Logical Entity eXchange Specifications 4.0 (LEXS)

Structured Payloads Guide

Revision 1

Change History

Rev	Date	Author	Description of Revision
1	9/27/2011	Jack Wallace	Initial version

Acknowledgments

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

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

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

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

Table of Contents

1 Introduction

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

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

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

This document covers aspects of the Logical Entity eXchange Specifications (LEXS, pronounced "lex") which leverages and reuses work from both LEISP and NIEM. It provides additional details regarding LEXS Structured Payloads beyond what is available in the LEXS User Guide. This document is for LEXS 4.0.

1.1 Logical Entity eXchange Specifications

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

LEXS is intended to address two aspects of information sharing:

- define and consistently describe units of information to be shared
- define interfaces and protocols to provide (publish) information, search and retrieve, and subscribe to and receive notification regarding information

LEXS provides an extensible framework for consistent packaging of the information, with specific placement and markings for various elements of the shared information. The LEXS

specification shields both data sources and data recipients from the complexity of multiple interfaces and allows for the multipurpose use of information: for example, a data item created by a source can be consumed by multiple recipients that should be able to understand as much or as little of the data as necessary.

LEXS 4.0 specifications can be obtained from the LEXS community web site. Any updates to this document will also be posted at the same site.

1.2 Audience

This document is intended for the reader who needs to build a LEXS 4.0 Structured Payload schema or needs to leverage an existing information exchange schema utilizing LEXS constructs.

This document does not provide extensive background or detailed technical information on the LEXS specifications, which is provided in the LEXS 4.0 User Guide. This document assumes the reader is familiar with basic LEXS concepts, or has at least reviewed the LEXS 4.0 User Guide or Quick Start 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 LEXS Background Information

2.1 Terminology

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 prison system, a logical unit of information is an inmate record that may just have information about a person. The data Item concept provides a single, generic container that can be used to encapsulate different types of data needed by various communities. The data Item can be thought of as a collection of structured entities, attributes of these entities, relationships between these entities and unstructured textual information.

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

2.2 Digest

The concept of the **Digest** is central to the power of LEXS as a framework for information sharing. The contents of the Digest can be understood by anyone who understands LEXS. 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 do not have to develop large applications for each exchange, but rather build one that handles the basics and then additional smaller modules in order to produce or consume more complex exchanges. The 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.

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 4.0 defines the following high level entities:

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

The objective of the Digest is to present the most common characteristics of these real-world objects that can be supported by most data sources or data consumers. The Digest also contains associations and roles related to these real-world, or "base", objects. Digest-level data objects may be further augmented or described with additional details in the Structured Payload.

2.3 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. Ideally these schemas are built to be aware of the LEXS Digest, that is, Structured Payload entities build on and link to LEXS Digest entities. These are referred to as LEXS-aware Structured Payloads. If an existing schema is being used that was not built to leverage LEXS, the Structured Payload is referred to as a non-LEXS-aware Structured Payload. LEXS-aware Structured Payloads can build upon the contents of the Digest, or upon the contents of another Structured Payload.

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.

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.

Since a Structured Payload is based on a separate schema, Structured Payload instances must be valid as standalone instances. The data content of a LEXS-aware Structured Payload is not expected to be a "complete" instance by itself since the content of the Structured Payload does not make logical sense by itself without the content of the Digest; however the Structured Payload instance must still be valid against the Structured Payload schema. Keep in mind that even though LEXS-aware Structured Payload instances must be valid by themselves, they are intended to be part of a LEXS message, not separate instances that exist independently of the LEXS message.

In general, non-LEXS-aware Structured Payloads are intended for communities that already have a schema containing all the data elements they want to exchange, but that want to leverage LEXS structures and concepts, or that want to establish a level of data compatibility with other communities that are or will be using LEXS. Instances based on a non-LEXS-aware Structured Payload schema are expected to be "complete" instances by themselves, where all information is in the Structured Payload and the Digest just provides a summary of the information.

LEXS-aware Structured Payloads are preferred since they don't include the potentially significant duplication of data between the Digest and Structured Payload. In addition, LEXS-aware Structured Payloads can be "stacked" so that a community can create a Structured Payload that builds upon some other community's Structured Payload. This allows Structured Payloads to be more compact and modular than the more monolithic non-LEXS-aware Structured Payload.

3 Structured Payload Concepts

This chapter is intended to provide details on Structured Payloads in general and is not specific to LEXS-aware versus non-LEXS-aware 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 issues. Portions of this document refer to NIEM Information Exchange Package Documentation (IEPD) 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.

3.1 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. LEXS-aware Structured Payload schemas utilize LEXS constructs so that the Structured Payload builds upon the information in the Digest, while non-LEXS-aware Structured Payloads do not. Regardless of whether the Structured Payload is LEXS-aware or not, the Structured Payload schemas need to 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 4.0 includes four top-level exchange schemas plus a number of supporting schemas. The four top-level exchange schemas are for publish (ulex-publish-discover.xsd), search and retrieve (ulex-search-retrieve.xsd), subscription and notification (ulex-subscribe-notify.xsd), and domain exchange (ulex-domain-exchange.xsd). What is appropriate depends on the use case. For example, those who are implementing a publish capability require the ulex-publish-discover.xsd, but the other three top-level schemas can be removed. Note that the LEXS 4.0 specification is available in three different distributions: publish only, publish plus search and retrieve, and all four.

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 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 builds upon 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 a reference to an Email address. The LEXS Digest defines one for use in the Digest, so rather than redefining the reference in the Structured Payload, the payload schema set can include a subset copy of the Digest schema that includes just the EmailAddressReference 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.

3.2 Directory Structures

A well-organized hierarchical directory structure is key to providing the proper segregation of LEXS schemas from Structured Payload schemas. There is no single mandated directory

structure that must be followed by LEXS implementations; however this section provides an approach that has been utilized successfully. The basic directory structure is what is important; the exact naming conventions for directories and files can be changed as desired. The LEXS directory structure is based on NIEM conventions, and the Structured Payload subdirectories follow the same conventions. Structured Payload implementers may collapse their portion of the hierarchy; however, the directory structure for the LEXS schemas must be maintained exactly as it exists in the LEXS distribution.

As discussed previously, a copy of the LEXS schemas must be included. The directory structure shown below include only the top-level publish schemas. Beneath the parent *xsd* directory are separate directories for LEXS and for the Structured Payload. In order to simplify the illustration, many subdirectories have not been expanded.

