.

So if an exchange has four IEPD-specific code lists, the spreadsheet would include four tabs. The NIEM tools web page provides a template for the code list spreadsheet. Basically, each tab contains the name of the code type to be used, a definition for the type, and then a list of code values and the definition for each. The figure below shows part of a tab in a code list spreadsheet.

LocationGeneralCategoryCodeType	
A data type for codes that identify the general type of location.	
Code	**Description**
Commercial	A location or area where goods or services are exchanged for profit, e.g., store, restaurant, office building, daycare, financial institution, hotel, etc.
Open Area	A location or area with little to no building structures, e.g., field, camp, woods, lake, right of way, cemetery, etc.
Public Non Residential	A location or area open to the public, and possibly supported by tax dollars, e.g., school, medical center, terminal, government building, fire department, religious building, or meeting hall.
Residential	A location or area where persons or groups of persons reside, e.g., house, townhouse, mobile home, etc.
Industrial	A location or area where goods are created or assembled using mechanical equipment, e.g., factory, plant, mine, assembly line.
Unknown	Unknown

Figure 3 - Example code list spreadsheet tab

In general, code values should be self-documenting. So for example, the use of the word "Commercial" above is preferable to just the letter "C" or the abbreviation "Com." as the code value. With self-documenting code values, the code value and the definition may be the same, although the definition may include additional information or examples as in the figure above.

Codes must be valid XML Schema NMToken values. The description can include characters such as commas, single quotes, and parenthesis. It is recommended that descriptions not contain either forward or backward slashes as these may cause issues with some software tools.

The spreadsheet can then be uploaded to the NIEM code list generation tool, which will generate a zip file containing a code list schema based on the spreadsheet, as well as supporting NIEM schema files. The code list generation tool requires a namespace and namespace prefix to be defined. It is recommended that the namespace be the same as planned for the extension schema, but with the word "code" or "codes" included. So for example if the namespace for the extension schema is to be "http://somewhere.gov/new-community", the namespace for the code list schema might be "http://somewhere.gov/new-community/codes". Note that the namespace should reflect the organization and the exchange, rather than generic names as shown above.

5.4 Extension Schemas

Once the NIEM subset has been created and any code list schemas generated, an extension schema can be built. (While the code list schema is an extension schema as well, we use the term extension schema here to refer to extension schemas that are not code list schemas.) The extension schema must follow the NIEM NDR if the Structured Payload is NIEM-based. The following discussion refers to the definition of data elements; however, keep in mind that elements are defined to be of specific types per the NDR rather than defined in-line without explicit type definitions. Examples show explicit types, but the document text just refers to elements in order to simplify the discussion.

A Structured Payload may include multiple extension schemas. This may be because the exchange includes data types and elements from different agencies and therefore namespaces. Some organizations may define agency standards through the use of a “standard” extension schema(s) that must be included. This document assumes a single extension schema. The primary differences when there are multiple extension schemas have to do with ensuring that schema imports are complete, and the fact that an extension schema may extend types from a different extension schema rather than just the subset schemas as documented here.

The Structured Payload object/association needs to be created by extending some NIEM type. The NIEM type may be the ObjectType when NIEM does not contain the concept that needs to be represented. Examples later in this section illustrate extension of NIEM types, including the NIEM ObjectType. There may be cases where the Structured Payload object/association is a specialization of a NIEM concept rather than an augmentation. Augmentation is used when a type is being extended to add content to an existing type. For example, if an exchange needs to add data elements that do not exist in NIEM or LEXS to Vehicle, then Vehicle would be augmented to add the additional content. Specialization is used when a type is creating a new, specialized version of an existing type, for example if an exchange needs to create a ConstructionVehicle element that defines a more specialized version of the NIEM Vehicle.

The first step is to create the basic structure for the extension schema, along with one or more root elements, into which everything else will go. Then a code list schema can be created if needed, along with an extension schema that defines the Structured Payload’s objects and associations. These various steps are described in the following sections. While these sections cover steps in a certain sequence, that doesn’t mean that implementers have to follow this pattern exactly. Development of Structured Payload schemas is generally an iterative process where certain objects are defined and fleshed out with detailed contents and code lists, followed by other objects. So the sections below describe the steps, but implementers are free to use them in any sequence desired.

5.4.1 Creating a Template for the Extension Schema

The extension schema must define a namespace specific to the extension schema, and import all the subset schema files. During development of the extension schema, the subset schema may need to be updated. When that happens, the import statements may need to be updated either to add a new subset namespace or to remove one that is no longer being used. In addition, the following must be included as part of the extension schema:

- If a code list schema has been created, it must also be imported into the extension schema file.
- The extension schema must include a documentation element to describe the extension schema.
- If the extension schema properly adheres to the NIEM NDR, the NIEM Extension Schema Document conformance target must be provided.

- The schema version attribute must be provided.

The first step in creating the extension schema is to create a template to which other things will be added. The figure below shows an example template for an extension schema.

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:i="http://release.niem.gov/niem/appinfo/3.0/"
  xmlns:ct="http://release.niem.gov/niem/conformanceTargets/3.0/"
  xmlns:new="http://somewhere.gov/new-community/1.0"
  xmlns:newcodes="http://somewhere.gov/new-community-codes/1.0"
  xmlns:nc="http://release.niem.gov/niem/niem-core/3.0/"
  xmlns:niem-xsd="http://release.niem.gov/niem/proxy/xsd/3.0/"
  xmlns:s="http://release.niem.gov/niem/structures/3.0/"
  targetNamespace="http://somewhere.gov/new-community/1.0"
  version="1.0"
  ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-and-design-
rules/3.0/#ExtensionSchemaDocument">

  <xsd:import namespace="http://release.niem.gov/niem/conformanceTargets/3.0/"
        schemaLocation="../../../niem-lexs/conformanceTargets/3.0/conformanceTargets.xsd"/>
  <xsd:import namespace="http://release.niem.gov/niem/structures/3.0/"
        schemaLocation="../../../niem-lexs/structures/3.0/structures.xsd"/>
  <xsd:import namespace="http://release.niem.gov/niem/appinfo/3.0/"
        schemaLocation="../../../niem-lexs/appinfo/3.0/appinfo.xsd"/>
  <xsd:import namespace="http://release.niem.gov/niem/niem-core/3.0/"
        schemaLocation="../../../niem-lexs/niem-core/3.0/niem-core.xsd"/>
  <xsd:import namespace="http://somewhere.gov/new-community-codes/1.0"
        schemaLocation="../../new-codes/1.0/new-codes.xsd"/>
  <xsd:import namespace="http://release.niem.gov/niem/proxy/xsd/3.0/"
        schemaLocation="../../../niem-lexs/proxy/xsd/3.0/xs.xsd"/>

  <xsd:annotation>
    <xsd:documentation>XYZ extension schema</xsd:documentation>
  </xsd:annotation>

 ……
</xsd:schema>
```

Version number

Conformance target

Extension code list schema import

Documentation element

Figure 4 - Sample Extension Schema Template

5.4.2 Defining Root Element(s)

The extension schema must define at least one element that will be used as a root element in Structured Payload instances, and this element is generally named for the data item being represented. For example, if instances are intended to represent an incident report, the element IncidentReport would be an appropriate root element for the Structured Payload extension schema. There can be more than one element defined for use as a root element. For example, instances might represent either an incident report or an arrest report, in which case the elements IncidentReport and ArrestReport could be defined for use as root elements.

The root element(s) would then be defined to contain the objects/associations for the data item. Continuing with the incident and arrest report examples, both report elements could be defined to contain all objects/associations as determined during the mapping step. If there are differences between what an incident report contains versus an arrest report, the IncidentReport and ArrestReport elements may be defined differently. The designer must balance the benefits of defining the root elements differently in schema in order to provide schema enforcement of content rules versus defining the root elements the same but with business rules to indicate what is allowed in each.

Keep in mind that the root element is for XML instances generated based on the Structured Payload schema. As noted previously, a LEXS instance that utilizes a Structured Payload is really a Structured Payload instance inserted into a LEXS instance. The root element for the complete XML instance is defined by LEXS, such as doPublish. The diagram below illustrates this nesting, using “Report” as the Structured Payload root element. Note that the example has been simplified significantly for readability.

