August 7, 2007

TECHNIQUES FOR BUILDING AND EXTENDING NIEM XML COMPONENTS

VERSION 2.0.1

Georgia Tech Research Institute

URI: http://reference.niem.gov/niem/guidance/techniques-for-building-and-extending/2.0.1/

Abstract:
The National Information Exchange Model (NIEM) provides structure, standards, and methods for defining and sharing information exchanges between and within agencies and domains. This paper 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 NIEM. The paper concludes with a discussion of some standard NIEM extensions.

⟨NIEM⟩™

Table of Contents

1 Introduction

The National Information Exchange Model (NIEM) provides structure, standards and methods for defining and sharing information exchanges between and within agencies and domains. Developing and implementing exchanges using NIEM means that the major investments local, state, tribal, and federal governments have made in existing information systems can be leveraged, and that these government agencies can efficiently participate in a truly national information sharing environment. NIEM standards enable different information systems to share and exchange information irrespective of the particular technologies at use. Moreover, creating and adopting NIEM standards means that local, state, tribal, and federal organizations avoid the problem of rebuilding or significantly altering their systems to share information.

Conceptually, NIEM can be thought of as a data model and a reference vocabulary from which XML Schema-based data components are rendered. These components serve as the basis for these information exchanges. In conjunction with 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 specification, or 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. An exchange instance that obeys these rules is referred to as an Information Exchange Package (IEP).

In some cases, the NIEM data model provides everything needed to create an information exchange, and the practitioner may simply wish to use the existing data components, or a subset of them, to model the exchange. However, for concepts needed in the exchange that are not adequately represented by existing NIEM data components, the practitioner can model additional concepts to supplement those data components. These new concepts are represented in the form of new XML Schema types and elements which unambiguously define the additional syntax and semantics of the information being exchanged.

This paper introduces 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 and augmenting the NIEM provided data components, for creating meaningful links between new and existing data items and for adapting external standards for use in the NIEM framework.

The audience for this paper is government practitioners and developers who employ XML for information exchange and interoperability. This community will use the NIEM framework and the techniques outlined in this paper to establish and standardize their schemas and documents for use on a national level. At the same time, the NIEM framework and extension techniques allow these users the freedom to create data constructs that also satisfy requirements at the local level.

2 NIEM Overview

The NIEM is a reference model of unconstrained components rendered in XML Schema. Associated with the NIEM schemas is an XML reference architecture that organizes and guides the employment of the various kinds of schemas that compose a NIEM information exchange. The XML reference architecture describes the relationships between XML schemas for NIEM IEPDs.