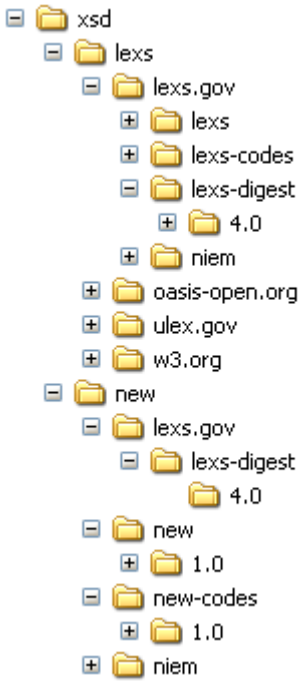

Figure 1 - Sample Information Exchange Directory Structure

For the example, the Structured Payload community name used is “new”, with a version number of 1.0. 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 *new* subdirectory includes a *lexs-digest* entry as does the *lexs* subdirectory. The one under *lexs* is the original that comes with the LEXS distribution, while the one under *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 *xsd/lexs/lexs.gov/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.

3.3 Relationships among Digest and Structured Payloads

Data in the Digest and Structured Payload is expected to be linked when both represent the same real world object, regardless of whether the Structured Payload is LEXS-aware or not. Data in a Structured Payload may also be linked to data in another Structured Payload when they represent the same real world object.

3.3.1 Structured Payload Building upon the Digest

In the case of LEXS-aware Structured Payloads, 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 should 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 TangibleItem object.

For example, an exchange may need to represent Jewelry with a jewelry description and a jewelry category code. While the LEXS Digest includes TangibleItem, it does not include Jewelry. However, in NIEM, JewelryType extends TangibleItemType, 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 TangibleItem. The TangibleItem 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.

For example, an exchange may require representation of a Tool object as a role of a tangible item. The LEXS Digest does not include any roles that would make sense as the foundation of a Tool role. Therefore, the Structured Payload would include the Tool data element.

Duplication of Data between Digest and Structured Payload
As alluded to above, Structured Payload schemas should not duplicate Digest data elements. It is important to avoid data duplication between the Digest and the Structured Payloads in order to provide consistent representations 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 item value as part of tangible 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 tangible item in the Digest would be linked to the tangible item in the Structured Payload through the SameAsConnection element.

3.3.2 Structured Payloads That Do Not Build upon the Digest

In the case of non-LEXS-aware Structured Payloads, Structured Payload objects do not build upon Digest objects but are complete by themselves. However, Digest objects are used to provide a set of information that can be understood by all LEXS consumers. 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.

3.3.3 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. Objects are linked between Structured Payloads in XML instances through the use of a SameAsConnection element in the Linkages block.

3.4 Structured Payloads and NIEM

Structured Payloads do not have 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 expected to use NIEM tools, such as the Subset Generation Tool.

As noted previously, the contents of Structured Payloads should build upon the contents of the Digest when possible, rather than duplicating content.

3.5 Extending NIEM

NIEM-based Structured Payloads may require data elements that are not available in NIEM. For LEXS-aware Structured Payloads the required data elements may not be available in LEXS Digest extensions to NIEM either. When these data elements are not available, the Structured Payload must extend NIEM in order to add the necessary data elements. This document provides an overview along with examples illustrating the creation of Structured Payload objects. More detailed information is available in the NIEM document “Techniques for Building and Extending NIEM”. See section 6.10 for more information on this document.

3.6 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. Details for the contents of the Structured Payload metadata block are provided in the LEXS 4.0 User Guide.

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. For those creating instances based on the schemas, mapping must be performed against the separate schemas. This doesn't mean that it is impossible to build, consume, or validate instances in a single pass, just that most tools currently require these "separate" instances be treated separately in a two-pass process.

3.7 IEPD Implications

NIEM IEPDs require an exchange schema (sometimes referred to as a document schema) which defines the top level element(s) in an exchange. Communities that develop exchanges based on LEXS do not need to create an exchange schema since LEXS defines the top level exchange elements, such as doPublish that is part of the *ulex-publish-discover.xsd* schema.

Details on available LEXS top-level exchange elements, or LEXS operations, are provided in the LEXS 4.0 User Guide.

4 Building LEXS-aware Structured Payloads

Communities that are in a position to build a LEXS-aware Structured Payload must address a number of issues. The community needs to determine what data is required in the Structured Payload versus 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-used the existing exchange and build a 2nd Structured Payload to fill in the gaps rather than building a Structured Payload that overlaps the existing exchange.

If a community determines that a new Structured Payload is needed, they must determine what objects in the Structured Payload build upon what objects in the Digest, what objects can exist solely in the Digest, and what objects will exist solely in the Structured Payload. If the community is building upon an existing exchange, the process is more a matter of evaluating the complete exchange rather than just evaluating the Digest.

If any of the data necessary for the new Structured Payload exists in NIEM, a NIEM subset schema must be created along with extension schemas. The extension schemas may include a code list schema if the Structured Payload needs to include additional code lists beyond what is available in NIEM. Finally, all this needs to be incorporated into a directory structure that segregates the Structured Payload schemas from the LEXS schemas, as discussed previously.

4.1 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 TangibleItem. However, the community may require details about Jewelry. Since Jewelry is a tangible thing, the community can use the Digest's TangibleItem 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.

Mapping 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 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. LEXS provides the Component Mapping Workbook, or CMW, which is a Microsoft Excel spreadsheet that includes all high level data objects and associations along with their complete contents. This can be used as the starting point for developing a mapping spreadsheet that includes all data required in the information exchange. Information can be added in a new column to indicate the mapping of Digest elements to the community's data model, and new lines can be added for high level objects or individual elements that do not exist in LEXS.

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 EnforcementOfficial and 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 in turn leads to data integrity problems later.

When the mapping is done, work can start on creating the Structured Payload schemas. Assuming the Structured Payload will be based on NIEM, this means that 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.

4.2 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. Note that for a LEXS-aware Structured Payload, data elements that exist in the Digest should not be duplicated in the Structured Payload subset schema. There may be cases where this cannot be avoided, perhaps because a Structured Payload needs more relaxed cardinality on a Digest data element or needs a Digest data element as part of a different type. But duplication should be avoided if possible.

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.

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. 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, for example for the most recent vehicle models, 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 empty subset of a type is needed. This 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 Person, and the Structured Payload may need to include Person as well. But if there are no person data elements available in NIEM for the necessary concept, the person elements will have to be defined in an extension schema. In this case an empty PersonType should be included to form the basis of the extended PersonType. The example below shows an empty subset of PersonNameType that has been created to provide the foundation for a Structured Payload PersonNameType that includes content from the Justice domain plus a community augmentation that adds hacker name.