```
<lexspd:doPublish>
    <lexs:PublishMessageContainer>
        <lexs:PublishMessage>
            <lexsmeta:MessageMetadata>
                .....
            </lexsmeta:MessageMetadata>

            <lexs:DataItemPackage>
                <lexs:PackageMetadata>
                    ....
                </lexs:PackageMetadata>
                <lexsdigest:Digest>
                    <lexsdigest:EntityPerson s:id="EPerson1">
                        <nc:Person s:id="Person1">
                            ....
                        </nc:Person>
                    </lexsdigest:EntityPerson>
                </lexsdigest:Digest>
                <!--================== Data from another community ==================-->
                <lexs:StructuredPayload lexslib:id="SP1">
                    <lexs:StructuredPayloadMetadata>
                        <lexs:CommunityURI>http://somewhere.gov/new-community</lexs:CommunityURI>
                        <lexs:CommunityDescriptionText>New Community</lexs:CommunityDescriptionText>
                        <lexs:CommunityVersionText>1.0</lexs:CommunityVersionText>
                    </lexs:StructuredPayloadMetadata>
                    <lexs:StructuredPayloadContent>
                        <new:Report xmlns:new="http://somewhere.gov/new-community">
                            <nc:Person s:id="SPPerson1">
                                <nc:PersonHairStyleText>pageboy</nc:PersonHairStyleText>
                            </nc:Person>
                        </new:Report>
                    </lexs:StructuredPayloadContent>
                </lexs:StructuredPayload>
                <!--========  Link objects from the structured payload to the digest  =========-->
                <lexs:Linkages>
                    <lexslib:SameAsConnection>
                        <lexslib:DigestObjectReference lexslib:validatingObjectReference="Person1"/>
                        <lexslib:StructuredPayloadObjectReference
                                    lexslib:structuredPayloadReference="SP1"
                                    lexslib:nonValidatingObjectReference="SPPerson1"/>
                    </lexslib:SameAsConnection>
                </lexs:Linkages>
            </lexs:DataItemPackage>
        </lexs:PublishMessage>
    </lexs:PublishMessageContainer>
</lexspd:doPublish>
```

Figure 5 - Structured Payload Root Element Inside a LEXS Instance

The content inside the box is defined by the Structured Payload, with the remainder of the instance defined by the LEXS schemas. The Structured Payload root element “Report” is embedded inside the LEXS StructuredPayloadContent element.

The Structured Payload extension schema may include a number of different objects and associations. Some may build upon the contents of a Digest object or association, or even build upon the contents of another Structured Payload. Still others may exist only in the Structured Payload, without any corresponding content in the Digest or another Structured Payload. These scenarios are covered in the following sections.

Structured Payload designers have flexibility in defining their root elements and the types for their root elements. If there is only one root element, only one root element type is required. If multiple root elements are needed, but the contents are the same, it may be beneficial to define a single report or document type and make all the root elements of that type. If multiple root elements are needed and the contents of each are different, then multiple root element types will have to be created. It all depends on the needs of the exchange. In the descriptions below, a single ReportType has been created, with multiple root elements of that type; such as IncidentReport.

5.4.3 Incorporating Objects and Associations in the Root Element

The Structured Payload extension schema defines high-level objects and associations that define the contents of Structured Payload instances. Regardless of whether these objects and associations utilize NIEM only, community content only, or some combination, the objects and associations must be incorporated into the root element(s).

The root element type(s) may define the contents as a sequence that includes all possible content in a fixed sequence, or may group the contents into categories, such as Entity and Association as used in the LEXS Digest. The examples below illustrate these two approaches. For these examples, the Structured Payload includes the objects Incident, Jewelry, Person, and Subject (a Role), plus the associations FinancialSupporterAssociation, and ItemOwnerAssociation. Some objects and associations are defined by the Structured Payload while others are part of a NIEM subset. In order to simplify the examples, the definition of some types is not included, and documentation and annotation elements have been removed.

The first example and corresponding diagram below show two organizational groups: Entity and Association, which make up the contents of the ReportType. This grouping may be advantageous where there are many high-level objects and associations, and the grouping helps to categorize the content.

```
<!--   Root Element Type and Elements  -->
<xsd:complexType name="ReportType">
   <xsd:complexContent>
      <xsd:extension base="s:ObjectType">
         <xsd:sequence>
            <xsd:element ref="new:Entity" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="new:Association" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="ArrestReport" type="new:ReportType" nillable="true"/>
<xsd:element name="IncidentReport" type="new:ReportType" nillable="true"/>

<!--    Organizational Category Types and Elements   -->
<xsd:complexType name="AssociationGroupType">
   <xsd:complexContent>
      <xsd:extension base="s:ObjectType">
         <xsd:sequence>
            <xsd:element ref="nc:ItemOwnerAssociation" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="new:FinancialSupporterAssociation" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="EntityGroupType">
   <xsd:complexContent>
      <xsd:extension base="s:ObjectType">
         <xsd:sequence>
            <xsd:element ref="nc:Incident" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="j:Jewelry" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="nc:Person" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="j:Subject" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="new:Tool" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Association" type="new:AssociationGroupType"/>
<xsd:element name="Entity" type="new:EntityGroupType"/>

<!-- ============================================================== -->
<!-- Structured Payload-defined Entity Elements -->
<!-- ============================================================== -->
<xsd:element name="Tool" type="new:ToolType" nillable="true"/>

<!-- ============================================================== -->
<!-- Structured Payload-defined Association Elements-->
<!-- ============================================================== -->
<xsd:element name="FinancialSupporterAssociation" type="new:FinancialSupporterAssociationType"/>
```

Figure 6 - Root Element Using Organizational Grouping Elements in Schema

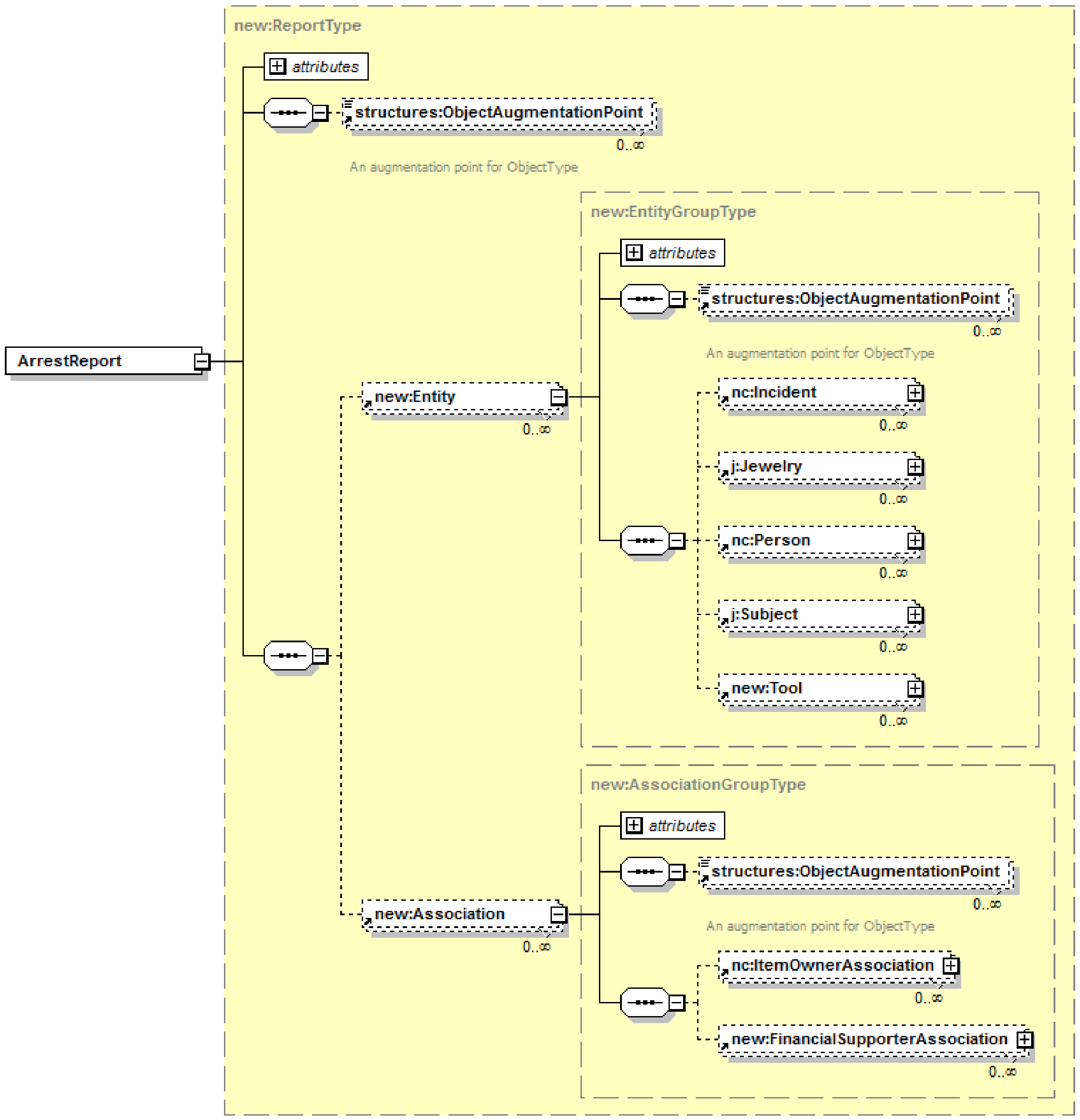

new:ReportType
attributes
structures:ObjectAugmentationPoint
0..∞
An augmentation point for ObjectType
ArrestReport
new:Entity
0..∞
new:EntityGroupType
attributes
structures:ObjectAugmentationPoint
0..∞
An augmentation point for ObjectType
nc:Incident
0..∞
j:Jewelry
0..∞
nc:Person
0..∞
j:Subject
0..∞
new:Tool
0..∞
new:Association
0..∞
new:AssociationGroupType
attributes
structures:ObjectAugmentationPoint
0..∞
An augmentation point for ObjectType
nc:ItemOwnerAssociation
0..∞
new:FinancialSupporterAssociation
0..∞

Figure 7- Root Element Using Organizational Grouping Elements Illustrated

The next example and diagram show the root element type defined with all high-level objects and associations combined into a single sequence.