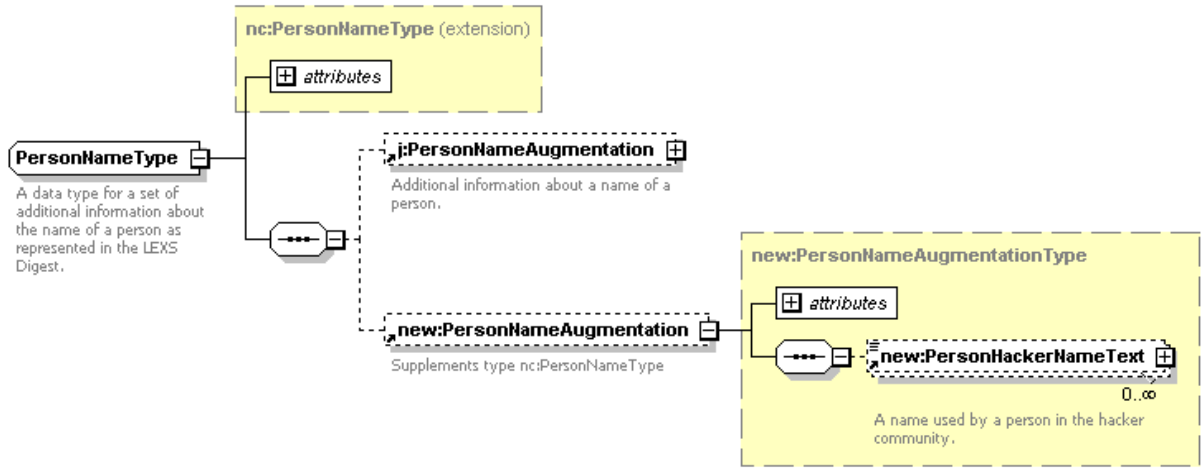

Figure 2 - Extending an Empty Subset of a NIEM Type

4.3 NIEM Constraint Schemas

NIEM defines all data elements as optional and multiply occurring. As noted previously, the NIEM SSGT provides for the creation of subset schemas that include more specific cardinality for any data element. However, there may be cases where an information exchange needs to define more specific constraints, for example that a particular data element may only contain letters, or can only be of a certain length. In these cases, a constraint schema may be utilized.

Depending on the desired constraints, their inclusion in schema may make it more difficult for implementers to understand and utilize the schemas. Therefore, creators of Structured Payload schemas need to balance the added complexity of the schemas against the benefit of having the constraints enforced in schema. It may be more practical to define business rules that are not enforced by schema but that are documented in the IEPD. Business rules can be enforced through tools such as the Conformance Testing Assistant (ConTesA) so that XML instance builders can ensure their instance meets such business rules.

4.4 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 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 xsd: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.

If the Structured Payload schemas include NIEM subset schemas, the code list schema can be manually edited to change the import statements to reference the NIEM schema files that are already part of the subset rather than including another copy of the NIEM supporting schemas.

4.5 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 ComplexObjectType 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 ComplexObjectType. 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 don't 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 MilitaryVehicle element that includes content not found in 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 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.

4.5.1 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. If a code list schema has been created, that must also be imported into the extension schema file. The extension schema must include a documentation element. If the extension schema properly adheres to the NIEM NDR, the NIEM conformance indicator must be set to true, and the schema version attribute must be provided.

So the first step in creating the extension schema is to create a template into which other things will be added. The figure below shows a possible template for an extension schema. Depending on the size and complexity of the extension schema, it may be beneficial to both the developers and users of the schema to segregate the extension schema into sections. The example below shows a shell with separate sections for types, high level objects (entities), associations, augmentation elements, and all other elements.

```
<xsd:schema
   xmlns:i="http://niem.gov/niem/appinfo/2.0"
   xmlns:nc="http://niem.gov/niem/niem-core/2.0"
   xmlns:new="http://somewhere.gov/new-community"
   xmlns:newcodes="http://somewhere.gov/new-community/codes"
   xmlns:niem-xsd="http://niem.gov/niem/proxy/xsd/2.0"
   xmlns:s="http://niem.gov/niem/structures/2.0"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://somewhere.gov/new-community"
   elementFormDefault="qualified" attributeFormDefault="unqualified" version="1.0">

   <xsd:import namespace="http://niem.gov/niem/structures/2.0" schemaLocation="../../niem/structures/2.0/structures.xsd"/>
   <xsd:import namespace="http://niem.gov/niem/appinfo/2.0" schemaLocation="../../niem/appinfo/2.0/appinfo.xsd"/>
   <xsd:import namespace="http://niem.gov/niem/niem-core/2.0" schemaLocation="../../niem/niem-core/2.0/niem-core.xsd"/>
   <xsd:import namespace="http://niem.gov/niem/proxy/xsd/2.0" schemaLocation="../../niem/proxy/xsd/2.0/xsd.xsd"/>
   <xsd:import namespace="http://somewhere.gov/new-community/codes"
               schemaLocation="../../new-codes/1.0/new-codes.xsd"/>

   <xsd:annotation>
      <xsd:documentation>New community structures</xsd:documentation>
      <xsd:appinfo>
         <i:ConformantIndicator>true</i:ConformantIndicator>
      </xsd:appinfo>
   </xsd:annotation>

   <!-- Types -->

   <!-- Entity Elements -->

   <!-- Associations -->

   <!-- Augmentation Elements -->

   <!-- Other Elements -->
   ……
</xsd:schema>
```

Figure 4 - Sample Extension Schema Template

4.5.2 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 "IncidentReport" as the Structured Payload root element.

```
<ulexpd:doPublish>
    <ulex:PublishMessageContainer>
        <ulex:PublishMessage>
            <ulex:PDMessageMetadata>
            ...
            <ulex:DataSubmitterMetadata>
            ...
            <ulex:DataItemPackage>
                <ulex:PackageMetadata>
                ...
                <lexs:Digest>
                    <lexsdigest:EntityActivity>
                    ...
                    <lexsdigest:EntityPerson>
                    ...
                    <lexsdigest:Associations>
                    ...
                </lexs:Digest>
                <!--====================== Data from another community ======================-->
                <ulex:StructuredPayload ulexlib:id="SP1">
                    <ulex:StructuredPayloadMetadata>
                        <ulex:CommunityURI>http://somewhere.gov/new-community</ulex:CommunityURI>
                        <ulex:CommunityDescriptionText>New Community</ulex:CommunityDescriptionText>
                        <ulex:CommunityVersionText>1.0</ulex:CommunityVersionText>
                    </ulex:StructuredPayloadMetadata>
                    <ulex:StructuredPayloadContent>
                        <new:IncidentReport>
                            <new:Person s:id="SPPerson1">
                                <new:PersonAlternateName>
                                    <new:PersonNameAugmentation>
                                        <new:PersonHackerNameText>CyberDude</new:PersonHackerNameText>
                                    </new:PersonNameAugmentation>
                                </new:PersonAlternateName>
                            </new:Person>
                        </new:IncidentReport>
                    </ulex:StructuredPayloadContent>
                </ulex:StructuredPayload>
                <ulex:Linkages>
                ...
            </ulex:DataItemPackage>
        </ulex:PublishMessage>
    </ulex:PublishMessageContainer>
</ulexpd: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 "IncidentReport" 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. Some may

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.

4.5.3 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. The examples below illustrate these two approaches. For these examples, the Structured Payload includes the objects Computer, Incident, Jewelry, Person, Subject (a Role), and Tool (a role), plus the associations DomesticPartnershipAssociation, FinancialSupporterAssociation, and OrganizationAssociation. In order to simplify the examples, documentation elements have been removed.

The first example below shows two abstract elements, Entity and Association, which make up the contents of the ReportType. Those abstract elements can be substituted with the elements shown under the Entity and Associations heading, respectively. The advantage to this approach is that in an instance, the Entity objects can appears in any order, so for example, a sequence consisting of Person, followed by Computer, followed by Person is perfectly legal.