```
<!--   Root Element Type and Elements  -->
<xsd:complexType name="ReportType">
   <xsd:complexContent>
      <xsd:extension base="s:ObjectType">
         <xsd:sequence>
            <xsd:element ref="nc:Incident" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="j:Jewelry" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="nc:Person" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="j:Subject" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="new:Tool" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="nc:ItemOwnerAssociation" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="new:FinancialSupporterAssociation" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="ArrestReport" type="new:ReportType" nillable="true"/>
<xsd:element name="IncidentReport" type="new:ReportType" nillable="true"/>

<!-- ================================================================ -->
<!-- Structured Payload-defined Entity Elements -->
<!-- ================================================================ -->
<xsd:element name="Tool" type="new:ToolType" nillable="true"/>

<!-- ================================================================ -->
<!-- Structured Payload-defined Association Elements-->
<!-- ================================================================ -->
<xsd:element name="FinancialSupporterAssociation" type="new:FinancialSupporterAssociationType"/>
```

Figure 8 - Root Element Using a Combined Sequence in Schema

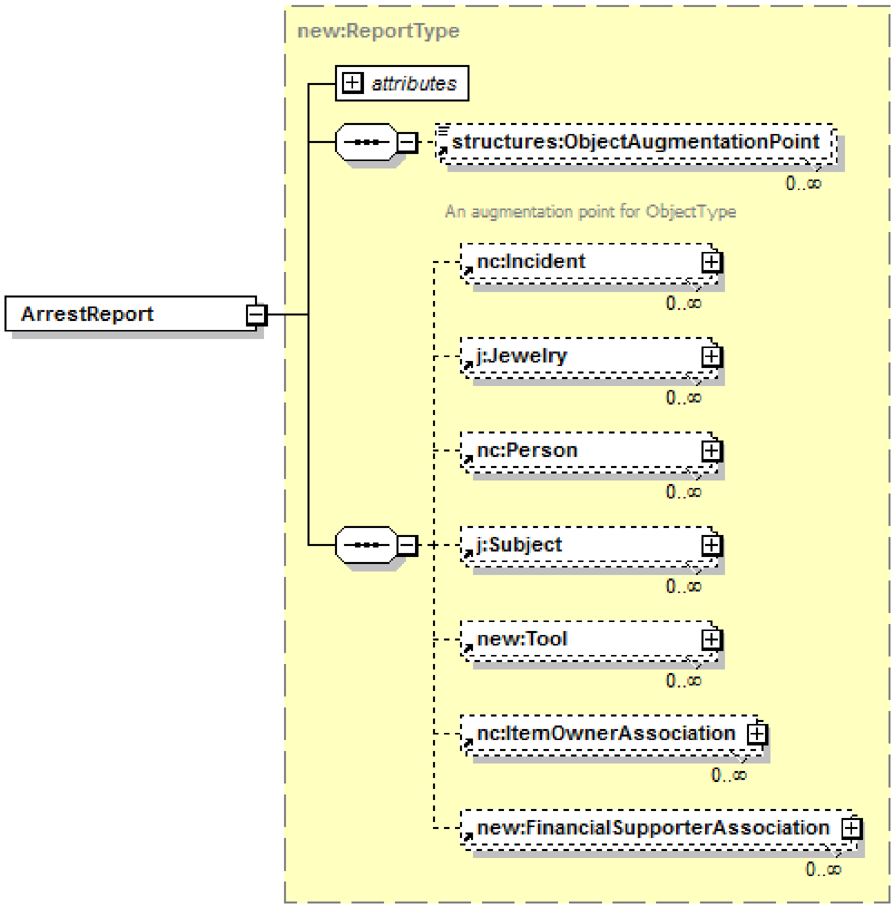


Figure 9 - Root Element Using a Combined Sequence Illustrated

5.4.4 Building Upon a Digest Object/Association

In some cases, the NIEM subset may include all the data that is required for the Structured Payload object/association, while in other cases there may be a need for additional, non-NIEM elements. It is also important to remember that references in the Structured Payload, whether for associations or roles, must refer to elements in the Structured Payload. For the split data model, there may even be cases where the Structured Payload element contains no data, but is only in the Structured Payload to provide an endpoint that a Structured Payload association or role reference can refer to.

Regardless of whether the Structured Payload and Digest objects are the same (e.g. both Person) or different (e.g. Jewelry versus Item), the basics for building the Structured Payload objects/associations are the same.

Object/Association Utilizing only NIEM Subset Elements

If the Structured Payload object/association only utilizes NIEM subset elements, then the only remaining work is to incorporate the NIEM element into the Structured Payload root element; no types or elements need to be created. A simple example is shown below, where the NIEM subset for the Structured Payload includes a person with the nc:PersonHairStyleText element. This example follows the split data model approach and shows the Structured Payload content building upon the Digest content. In order to simplify the examples, documentation and annotation elements have been removed.