```
<!-- ================================================================== -->
<!—Root Element Type -->
<!-- ================================================================== -->
<xsd:complexType name="ReportType">
    …
    <xsd:complexContent>
        <xsd:extension base="s:ComplexObjectType">
            <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>

<!-- ================================================================== -->
<!—Root Elements -->
<!-- ================================================================== -->
<xsd:element name="ArrestReport" type="new:ReportType" substitutionGroup="new:Report" nillable="true"/>
<xsd:element name="IncidentReport" type="new:ReportType" substitutionGroup="new:Report" nillable="true"/>

<!-- ================================================================== -->
<!-- Abstract Elements -->
<!-- ================================================================== -->
<xsd:element name="Association" abstract="true"/>
<xsd:element name="Entity" abstract="true"/>

<!-- ================================================================== -->
<!-- Entity Elements -->
<!-- ================================================================== -->
<!-- These elements are all substitutable for the Entity object, so that all high level objects that may
appear in a report are grouped together without having to be in a specific order. -->
<!-- ================================================================== -->
<xsd:element name="Computer" type="new:ComputerType" substitutionGroup="new:Entity" nillable="true"/>
<xsd:element name="Incident" type="new:IncidentType" substitutionGroup="new:Entity" nillable="true"/>
<xsd:element name="Jewelry" type="new:JewelryType" substitutionGroup="new:Entity" nillable="true"/>
<xsd:element name="Person" type="new:PersonType" substitutionGroup="new:Entity" nillable="true"/>
<xsd:element name="Subject" type="new:SubjectType" substitutionGroup="new:Entity" nillable="true"/>
<xsd:element name="Tool" type="new:ToolType" substitutionGroup="new:Entity" nillable="true"/>

<!-- ================================================================== -->
<!-- Association Elements-->
<!-- ================================================================== -->
<!-- These elements are all substitutable for the Association object, so that all associations that may
appear in a report are grouped together without having to be in a specific order. -->
<!-- ================================================================== -->
<xsd:element name="FinancialSupporterAssociation" type="new:FinancialSupporterAssociationType"
substitutionGroup="new:Association"/>
<xsd:element name="DomesticPartnershipAssociation" type="new:PersonUnionAssociationType"
substitutionGroup="new:Association"/>
<xsd:element name="OrganizationAssociation" type="new:OrganizationAssociationType"
substitutionGroup="new:Association"/>
```

Figure 6 - Root Element Using Abstract Elements in Schema

The next example shows the root element type defined with all child elements in a concrete sequence. XML instances based on this arrangement must follow this sequence exactly, meaning for example that all Computer elements must come before any Person element.

```
<!-- ====================================================================== -->
<!—Root Element Type -->
<!-- ====================================================================== -->
<xsd:complexType name="ReportType">
    …
    <xsd:complexContent>
        <xsd:extension base="s:ComplexObjectType">
            <xsd:sequence>
                <xsd:element ref="new: Computer " minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="new: Incident " minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="new: Jewelry " minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="new: Person " minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="new: Subject " minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="new: Tool " minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="new: FinancialSupporterAssociation " minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="new: DomesticPartnershipAssociation " minOccurs="0" maxOccurs="unbounded"/>
                <xsd:element ref="new: OrganizationAssociation " minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<!-- ====================================================================== -->
<!—Root Elements -->
<!-- ====================================================================== -->
<xsd:element name="ArrestReport" type="new:ReportType" substitutionGroup="new:Report" nillable="true"/>
<xsd:element name="IncidentReport" type="new:ReportType" substitutionGroup="new:Report" nillable="true"/>

<!-- ====================================================================== -->
<!-- Entity Elements -->
<!-- ====================================================================== -->
<xsd:element name="Computer" type="new:ComputerType" nillable="true"/>
<xsd:element name="Incident" type="new:IncidentType" nillable="true"/>
<xsd:element name="Jewelry" type="new:JewelryType" nillable="true"/>
<xsd:element name="Person" type="new:PersonType" nillable="true"/>
<xsd:element name="Subject" type="new:SubjectType" nillable="true"/>
<xsd:element name="Tool" type="new:ToolType" nillable="true"/>

<!-- ====================================================================== -->
<!-- Association Elements-->
<!-- ====================================================================== -->
<xsd:element name="FinancialSupporterAssociation" type="new:FinancialSupporterAssociationType"/>
<xsd:element name="DomesticPartnershipAssociation" type="new:PersonUnionAssociationType"/>
<xsd:element name="OrganizationAssociation" type="new:OrganizationAssociationType"/>
```

Figure 7 - Root Element Using a Concrete Sequence in Schema

4.5.4 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; so 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 TangibleItem), the basics for building the Structured Payload objects/associations is 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 instance is shown below, where the NIEM subset for the Structured Payload includes the nc:PersonHairStyleText element.

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

    <ulex:StructuredPayload ulexlib:id="SP1">
      <ulex:StructuredPayloadMetadata>
        <ulex:CommunityURI>http://somewhere.gov/new-community</ulex:CommunityURI>
        <ulex:CommunityVersion>1.0</ulex:CommunityVersion>
      </ulex:StructuredPayloadMetadata>
      <ulex:StructuredPayloadContent>
        <new:IncidentReport>
          <new:Person s:id="SPA">
            <nc:PersonHairStyleText>strawberry blond</nc:PersonHairStyleText>
          </new:Person>
        </new:IncidentReport>
      <ulex:StructuredPayloadContent>
    </ulex:StructuredPayload>

    <ulex:Linkages>
      <ulexlib:SameAsConnection>
        <ulexlib:DigestObjectReference ulexlib:validatingObjectReference="A"/>
        <ulexlib:StructuredPayloadObjectReference ulexlib:structuredPayloadReference="SP1"
                    ulexlib:nonValidatingObjectReference="SPA"/>
      </ulexlib:SameAsConnection>
    </ulex:Linkages>
  </ulex:DataItemPackage>
</ulex:PublishMessage>
```

Figure 8 –Sample Structured Payload Object with only NIEM Subset Elements in XML

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, or when a NIEM subset element needs to be put into a different location than defined in NIEM, an AugmentationType can be created as shown below, in this case to add an

element for a person's hair style code. This example also shows the use of a code element specific to the Structured Payload; the definition of the code type is not shown here.

```
<xsd:complexType name="PersonAugmentationType">
  <xsd:annotation>
    <xsd:documentation>A data type that supplements nc:PersonType.</xsd:documentation>
    <xsd:appinfo>
      <i:Base i:namespace="http://niem.gov/niem/structures/2.0" i:name="AugmentationType"/>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="s:AugmentationType">
      <xsd:sequence>
        <xsd:element ref="new:PersonHairStyleCode" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="PersonAugmentation" type="new:PersonAugmentationType" substitutionGroup="s:Augmentation">
  <xsd:annotation>
    <xsd:documentation>Supplements type nc:PersonType</xsd:documentation>
    <xsd:appinfo>
      <i:AppliesTo i:namespace="http://niem.gov/niem/niem-core/2.0" i:name="PersonType"/>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>

<xsd:complexType name="PersonType">
  <xsd:annotation>
    <xsd:documentation>A data type for a human being. Builds upon a LEXS Digest Person.</xsd:documentation>
    <xsd:appinfo>
      <i:Base i:namespace="http://niem.gov/niem/niem-core/2.0" i:name="PersonType"/>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="nc:PersonType">
      <xsd:sequence>
        <xsd:element ref="j:PersonAugmentation" minOccurs="0"/>
        <xsd:element ref="new:PersonAugmentation"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="Person" type="new:PersonType" nillable="true">
  <xsd:annotation>
    <xsd:documentation>A human being.</xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="PersonHairStyleCode" type="newcodes:PersonHairStyleCodeType" nillable="true">
  <xsd:annotation>
    <xsd:documentation>A code that identifies the style of the person's hair.</xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

Figure 9 – Structured Payload Object with Augmentation in Schema

The additional element for the person's hair style code 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 element that is part of the subset schema. The i:Base for the PersonAugmentationType references the namespace for the Structured Payload's extension schema. The i:AppliesTo element is required by NIEM to indicate the data type to which the augmentation adds content.

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 any person-to-person or person-to-organization association in the Digest to indicate that a person or organization provides financial support to a 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 or person-to-organization 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 a person or an organization that provides
        financial support or assistance.</xsd:documentation>
    <xsd:appinfo>
      <i:Base i:namespace="http://niem.gov/niem/niem-core/2.0" i:name="AssociationType"/>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="nc:AssociationType">
      <xsd:sequence>
        <xsd:element ref="new: FinancialSupportAmount" minOccurs="0"/>
        <xsd:element ref="new: FinancialSupportDescriptionText" minOccurs="0"/>
      </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 a person or an organization that provides financial
        support or assistance.</xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="FinancialSupportAmount" type="nc:AmountType" nillable="true">
  <xsd:annotation>
    <xsd:documentation>A monetary amount provided by the financial supporter to the entity receiving
      support.</xsd:documentation>
  </xsd:annotation>
</xsd:element>
<xsd:element name="FinancialSupportDescriptionText" type="nc:TextType" nillable="true">
  <xsd:annotation>
    <xsd:documentation>A description of the financial support or assistance provided by the financial supporter to the
        subject.</xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

Figure 10 - Specialization of Digest Association in Schema

Since the financial supporter association can build upon Digest associations of different types (person-to-person and person-to-organization), the Structured Payload association type extends nc:AssociationType, from which both the nc:PersonAssociationType and nc:PersonOrganizationAssociationType are derived. If the financial supporter association was intended to only build upon a person-to-person association, then the Structured Payload association could extend nc:PersonAssociationType instead. Note that in either case, the Structured Payload subset schema for nc:AssociationType should not contain the date elements, and nc:PersonAssociationType and nc:PersonOrganizationAssociationType should not contain the reference elements either since they already exist in the Digest associations. If the Structured Payload requires associations that link Structured Payload objects to each other and that require the association begin and end dates, the subset for AssociationType would need to include those date elements. In this case, deriving the financial supporter association above from NIEM's ComplexObjectType, which is discussed in the next section, would be a better approach.

The example below illustrates a simple instance with just two people who are friends where one provides financial support to the other.

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

      <lexsdigest:EntityPerson>
        <lexsdigest:Person s:id="B">
          <nc:PersonName>
            <nc:PersonGivenName>Bill</nc:PersonGivenName>
            <nc:PersonSurName>Smith</nc:PersonSurName>
          </nc:PersonName>
        </lexsdigest:Person>
      </lexsdigest:EntityPerson>

      <lexsdigest:Associations>
        <nc:FriendshipAssociation s:id="C">
          <nc:PersonReference s:ref="A"/>
          <nc:PersonReference s:ref="B"/>
        </nc:FriendshipAssociation>
      </lexsdigest:Associations>
    </lexs:Digest>

    <ulex:StructuredPayload ulexlib:id="SP1">
      <ulex:StructuredPayloadMetadata>
        <ulex:CommunityURI>http://somewhere.gov/new-community</ulex:CommunityURI>
        <ulex:CommunityVersion>1.0</ulex:CommunityVersion>
      </ulex:StructuredPayloadMetadata>
      <ulex:StructuredPayloadContent>
        <new:IncidentReport>
          <ndexib:FinancialSupporterAssociation s:id="FSA">
            <ndexib:FinancialSupporterAssociationAugmentation>
              <ndexib:FinancialSupportAmount>500</ndexib:FinancialSupportAmount>
              <ndexib:FinancialSupportDescriptionText>Allowance</ndexib:FinancialSupportDescriptionText>
            </ndexib:FinancialSupporterAssociationAugmentation>
          </ndexib:FinancialSupporterAssociation>
        </new:IncidentReport>
    <ulex:StructuredPayloadContent>
    <ulex:StructuredPayload>

    <ulex:Linkages>
      <ulexlib:SameAsConnection>
        <ulexlib:DigestObjectReference ulexlib:validatingObjectReference="C"/>
        <ulexlib:StructuredPayloadObjectReference ulexlib:structuredPayloadReference="SP1"
                    ulexlib:nonValidatingObjectReference="FSA"/>
      </ulexlib:SameAsConnection>
    </ulex:Linkages>
  </ulex:DataItemPackage>
</ulex:PublishMessage>
```

Figure 11 - Specialization of Digest Association in XML

Object/Association Derived from ComplexObjectType

There may be cases when a Structured Payload object/association should extend the NIEM ComplexObjectType rather than a higher-level type. This may be necessary due to inheritance issues with other types, for example, if data elements would be duplicated in the Structured Payload object/association because of the required contents of the subset schema. It may also be necessary in cases where the Structured Payload object/association needs to build upon multiple types where the only common denominator is ComplexObjectType, for example, if a Structured Payload object builds upon either a Person or Location.