```
<!--    Root Element Type and Elements  -->
<xsd:complexType name="ReportType">
   <xsd:complexContent>
      <xsd:extension base="s:ObjectType">
         <xsd:sequence>
            <xsd:element ref="nc:Person" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="IncidentReport" type="new:ReportType" nillable="true"/>
```

Figure 10 –Sample Structured Payload Object with only NIEM Subset Elements in Schema

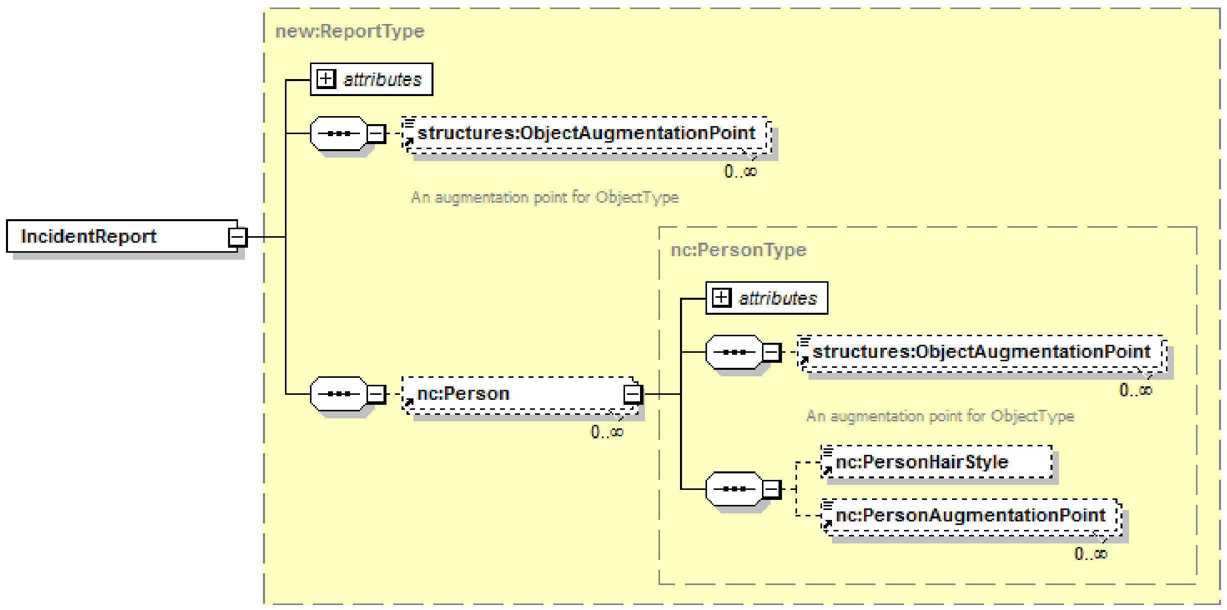

new:ReportType
attributes
structures:ObjectAugmentationPoint
0..∞
An augmentation point for ObjectType
IncidentReport
nc:PersonType
attributes
structures:ObjectAugmentationPoint
0..∞
An augmentation point for ObjectType
nc:Person
0..∞
nc:PersonHairStyle
nc:PersonAugmentationPoint
0..∞

Figure 11- Sample Structured Payload Object with only NIEM Subset Elements Illustrated

Object/Association with Subset Elements plus Additional Content

If the Structured Payload needs to include additional data elements beyond what is available in a NIEM subset, an AugmentationType can be created as shown below, in this case to add an element for a person’s attitude. NIEM 3.0 introduced a new mechanism for augmenting types, providing an abstract augmentation point element which in this case is substituted by a

Structured Payload-defined augmentation element. In order to simplify the examples, documentation and annotation elements have been removed.

```
<!--   Root Element Type and Elements  -->
<xsd:complexType name="ReportType">
   <xsd:complexContent>
      <xsd:extension base="s:ObjectType">
         <xsd:sequence>
            <xsd:element ref="nc:Person" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="IncidentReport" type="new:ReportType" nillable="true"/>

<!--   Structured Payload-defined Type and Element  -->
<xsd:complexType name="PersonAugmentationType">
   <xsd:complexContent>
      <xsd:extension base="s:AugmentationType">
         <xsd:sequence>
            <xsd:element ref="new:PersonAttitude" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="PersonAttitude" type="nc:TextType" nillable="true"/>
<xsd:element name="PersonAugmentation" type="new:PersonAugmentationType"
                  substitutionGroup="nc:PersonAugmentationPoint" nillable="true"/>
```

Figure 12 – Structured Payload Object with Augmentation in Schema

The additional element for the person’s attitude is included as part of the augmentation since the new element is “augmenting” the existing NIEM PersonType rather than creating a specialization of PersonType. The example also shows the inclusion of a NIEM augmentation point element that is part of the subset schema. The diagram below shows the same schema details, where the PersonAugmentation element is substituted for the PersonAugmentationPoint element. Note that in NIEM, PersonHairStyleText is substitutable for PersonHairStyle.

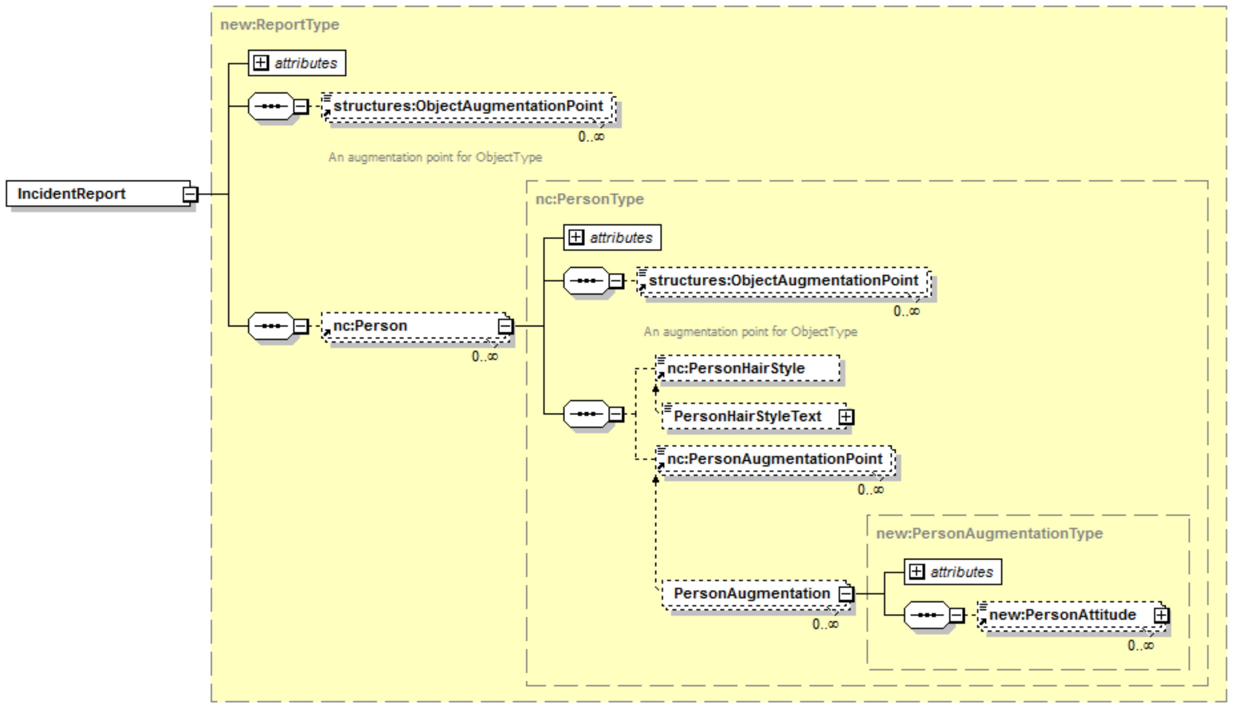

new:ReportType
attributes
structures:ObjectAugmentationPoint
0..∞
An augmentation point for ObjectType
IncidentReport
nc:Person
0..∞
nc:PersonType
attributes
structures:ObjectAugmentationPoint
0..∞
An augmentation point for ObjectType
nc:PersonHairStyle
PersonHairStyleText
nc:PersonAugmentationPoint
0..∞
new:PersonAugmentationType
attributes
PersonAugmentation
0..∞
new:PersonAttitude
0..∞

Figure 13 - Structured Payload Object with Augmentation Illustrated

The figure below shows what the augmentation would look like in an XML instance.

```
<new:IncidentReport>
    <nc:Person>
    <nc:PersonHairStyleText>braids</nc:PersonHairStyleText>
        <new:PersonAugmentation>
            <new:PersonAttitude>shy</new:PersonAttitude>
        </new:PersonAugmentation>
    </nc:Person>
</new:IncidentReport>
```

Figure 14 – Structured Payload Object with Augmentation as XML

Specialization of a Digest Object/Association

The Structured Payload may need to provide a more specialized version of a Digest object or association. The example below illustrates a Structured Payload association building upon a Digest association, in this case to indicate more detailed semantics for the Digest association. In this example, the Structured Payload includes a FinancialSupporterAssociation, which can build upon a PersonAssociation in the Digest to indicate that a person provides financial support to another person. Note that the association doesn't indicate which person supplies the financial support and which person receives it; if such detail were necessary in the exchange a Structured Payload association would need to be created with specific references (e.g. supporter reference versus receiver reference). However, the intent here is to just indicate that not only is there a person-to-person relationship, but that the relationship includes financial support of one for the other.