The example below illustrates the definition of an association that extends ComplexObjectType. This is not an unusual occurrence when the Structured Payload needs to include associations to objects that only exist in the Structured Payload, requiring the subset schema to include the date elements in AssociationType. Extension of AssociationType and having the association start/end dates available in schema for both the Digest and Structured Payload associations would lead to confusion on the part of implementers since the association begin/end dates could appear in two different places in the instance. Therefore, it is generally preferable for the Structured Payload association to extend ComplexObjectType as shown below.

The example below is the same as shown in the previous section, except for the use of ComplexObjectType as the base instead of a subset of AssociationType.

```
<xsd:complexType name="FinancialSupporterAssociationType">
  <xsd:annotation>
    <xsd:documentation>A data type for a relationship between a person and a person or an organization that provides
        financial support or assistance.</xsd:documentation>
    <xsd:appinfo>
      <i:Base i:namespace="http://niem.gov/niem/structures/2.0" i:name="Association"/>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="s:ComplexObjectType">
      <xsd:sequence>
        <xsd:element ref="new: FinancialSupportAmount" minOccurs="0"/>
        <xsd:element ref="new: FinancialSupportDescriptionText" minOccurs="0"/>
      </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 a person or an organization that provides financial
        support or assistance.</xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="FinancialSupportAmount" type="nc:AmountType" nillable="true">
  <xsd:annotation>
    <xsd:documentation>A monetary amount provided by the financial supporter to the entity receiving
      support.</xsd:documentation>
  </xsd:annotation>
</xsd:element>
<xsd:element name="FinancialSupportDescriptionText" type="nc:TextType" nillable="true">
  <xsd:annotation>
    <xsd:documentation>A description of the financial support or assistance provided by the financial supporter to the
        subject.</xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

Figure 12 - Specialization of Digest Association by Extending ComplexObjectType in Schema

Note the use of the i:name=”Association” in the appinfo for the FinancialSupporterAssociationType. When using ComplexObjectType to create a new type, either “Object” or “Association” must be used as the value for the i:name attribute; in this case we are creating an association.

The instance is the same regardless of whether the association type extends ComplexObjectType versus an empty AssociationType.

4.5.5 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 NIEM subset, along with any data elements that are necessary for the exchange. If the NIEM type contains everything that is needed for the exchange, the subset would also include the appropriate NIEM element of that type. 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 support 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.

```
<xsd:complexType name="TruckDepotType">
  <xsd:annotation>
    <xsd:documentation>A data type for a repair facility specializing in truck repair.</xsd:documentation>
    <xsd:appinfo>
      <i:Base i:namespace="http://niem.gov/niem/niem-core/2.0" i:name="FacilityType"/>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="nc:FacilityType">
      <xsd:sequence>
        <xsd:element ref="new:SupportedTruckMakeName" 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="SupportedTruckMakeName" type="nc:ProperNameTextType" nillable="true">
  <xsd:annotation>
    <xsd:documentation>A name of a truck make that can be accommodated by a repair facility.</xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

Figure 13 - Structured Payload-only Object Based on a Specialized NIEM Object in Schema

The example below shows the use of this truck depot element in an instance. The sample does not show any Digest since the depot element only appears in the Structured Payload.

```
<ulex:PublishMessage>
…
  <ulex:DataItemPackage>
  …
    <ulex:StructuredPayload ulexlib:id="SP1">
      <ulex:StructuredPayloadMetadata>
        <ulex:CommunityURI>http://somewhere.gov/new-community</ulex:CommunityURI>
        <ulex:CommunityVersion>1.0</ulex:CommunityVersion>
      </ulex:StructuredPayloadMetadata>
      <ulex:StructuredPayloadContent>
        <new:IncidentReport>
          <new:TruckDepot>
            <nc:FacilityName>Northern Truck Repair</nc:FacilityName>
            <nc:FacilityIdentification>
              <nc:IdentificationID>000000000003</nc:IdentificationID>
            </nc:FacilityIdentification>
            <new:SupportedTruckMakeName>GMC</new:SupportedTruckMakeName>
            <new:SupportedTruckMakeName>Kenworth</new:SupportedTruckMakeName>
          </new:TruckDepot>
        </new:IncidentReport>
      </ulex:StructuredPayloadContent>
    </ulex:StructuredPayload>
  </ulex:DataItemPackage>
</ulex:PublishMessage>
```

Figure 14 - 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 ComplexObjectType, which can be used as the basis for Structured Payload objects that cannot derive from any higher-level NIEM objects.

For example, assume an exchange needs to represent a mathematical formula, which doesn't correspond to any existing NIEM or LEXS objects or associations. For simplicity, assume that the mathematical formula consists of a simple string element for the equation and a text description. Therefore, the formula can be represented as shown in the following schema snippet.

```
<xsd:complexType name="MathematicalFormulaType">
  <xsd:annotation>
    <xsd:documentation>A data type for a mathematical formula, providing equation and description.</xsd:documentation>
    <xsd:appinfo>
      <i:Base i:namespace="http://niem.gov/niem/structures/2.0" i:name="Object"/>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="s:ComplexObjectType">
      <xsd:sequence>
        <xsd:element ref="new:EquationText"/>
        <xsd:element ref="nc:DescriptionText" minOccurs="0"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="MathematicalFormula" type="new:MathematicalFormulaType">
  <xsd:annotation>
    <xsd:documentation>A mathematical formula.</xsd:documentation>
  </xsd:annotation>
</xsd:element>

<xsd:element name="EquationText" type="nc:TextType" nillable="true">
  <xsd:annotation>
    <xsd:documentation>A text representation of a mathematical formula.</xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

Figure 15 - Structured Payload-only Object Based on ComplexObjectType in Schema

Note the use of the iName="Object" in the appinfo for the type. This indicates that this is an object type versus an association type. This example leverages the NIEM DescriptionText element rather than creating one specific for the formula element, although the schema could create a new element with specific semantics if desired, such as EquationDescriptionText.

The example below shows the use of this mathematical formula element in an instance. The sample does not show any Digest since the mathematical formula only appears in the Structured Payload.

```
<ulex:PublishMessage>
…
  <ulex:DataItemPackage>
  …
    <ulex:StructuredPayload ulexlib:id="SP1">
      <ulex:StructuredPayloadMetadata>
        <ulex:CommunityURI>http://somewhere.gov/new-community</ulex:CommunityURI>
        <ulex:CommunityVersion>1.0</ulex:CommunityVersion>
      </ulex:StructuredPayloadMetadata>
      <ulex:StructuredPayloadContent>
        <new:IncidentReport>
          <new:MathematicalFormula>
            <new:EquationText>A=B+C</new:EquationText>
            <nc:DescriptionText>A formula for adding two values.</nc:DescriptionText>
          </new:MathematicalFormula>
        </new:IncidentReport>
      </ulex:StructuredPayloadContent>
    </ulex:StructuredPayload>
  </ulex:DataItemPackage>
</ulex:PublishMessage>
```

Figure 16 - Structured Payload-only Object Based on ComplexObjectType in XML

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

A Structured Payload may need to build upon the contents of another Structured Payload. The existing Structured Payload object or association may in turn build upon the Digest, 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.

The primary difference between building upon something in the Digest versus another Structured Payload has to do with linking the various pieces together in an instance, as shown in the XML example below. Note that in order to simplify the instance, no Digest is shown.

```
<ulex:PublishMessage>
…
  <ulex:DataItemPackage>
  …
    <!--======================== Structured Payload #1 ========================-->
    <ulex:StructuredPayload ulexlib:id="SP1">
      <ulex:StructuredPayloadMetadata>
        <ulex:CommunityURI>http://somewhere.gov/new-community1</ulex:CommunityURI>
        <ulex:CommunityVersionText>1.0</ulex:CommunityVersionText>
      </ulex:StructuredPayloadMetadata>
      <ulex:StructuredPayloadContent>
        <new1:Report xmlns:new1="http://somewhere.gov/new-community1">
          <new1: XYZ s:id="ID1">
            <new1:UserName>xyz</new1:UserName>
            <new1:System>abc</new1:System>
          </new1:XYZ>
        </new1:Report>
      </ulex:StructuredPayloadContent>
    </ulex:StructuredPayload>
    <!--======================== Structured Payload #2 ========================-->
    <ulex:StructuredPayload ulexlib:id="SP2">
      <ulex:StructuredPayloadMetadata>
        <ulex:CommunityURI>http://somewhere.gov/new-community2</ulex:CommunityURI>
        <ulex:CommunityVersionText>1.0</ulex:CommunityVersionText>
      </ulex:StructuredPayloadMetadata>
      <ulex:StructuredPayloadContent>
        <new2:Report xmlns:new2="http://somewhere.gov/new-community2">
          <new2:XYZ s:id="ID2">
            <new2:FirstUseDate>2000-01-01</new2:FirstUseDate>
          </new2:XYZ>
        </new2:Report>
      </ulex:StructuredPayloadContent>
    </ulex:StructuredPayload>

    <ulex:Linkages>
      <ulexlib:SameAsConnection>
        <ulexlib:StructuredPayloadObjectReference ulexlib:structuredPayloadReference="SP1"
                ulexlib:nonValidatingObjectReference="ID1"/>
        <ulexlib:StructuredPayloadObjectReference ulexlib:structuredPayloadReference="SP2"
                ulexlib:nonValidatingObjectReference="ID2"/>
      </ulexlib:SameAsConnection>
    </ulex:Linkages>

  </ulex:DataItemPackage>
</ulex:PublishMessage>
```

Figure 17 - Linking Structured Payload Objects in XML

4.5.7 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 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 AttachmentURI to link the Structured Payload version to the LEXS one. The example below shows a specialization of Attachment for a fingerprint image.

```
<xsd:complexType name="FingerprintImageType">
  <xsd:annotation>
    <xsd:documentation>A data type for a fingerprint of a person.</xsd:documentation>
    <xsd:appinfo>
      <i:Base i:namespace="http://niem.gov/niem/structures/2.0" i:name="Object"/>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="s:ComplexObjectType">
      <xsd:sequence>
        <xsd:element ref="ulex:AttachmentURI"/>
        <xsd:element ref="ansi-nist:FingerprintImagePosition"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="FingerprintImage" type="new:FingerprintImageType" nillable="true">
  <xsd:annotation>
    <xsd:documentation>An image of a fingerprint of a person.</xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

Figure 18 - Extending a LEXS Attachment

There are a few things to note about the schema above. (1) The AttachmentURI is included and defined as mandatory in order to ensure that there is something in the Structured Payload that can be used to link to the correct Attachment. Note that LEXS does not require that an Attachment be included in an instance; the Attachment could have been provided through a different mechanism, or could have been part of another LEXS submission. Therefore there is no guarantee that a SameAsConnection element would have the LEXS Attachment to reference. So in keeping with the LEXS approach of using AttachmentURI to link an Attachment with an AttachmentSummary, Structured Payloads should also use the AttachmentURI to link. (2) The FingerprintImageType above is based on ComplexObjectType rather than ImageType or BinaryType. The LEXS Attachment does not extend the NIEM Binary or Image types, but rather includes the NIEM Binary as an element. Therefore, the Structured Payload FingerprintImageType should extend the more basic ComplexObjectType.

The XML example below shows the FingerprintImage linked to an Attachment.

```
<ulex:PublishMessage>
…
  <ulex:DataItemPackage>
  …
    <ulex:StructuredPayload ulexlib:id="SP1">
      <ulex:StructuredPayloadMetadata>
        <ulex:CommunityURI>http://somewhere.gov/new-community</ulex:CommunityURI>
        <ulex:CommunityVersion>1.0</ulex:CommunityVersion>
      </ulex:StructuredPayloadMetadata>
      <ulex:StructuredPayloadContent>
        <new:IncidentReport>
          <new:FingerprintImage>
            <lexs:AttachmentURI>http://somewhere.gov/images/image15.gif</lexs:AttachmentURI>
            <ansi-nist:FingerprintImagePosition>
              <ansi-nist:FingerPositionCode>0</ansi-nist:FingerPositionCode>
            </ansi-nist:FingerprintImagePosition>
          </new:FingerprintImage>
        </new:IncidentReport>
      </ulex:StructuredPayloadContent>
    </ulex:StructuredPayload>
  </ulex:DataItemPackage>

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

Figure 19 - Extending a LEXS Attachment in XML

Note that the AttachmentURI in the Structured Payload instance must match the AttachmentURI of the LEXS Attachment.

4.5.8 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. 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, a Vehicle reference element, and a Facility reference element. Since the Truck Depot is of FacilityType, the NIEM Facility reference 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://niem.gov/niem/niem-core/2.0" i:name="AssociationType"/>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="nc:AssociationType">
      <xsd:sequence>
        <xsd:element ref="nc:FacilityReference"/>
        <xsd:element ref="nc:VehicleReference"/>
      </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 20 - Structured Payload Association Referencing a Placeholder Element in Schema

```
<ulex:PublishMessage>
…
  <ulex:DataItemPackage>
  …
    <lexsdigest:EntityVehicle>
      <lexsdigest:Digest>
        <nc:Vehicle s:id="Vehicle1">
          <nc:ItemDescriptionText>Dent on driver door</nc:ItemDescriptionText>
        </nc:Vehicle>
      </lexsdigest:Digest>
    <lexsdigest:EntityVehicle>