```
<xsd:complexType name="FinancialSupporterAssociationType">
   <xsd:annotation>
      <xsd:documentation>A data type for a relationship between a person and another person that provides
                          financial support or assistance.</xsd:documentation>
   </xsd:annotation>
   <xsd:complexContent>
      <xsd:extension base="nc:PersonAssociationType">
         <xsd:sequence>
            <xsd:element ref="new:FinancialSupportDescriptionText" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="FinancialSupporterAssociation" type="new:FinancialSupporterAssociationType">
   <xsd:annotation>
      <xsd:documentation>A relationship between a person and another person that provides financial support or
                          assistance.</xsd:documentation>
   </xsd:annotation>
</xsd:element>

<xsd:element name="FinancialSupportDescriptionText" type="nc:TextType">
   <xsd:annotation>
      <xsd:documentation>A description of the financial support or assistance provided by the financial supporter to
                          the receiver.</xsd:documentation>
   </xsd:annotation>
</xsd:element>
```

Figure 15 - Specialization of Digest Association in Schema

Depending on whether the Structured Payload utilizes the split data model or the self-contained approach, the corresponding Structured Payload XML would look like one of the two examples below.

```
<new:FinancialSupporterAssociation s:id="FinSup1">
   <new:FinancialSupportDescriptionText>Monthly payment of $2,000</new:FinancialSupportDescriptionText>
</new:FinancialSupporterAssociation>
```

Figure 16 - Specialization of Digest Association in XML - Split Data Model

```
<new:FinancialSupporterAssociation>
   <nc:AssociationDateRange>
      <nc:StartDate>
         <nc:Date>2013-01-01</nc:Date>
      </nc:StartDate>
      <nc:EndDate>
         <nc:Date>2016-01-31</nc:Date>
      </nc:EndDate>
   </nc:AssociationDateRange>
   <nc:Person  s:ref="Person1"/>
   <nc:Person  s:ref="Person2"/>
   <new:FinancialSupportDescriptionText>Monthly payment of $2,000</new:FinancialSupportDescriptionText>
</new:FinancialSupporterAssociation>
```

Figure 17 - Specialization of Digest Association - Self-contained Data Model

5.4.5 Representing an Object/Association Not Built on the Digest or Another Structured Payload

There may be cases where an information exchange includes a concept for which there is no foundation in the Digest or another Structured Payload, in other words, a concept that is represented solely in this Structured Payload. In some cases, NIEM may provide the object, or at least some portion of it, while in other cases there may be nothing in NIEM that can provide the foundation of the concept. The basics in both cases are very similar, although there are some differences.

Corresponding NIEM Object/Association Available
In some cases, the concept that needs to be represented exists in NIEM, but not in the LEXS Digest or any other Structured Payload that is being used in the exchange. In this case, the NIEM type must be included in the Structured Payload's NIEM subset, along with any other data elements that are necessary for the exchange. If the NIEM type does not contain everything required, then the NIEM type can be augmented to add data elements.

For example, assume that an exchange needs to be able to represent an airport as a high-level object, including the airport name and an identification number. NIEM includes the concept of a Facility, where FacilityType includes FacilityName and FacilityIdentification. NIEM also includes an Airport element of FacilityType. In this case, the NIEM subset would be created to include FacilityType, along with its data elements FacilityName and FacilityIdentification, and the data element Airport. The Airport element would be added to the Structured Payload extension schema where appropriate so it would be available for use in instances.

As a more complex example, assume that an exchange needs to be able to represent a truck repair depot, including the depot name, an identification number, and a list of truck makes that the depot can service. FacilityType, along with FacilityName and FacilityIdentification, is still an appropriate foundation for a truck depot. However, the type doesn't include a list of truck makes, so that would have to be added. In addition, NIEM doesn't include a data element for a truck depot. The truck depot could be considered a semantic name for a facility, in which case we could augment NIEM's FacilityType to add the missing data element that is needed. For illustrative purposes, assume that a truck depot is a special kind of facility, in the same fashion as NIEM defines a vessel as a specialized kind of conveyance. Therefore, rather than augmenting FacilityType, we will create a specialization as shown below. The NIEM subset must still include FacilityType with contents FacilityName and FacilityIdentification. This example also uses the NIEM VMACodeType as the foundation for the element indicating the truck makes supported by the depot.

```
<xsd:complexType name="TruckDepotType">
   <xsd:complexContent>
      <xsd:extension base="nc:FacilityType">
         <xsd:sequence>
            <xsd:element ref="new:SupportedTruckMakeCode" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="TruckDepot" type="new:TruckDepotType">
   <xsd:annotation>
      <xsd:documentation>A repair facility specializing in truck repair.</xsd:documentation>
   </xsd:annotation>
</xsd:element>

<xsd:element name="SupportedTruckMakeCode" type="ncic:VMACodeType" nillable="true">
   <xsd:annotation>
      <xsd:documentation>A code for a truck make that can be accommodated by facility.</xsd:documentation>
   </xsd:annotation>
</xsd:element>
```

Figure 18 - Structured Payload-only Object Based on a Specialized NIEM Object in Schema

The example below shows the use of this truck depot element as it might appear in an instance.

```
<new:TruckDepot>
    <nc:FacilityIdentification>
        <nc:IdentificationID>00000003</nc:IdentificationID>
    </nc:FacilityIdentification>
    <nc:FacilityName>Northern Truck Repair</nc:FacilityName>
    <new:SupportedTruckMakeCode>GMC</new:SupportedTruckMakeCode>
    <new:SupportedTruckMakeCode>KW</new:SupportedTruckMakeCode>
</new:TruckDepot>
```

Figure 19 - Structured Payload-only Object Based on a Specialized NIEM Object in XML

No Foundation Available in NIEM

In some cases, there are no objects or associations available in NIEM that can be utilized, either as is or with extension, to represent the concept required by the exchange. Therefore, the object/association must be created from scratch as part of the Structured Payload extension schema. NIEM objects are all derived, at their lowest-level, from ObjectType, which can be used as the basis for Structured Payload objects that cannot derive from any higher-level NIEM objects.

For the example below, the exchange needs an object NewThing which doesn’t correspond to any existing NIEM or LEXS objects or associations. For simplicity, assume that this new object consists of a simple string element and a text description. This example leverages the NIEM Description Text element rather than creating one specific for this NewThing object, although the schema could create a new element with semantics specific to this NewThing, such as NewThingDescriptionText.

```
<xsd:complexType name="NewThingType">
  <xsd:complexContent>
    <xsd:extension base="s:ObjectType">
      <xsd:sequence>
        <xsd:element ref="new:NewThingElement" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="nc:DescriptionText" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="NewThing" type="new:NewThingType">
  <xsd:annotation>
    <xsd:documentation>A new thing that doesn't have a foundation in NIEM or LEXS.</xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="NewThingElement" type="nc:TextType" nillable="true">
  <xsd:annotation>
    <xsd:documentation>A new element not in NIEM or LEXS.</xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

Figure 20 - Structured Payload-only Object Based on ObjectType in Schema

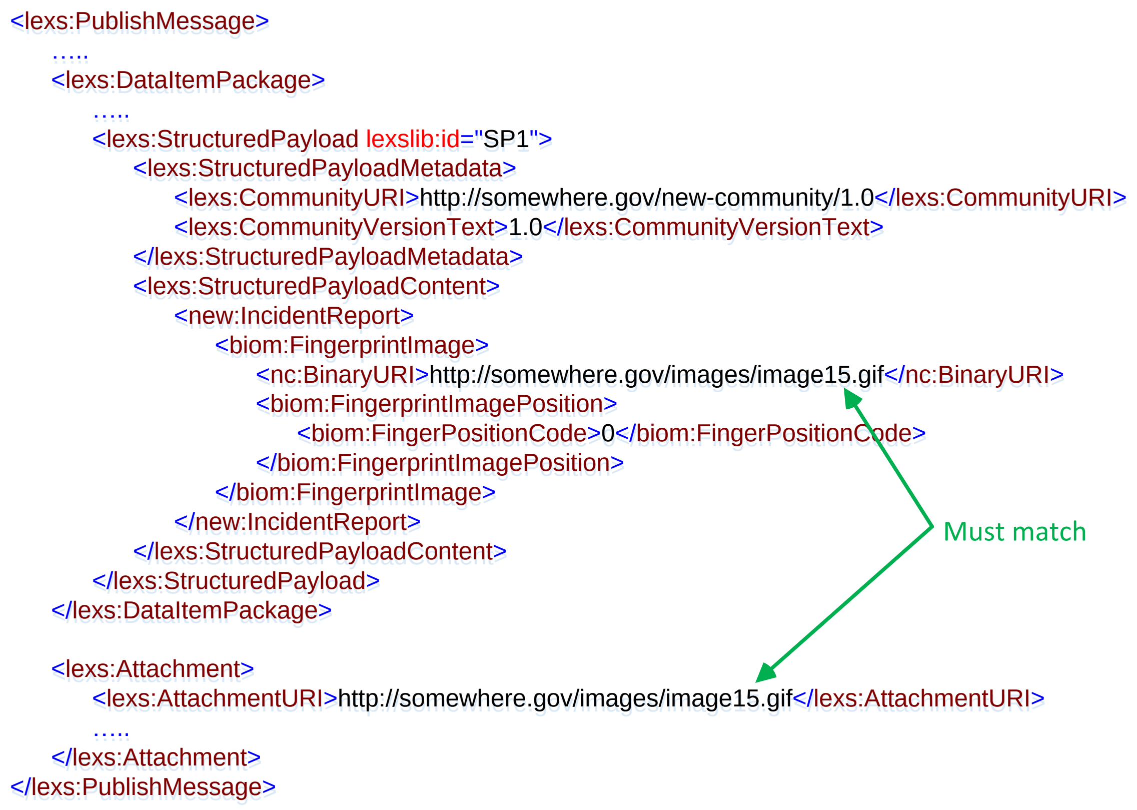
5.4.6 Building Upon an Object/Association in a Different Structured Payload

A Structured Payload may need to build upon (split data model) or include (self-contained data model) the contents of another Structured Payload. The existing Structured Payload object or association may in turn build upon or include Digest contents, or may not. In either case, the creation of the new Structured Payload object or association does not differ significantly from that described previously for building upon a Digest object or association.

For the split data mode, the only difference between building upon something in the Digest versus another Structured Payload has to do with linking the various pieces together in an instance using the LEXS SameAsConnection since these connections will be between two Structured Payloads instead of between a Digest and a Structured Payload.

5.4.7 Building Upon a LEXS Attachment

There may be cases where a Structured Payload needs to extend or specialize a LEXS Attachment. For example, an extension might be needed for Attachment to indicate how an image or video was taken. A specialization, from NIEM or otherwise, may be needed to differentiate between different types of attachment; for example, an exchange might include images with specifics on how they were taken, or fingerprints that include information on which finger the image is for and what kind of machine was used to capture it. The specifics for extending Attachment are no different from extending a Digest object except for the use of the LEXS AttachmentURI to link the LEXS Attachment to the Structured Payload version. The XML example below shows a specialization using the NIEM fingerprint image, and is based on the split data model approach. The NIEM Fingerprint Image includes a BinaryURI, which can be used to link to the LEXS Attachment. Note that the BinaryURI in the Structured Payload instance must match the AttachmentURI of the corresponding LEXS Attachment.



```
<lexs:PublishMessage>
    …..
    <lexs:DataItemPackage>
        …..
        <lexs:StructuredPayload lexslib:id="SP1">
            <lexs:StructuredPayloadMetadata>
                <lexs:CommunityURI>http://somewhere.gov/new-community/1.0</lexs:CommunityURI>
                <lexs:CommunityVersionText>1.0</lexs:CommunityVersionText>
            </lexs:StructuredPayloadMetadata>
            <lexs:StructuredPayloadContent>
                <new:IncidentReport>
                    <biom:FingerprintImage>
                        <nc:BinaryURI>http://somewhere.gov/images/image15.gif</nc:BinaryURI>
                        <biom:FingerprintImagePosition>
                            <biom:FingerPositionCode>0</biom:FingerPositionCode>
                        </biom:FingerprintImagePosition>
                    </biom:FingerprintImage>
                </new:IncidentReport>
            </lexs:StructuredPayloadContent>
        </lexs:StructuredPayload>
    </lexs:DataItemPackage>

    <lexs:Attachment>
        <lexs:AttachmentURI>http://somewhere.gov/images/image15.gif</lexs:AttachmentURI>
        …..
    </lexs:Attachment>
</lexs:PublishMessage>
```

Figure 21 - Extending a LEXS Attachment in XML

5.4.8 Representing Split Data Model Associations Involving Digest and Structured Payload Objects

As noted previously, Structured Payloads are based on a separate schema(s), and therefore instances based on them must be valid as standalone instances. With the split data model, this creates a complication when a new association is required in the Structured Payload that references an object that only exists in the Structured Payload. If the other "end" of that association exists in the Digest, there is a temptation to link the object in the Structured Payload directly to the object in the Digest. However, this would result in an instance based on the Structured Payload to fail XML validation since one of the references would not be resolvable.

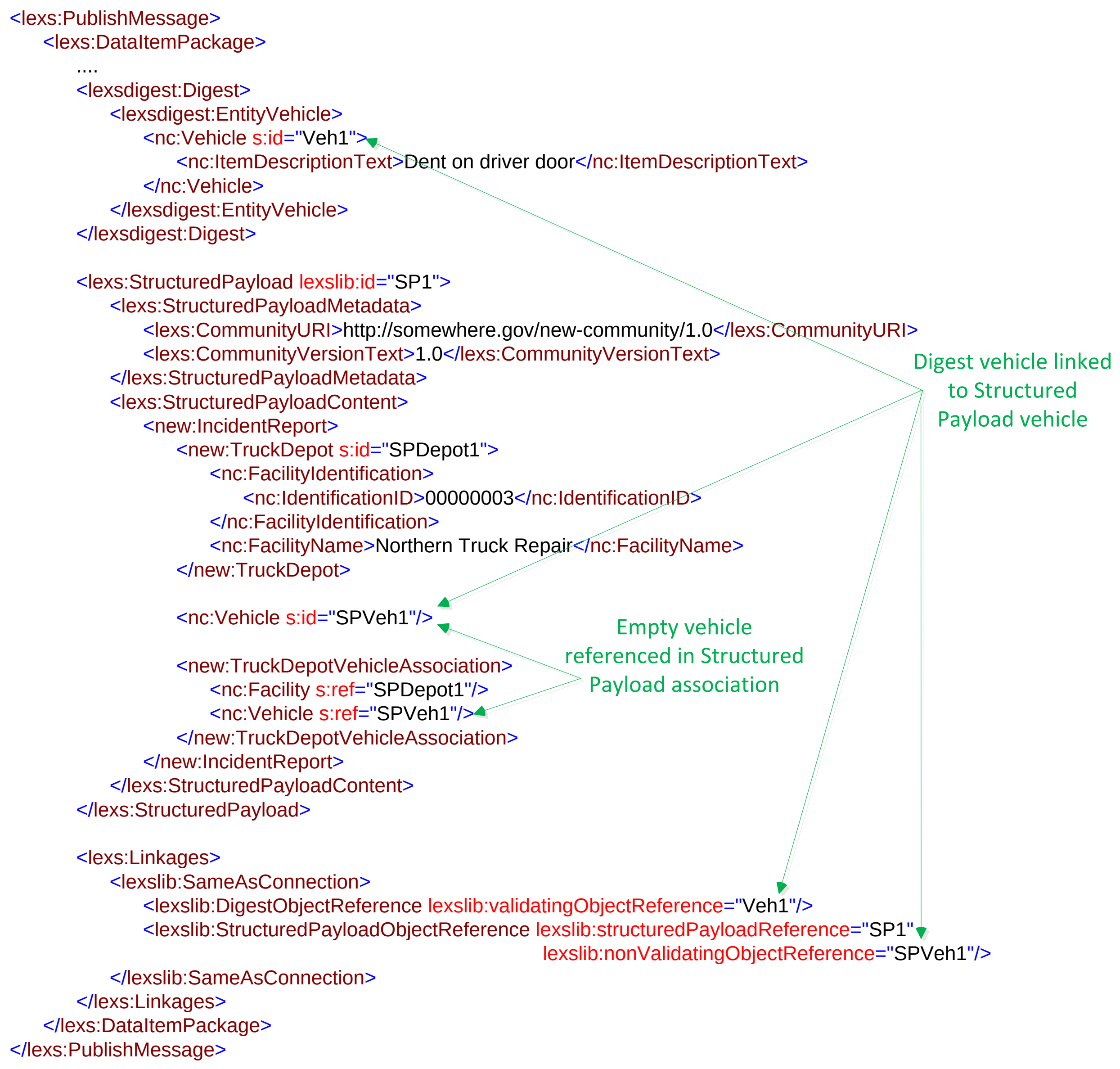
For example, a previous example described a truck repair depot where all the information resides in the Structured Payload. What if an exchange needed to be able to link that depot to specific vehicles that were repaired? The truck repair depot information would only be found in the Structured Payload, and therefore an association linking the depot to vehicles would have to reside in the Structured Payload. If Vehicle is extended in the Structured Payload, the association would be able to refer to a Vehicle element in the Structured Payload, although in some cases the instance may require an empty Vehicle element in the Structured Payload instance so the association has something to link to. If the exchange does not require any additional content in the Structured Payload for Vehicle, the Structured Payload must have a "placeholder" for Vehicle to be defined for the association to reference.