    <ulex:StructuredPayload ulexlib:id="SP1">
      <ulex:StructuredPayloadMetadata>
        <ulex:CommunityURI>http://somewhere.gov/new-community</ulex:CommunityURI>
        <ulex:CommunityVersion>1.0</ulex:CommunityVersion>
      </ulex:StructuredPayloadMetadata>
      <ulex:StructuredPayloadContent>
        <new:IncidentReport>
          <new:TruckDepot s:ref="SPF1"/>
            <nc:FacilityName>Northern Truck Repair</nc:FacilityName>
            <nc:FacilityIdentification>
              <nc:IdentificationID>000000000003</nc:IdentificationID>
            </nc:FacilityIdentification>
          </new:TruckDepot>

          <nc:Vehicle s:id="SPV1"/>

          <new:TruckDepotVehicleAssociation>
            <nc:FacilityReference s:ref="SPF1"/>
            <nc:VehicleReference s:ref="SPV1"/>
          </new:TruckDepotVehicleAssociation >
        </new:IncidentReport>
      </ulex:StructuredPayloadContent>
    </ulex:StructuredPayload>

    <ulex:Linkages>
      <ulexlib:SameAsConnection>
        <ulexlib:DigestObjectReference ulexlib:validatingObjectReference="Vehicle1"/>
        <ulexlib:StructuredPayloadObjectReference ulexlib:structuredPayloadReference="SP1"
                ulexlib:nonValidatingObjectReference="SPV1"/>
      </ulexlib:SameAsConnection>
    </ulex:Linkages>

  </ulex:DataItemPackage>
</ulex:PublishMessage>
```

Figure 21 - 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. So a consumer would know that the vehicle being referenced in the association is the same vehicle contained in the Digest.

5 Utilizing Non-LEXS-aware Structured Payloads

Non-LEXS-aware Structured Payloads are used as is in conjunction with LEXS. Instances conforming to the non-LEXS-aware schemas are "plugged in" to the LEXS message in the StructuredPayloadContent element exactly the same way as LEXS-aware Structured Payload instances. Instances utilize the SameAsConnection element exactly the same way as for LEXS-aware Structured Payloads. The only difference in instances is that non-LEXS-aware Structured Payloads will duplicate at least some of the information in the Digest. Communities that utilize LEXS in this way should ensure that the information in the Digest is as complete and accurate a representation of overlapping content as much as possible. In other words, developers should not limit the contents of the Digest to a handful of elements when more elements are available. The Digest is intended to provide interoperability between implementations by providing a consistent and well-defined set of data for all exchanges that utilize LEXS. Scrimping on the contents of the Digest nullifies one of the main advantages of using LEXS.

It should also be noted that a non-LEXS-aware Structured Payload may have been defined without the XML "id" attributes that are part of LEXS and NIEM. If this is the case, the Linkages elements cannot refer to objects inside the StructuredPayloadContents element by id. However, LEXS also provides the capability to link elements with an XPath-based mechanism. Keep in mind that this XPath-based linking should be limited to schemas that do not include the XML "id" attribute. The example below illustrates the use of the XPath linking mechanism.

```
<ulex:PublishMessage>
…
  <ulex:DataItemPackage>
  …
    <lexs:Digest>
      <lexsdigest:EntityPerson >
        <lexsdigest:Person s:id="Person1">
          <nc:PersonName>
            <nc:PersonGivenName>Jane</nc:PersonGivenName>
            <nc:PersonSurName>Doe</nc:PersonSurName>
          </nc:PersonName>
        </lexsdigest:Person>
      </lexsdigest:EntityPerson>
    </lexs:Digest>

    <ulex:StructuredPayload ulexlib:id="SP1">
      <ulex:StructuredPayloadMetadata>
        <ulex:CommunityURI>http://somewhere.gov/new-community</ulex:CommunityURI>
        <ulex:CommunityVersion>1.0</ulex:CommunityVersion>
      </ulex:StructuredPayloadMetadata>
      <ulex:StructuredPayloadContent>
        <new:Report>
          <new:Person>
            <new:FirstName>Jane</new:FirstName>
            <new:LastName>Doe</new:LastName>
            <new:BuildCode>ST</new:BuildCode>
          </new:Person>
        </new:Report>
      </ulex:StructuredPayloadContent>
    </ulex:StructuredPayload>

    <ulex:Linkages>
      <ulexlib:SameAsConnection>
        <ulexlib:DigestObjectReference ulexlib:validatingObjectReference="Person1"/>
        <ulexlib:StructuredPayloadObjectReference ulexlib:structuredPayloadReference="SP1"
                    ulexlib:xpathObjectReference="/new:Report/new:Person"/>
      </ulexlib:SameAsConnection>
    </ulex:Linkages>

  </ulex:DataItemPackage>
</ulex:PublishMessage>
```

Figure 22 - XPath Referencing for non-LEXS-aware Structured Payload in XML

6 References

6.1 Conformance Testing Assistant

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 on-line, as a desktop tool, and as Web components that can be installed in an implementer's Web server. The online version, as well as links to the desktop and Web components, are available at http://contesa.ittl.gtri.org.

6.2 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.

6.3 LEXS User Guide

The LEXS user guide provides detailed documentation on the LEXS specifications. 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 high level LEXS constructs used in the various operations and includes a detailed list of the various roles, objects, and associations included in the specifications. The latest version of the user guide can be downloaded from the LEXS community web site.

6.4 LEXS Component Mapping Workbook

The Component Mapping Workbook (CMW) is the LEXS data model represented as a spreadsheet. This multi-page spreadsheet documents the data and metadata elements as well as associations that are part of the information exchange, including their locations in schema. The CMW also documents the cardinality of all data elements and attributes. The latest version of the workbook can be downloaded from the LEXS community web site.

6.5 LEXS Code Tables Spreadsheet

The Code Tables spreadsheet contains all code lists used in the LEXS schemas, regardless of whether they are from NIEM or are specific to LEXS. The latest version of the workbook can be downloaded from the LEXS community web site.

6.6 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 mentioned above is available on the web site. The site is located at http://niem.gov.

6.7 Subset Generator Tool

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 wantlist file, allowing for the subset to be modified and regenerated. It is available at http://tools.niem.gov/niemtools/ssgt/index.iepd.

6.8 NIEM Conformance Tool

The NIEM conformance tool assists developers by automatically identifying potential locations of non-conformance within IEPD artifacts (such as schemas, metadata, catalog, etc.) using the latest published NIEM NDR, and associated specifications. A user uploads an IEPD, a set of

schemas in a zip file, or an individual schema. A report is generated outlining any rules that the files violate. The tool is located at http://tools.niem.gov/conformance/index.html.

6.9 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.

6.10 Techniques for Building and Extending NIEM XML Components Document

The "Techniques for Building and Extending NIEM XML Components" document introduces the fundamental technical aspects of NIEM at a high level, and provides an overview of the NIEM IEPD development lifecycle. It discusses the key NIEM data model concepts, and then outlines the basic techniques for extending the NIEM. The paper concludes with a discussion of some standard NIEM extensions. The document is available on the NIEM.gov web site as well as from the LEXS community web site.