The schema and instance examples below show the definition of an association and placeholder Vehicle to work in conjunction with the truck repair depot defined earlier. The example uses a NIEM subset that includes an empty VehicleType, a Vehicle element, and a Facility element. Since the Truck Depot is of FacilityType, the NIEM Facility can be used for the Truck Depot.

```
<xsd:complexType name="TruckDepotVehicleAssociationType">
  <xsd:annotation>
    <xsd:documentation>A data type for a relationship between a truck depot and a repaired
                        vehicle.</xsd:documentation>
    <xsd:appinfo>
      <i:Base i:namespace="http://release.niem.gov/niem/niem-core/3.0" i:name="AssociationType"/>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="nc:AssociationType">
      <xsd:sequence>
        <xsd:element ref="nc:Facility"/>
        <xsd:element ref="nc:Vehicle"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="TruckDepotVehicleAssociation" type="new:TruckDepotVehicleAssociationType">
  <xsd:annotation>
    <xsd:documentation>A relationship between a truck depot and a repaired vehicle.</xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

Figure 22 - Structured Payload Association



```
<lexs:PublishMessage>
    <lexs:DataItemPackage>
        ....
        <lexsdigest:Digest>
            <lexsdigest:EntityVehicle>
                <nc:Vehicle s:id="Veh1">
                    <nc:ItemDescriptionText>Dent on driver door</nc:ItemDescriptionText>
                </nc:Vehicle>
            </lexsdigest:EntityVehicle>
        </lexsdigest:Digest>

        <lexs:StructuredPayload lexslib:id="SP1">
            <lexs:StructuredPayloadMetadata>
                <lexs:CommunityURI>http://somewhere.gov/new-community/1.0</lexs:CommunityURI>
                <lexs:CommunityVersionText>1.0</lexs:CommunityVersionText>
            </lexs:StructuredPayloadMetadata>
            <lexs:StructuredPayloadContent>
                <new:IncidentReport>
                    <new:TruckDepot s:id="SPDepot1">
                        <nc:FacilityIdentification>
                            <nc:IdentificationID>00000003</nc:IdentificationID>
                        </nc:FacilityIdentification>
                        <nc:FacilityName>Northern Truck Repair</nc:FacilityName>
                    </new:TruckDepot>

                    <nc:Vehicle s:id="SPVeh1"/>

                    <new:TruckDepotVehicleAssociation>
                        <nc:Facility s:ref="SPDepot1"/>
                        <nc:Vehicle s:ref="SPVeh1"/>
                    </new:TruckDepotVehicleAssociation>
                </new:IncidentReport>
            </lexs:StructuredPayloadContent>
        </lexs:StructuredPayload>

        <lexs:Linkages>
            <lexslib:SameAsConnection>
                <lexslib:DigestObjectReference lexslib:validatingObjectReference="Veh1"/>
                <lexslib:StructuredPayloadObjectReference lexslib:structuredPayloadReference="SP1"
                                                          lexslib:nonValidatingObjectReference="SPVeh1"/>
            </lexslib:SameAsConnection>
        </lexs:Linkages>
    </lexs:DataItemPackage>
</lexs:PublishMessage>
```

Digest vehicle linked to Structured Payload vehicle
Empty vehicle referenced in Structured Payload association

Figure 23 - Structured Payload Association Referencing a Placeholder Element in XML

The instance above includes the empty Vehicle defined in the NIEM subset used by the Structured Payload, linking it to the Digest Vehicle via the Linkages section. Therefore, a consumer would know that the vehicle being referenced in the association is the same vehicle contained in the Digest.

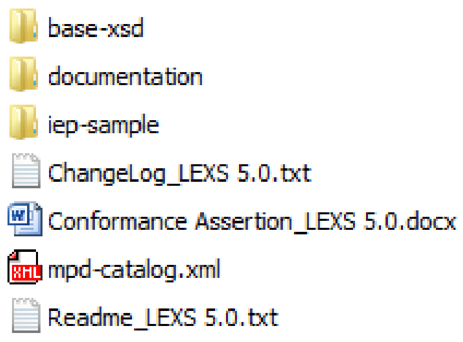
6 Directory Structures and File Naming Conventions

Consistent organization and naming of directories and files facilitates understanding by implementers. While LEXS does not mandate directory structure or directory/file names, the guidance in this section can simplify the development of LEXS-based IEPDs, and provide a consistent approach for LEXS implementers.

6.1.1 Directory Structures

A well-organized hierarchical directory structure is key to providing the proper segregation of LEXS schemas from Structured Payload schemas. Neither LEXS nor the NIEM MPD mandate a directory structure; however this section provides an approach that aligns with the guidance in the NIEM MPD and the layout of the LEXS schemas. The LEXS directory structure is based on the NIEM MPD guidance, and was developed to provide a consistent organization for an IEPD. Structured Payload implementers may collapse or rearrange the Structured Payload portion of the hierarchy; however, the directory structure for the LEXS schemas must be maintained exactly as it exists in the LEXS distribution.

Based on the MPD guidelines, LEXS provides three top-level directories in the IEPD directory structure: *base-xsd* for schemas, *iep-sample* for XML sample instances, and a *documentation* directory. Per the MPD, changelog, MPD catalog, readme, and conformance assertion files must be in the root directory of the IEPD; additional details for filenames are provided later in this section. The root directory for the LEXS specification is shown below.

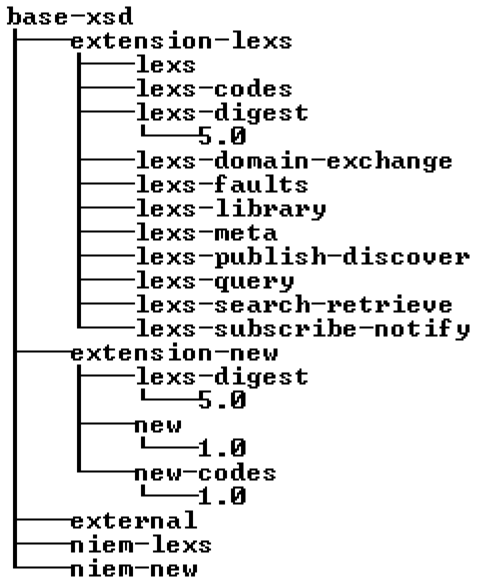


base-xsd
documentation
iep-sample
ChangeLog_LEXS 5.0.txt
Conformance Assertion_LEXS 5.0.docx
mpd-catalog.xml
Readme_LEXS 5.0.txt

Figure 24 - LEXS Root Directory Contents

The organization of the directories for documentation and XML samples are not addressed in the MPD and can be organized as desired.

As discussed previously, a copy of the LEXS schemas must be included. Per the MPD guidelines, the *base-xsd* directory includes subdirectories for extensions and NIEM subsets. LEXS also includes an external directory for non-NIEM and non-LEXS schemas. Since a typical LEXS-based IEPD will include LEXS extension schemas as well as IEPD extension schemas, and since LEXS and IEPD schemas must be segregated, the *base-xsd* directory shown below includes an *extension-new* directory specifically for the IEPD's Structured Payload extension schemas. (The *extension-lexs* directory structure and naming is defined by LEXS and must not be rearranged or renamed.) Similarly, there are separate *niem-lexs* and *niem-new* directories specifically for the NIEM subsets of LEXS and the IEPD, respectively. Note that IEPD should name their extension and NIEM subset directories appropriately for the IEPD, rather than using "-new" as shown in these examples. In order to simplify the illustration, the directory structure shown below includes only the top-level schemas with the exception of the LEXS Digest and IEPD extension directories which also show all subdirectory names.



```
base-xsd
├───extension-lexs
│   ├───lexs
│   ├───lexs-codes
│   ├───lexs-digest
│   │   └───5.0
│   ├───lexs-domain-exchange
│   ├───lexs-faults
│   ├───lexs-library
│   ├───lexs-meta
│   ├───lexs-publish-discover
│   ├───lexs-query
│   ├───lexs-search-retrieve
│   └───lexs-subscribe-notify
├───extension-new
│   ├───lexs-digest
│   │   └───5.0
│   ├───new
│   │   └───1.0
│   └───new-codes
│       └───1.0
├───external
├───niem-lexs
└───niem-new
```

Figure 25 - Sample Information Exchange Directory Structure

For the example, the Structured Payload community name used is “new”, with a version number of 1.0. Note that real Structured Payloads should use real names rather than names like “new”; for example, the N-DEx Incident/Arrest exchange uses “ndexia”. Note that the *extension-new* subdirectory includes a *lexs-digest* entry as does the *extension-lexs* subdirectory. The one under *extension-lexs* is the original that comes with the LEXS distribution, while the one under *extension-new* is a subset copy of the original.

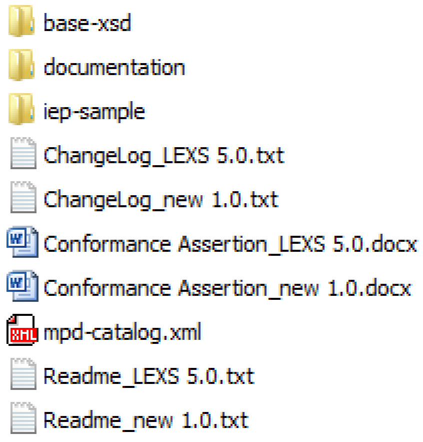
LEXS utilizes a separate directory for the schema file that contains the enumerated code types defined by LEXS, specifically the *base-xsd/extension-lexs/lexs-codes* directory. While Structured Payloads are not required to follow this convention of segregating the code list schema file from other schema files, this segregation can be very beneficial since NIEM provides a tool that can generate a schema file from a spreadsheet containing code lists. Therefore, if the code list schema file is segregated, the code list schema file can be regenerated when the spreadsheet is updated without impacting any other schema files.

Note that if an exchange is being developed to add an additional Structured Payload to work with an existing Structured Payload, the schemas for the new Structured Payload must also be segregated from the schemas for the existing Structured Payload.

6.1.2 File Naming Conventions

The NIEM MPD does not specify file naming conventions except for one. A NIEM IEPD must include an MPD catalog in the IEPD’s root directory with the name *mpd-catalog.xml*. The MPD also specifies that an IEPD must include changelog, readme, and conformance assertion files in the root directory of the IEPD, but does not specify the filenames.

LEXS provides the MPD catalog, changelog, readme, and conformance assertion files in the root directory. However, a LEXS-based IEPD must provide IEPD-specific versions of those mandated files since those files are required to document the IEPD itself, not the LEXS specification upon which the IEPD is based. IEPD developers can retain the LEXS versions of the changelog, readme and conformance assertion if desired, but must also include IEPD specific versions. The LEXS files include the notation "_LEXS 5.0" in the filenames, so if they are retained the IEPD could use a notation similar to "_new 1.0" in the IEPD-specific filenames. The diagram below shows an example IEPD root directory that retains the existing LEXS documentation files plus the IEPD-specific versions.



```
base-xsd
documentation
iep-sample
ChangeLog_LEXS 5.0.txt
ChangeLog_new 1.0.txt
Conformance Assertion_LEXS 5.0.docx
Conformance Assertion_new 1.0.docx
mpd-catalog.xml
Readme_LEXS 5.0.txt
Readme_new 1.0.txt
```

Figure 26 - Sample IEPD Root Directory with IEPD-specific MPD Artifacts

Additionally, the LEXS specification also includes a number of files in the *documentation* subdirectory that are specific to LEXS and that also utilize the "_LEXS 5.0" notation in the filenames. None of the files in the *documentation* directory are required by the MPD, but may be useful to IEPD implementers. IEPD developers may choose to retain or delete any of these LEXS artifacts. Similar to the required root directory artifacts, IEPD developers that choose to retain any of the LEXS provided artifacts may add IEPD-specific artifacts to the *documentation* directory as well, but may want to consider utilizing notation similar to "_new 1.0" in the IEPD-specific filenames.

7 Resources

7.1 Conformance Testing Assistant (ConTesA)

The Conformance Testing Assistant (ConTesA) is a tool that tests a data instance for conformance to business rules and provides visualization of XML instances. ConTesA performs LEXS schema validation and tests for business rules that cannot be represented in XML Schema. Information exchanges can define additional business rules which can be added to ConTesA's rule base. ConTesA is available online, as a desktop tool, and as a WAR file that can be deployed in a developer's local environment. The online version, which also provides a link to ConTesA Desktop and contact information for requesting the WAR file, is available at http://contesa.ittl.gtri.org.

7.2 LEXS Community Web Site

The LEXS community web site is available for use by managers and implementers alike. It includes a download section where the LEXS specifications and various documentation artifacts can be accessed. It also includes a community forum where users and implementers can post questions and provide answers to others. The site also has links to other sites, such as ConTesA. The site is located at http://lexsdev.org.

7.3 LEXS Specification Guide

The LEXS Specification Guide, which is included as part of the LEXS 5.0 distribution zip file, provides detailed documentation on the LEXS specification. The document includes a high-level overview of the specifications, as well as details on LEXS PD, SR, SN, and DE operations. The guide also documents LEXS business rules that must be adhered to for an information exchange to be LEXS conformant.

7.4 NIEM.gov

This is the web site for the NIEM program that provides access to the NIEM specifications, documentation, news, training information, implementer tools, etc. Current documentation for the NDR and MPD mentioned above is available on the web site as well. The site is located at http://niem.gov.

7.5 Subset Generator Tool (SSGT)

The Subset Generator Tool (SSGT) provides a means for creating a subset of the full NIEM schemas. A user selected properties and types required for a data exchange, and the tool generates a conformant schema subset of the full NIEM schema set. All dependencies are automatically added to ensure the resulting schema subset is valid. The user requirements can be saved and/or reloaded in a NIEM wantlist file, allowing for the subset to be modified and regenerated. It is available at http://tools.niem.gov/niemtools/ssgt/index.iepd.

7.6 NIEM Codelist Generator

The NIEM codelist generator tool builds a NIEM conformant code list schema from an Excel or CSV file. Multiple code lists can be included in a single code list schema. The tool is located at http://tools.niem.gov/niemtools/codelist/index.iepd.

7.7 NIEM IEPD Conformance Testing Assistant (NIEM ConTesA)

The NIEM Conformance Testing Assistant is a version of ConTesA geared towards validation testing of IEPDs, and performs automated testing of IEPDs against the NIEM NDR and MPD specifications. At the time this document was developed, the NIEM 3.x version of NIEM ConTesA was in beta. The tool is expected to be available at https://tools.niem.gov/contesa.