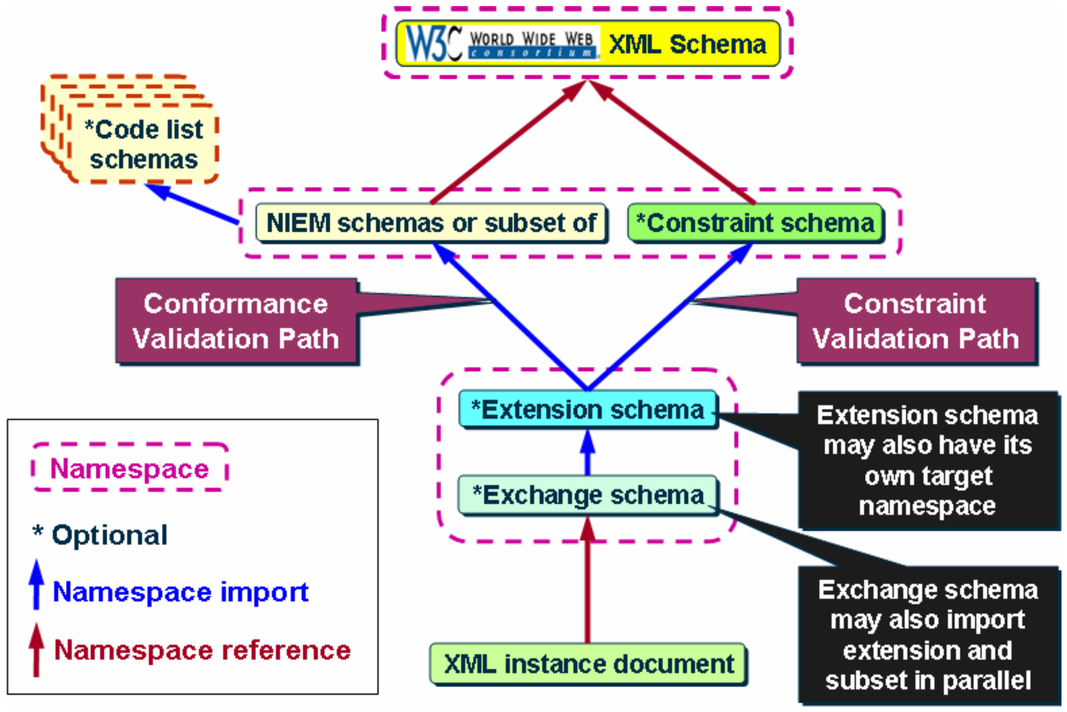

W3C WORLD WIDE WEB consortium XML Schema
*Code list schemas
NIEM schemas or subset of
*Constraint schema
Conformance Validation Path
Constraint Validation Path
*Extension schema
*Exchange schema
Extension schema may also have its own target namespace
Exchange schema may also import extension and subset in parallel
XML instance document
Namespace
* Optional
Namespace import
Namespace reference

Figure 1: The NIEM XML Reference Architecture

The NIEM Naming and Design Rules (NDR) document (available from (http://www.niem.gov//files/NIEM-NDR-1-1-lineNum.pdf) specifies the rules for the creation of the XML components and XML data appearing in the NIEM XML reference schemas. XML schemas and components which obey the rules set forth in the NDR are considered to be *NIEM-conformant*.

A NIEM IEPD is a set of artifacts that describe an Information Exchange Package (IEP), a standard message structure as defined by the Federal Enterprise Architecture Consolidated Reference Model Document (available from http://www.whitehouse.gov/omb/egov/documents/FEA_CRM_v21_Final_Dec_2006.pdf) The NIEM IEPD Specification contains a more detailed explanation of IEPDs and their contents, and is available from http://www.niem.gov//files/NIEM_IEPD_Requirements_v2_1.pdf

The following kinds of XML schemas are associated with the NIEM reference architecture:

- NIEM reference schemas: Schemas containing content created or approved by the NIEM steering committees are periodically released in schema distributions.
- NIEM support schemas: NIEM includes two special schemas, the appinfo and the structures schemas, for annotating and structuring NIEM-conformant schemas.
- Extension Schema: a NIEM-conformant schema which adds domain- or application-specific content to the base NIEM model.

- Exchange Schema: a NIEM-conformant schema which specifies a document in a particular exchange.
- Subset Schema: a profile of a NIEM-conformant schema, derived from a reference schema, but which specifies instances that only require a portion of the reference schema.
- Constraint Schema: a schema which adds additional constraints to NIEM-conformant instances, but which is assumed to validate in concert with existing NIEM-conformant or subset schemas. A constraint schema need not validate constraints that are applied by other schemas.

The only mandatory schemas for validation are the NIEM reference schemas or correct subsets. The NIEM schemas may import additional schemas, such as code table schemas, as needed. The optional exchange schema imports, re-uses, and organizes the components from the NIEM for the particular exchange. An optional extension schema may be used to add extended types and properties for components not contained in the NIEM, but which are needed for the exchange.

Note that while only the reference schemas, or subsets thereof, are required for validation of a NIEM-conformant instance, the IEPD specification requires that an IEPD include an exchange schema along with the reference schemas (or subsets) to be considered a complete IEPD.

The exchange and extension schemas can be combined into a single schema and namespace, or can be broken out into separate schemas and corresponding namespaces. The user may decide the best way to organize components. If the extension components will be reused elsewhere, it may be more efficient to maintain them in a separate namespace, rather than including them in a document namespace.

The NIEM reference schemas are over-inclusive and under-constrained. The reason for this approach is that pre-determining all user needs and constraints is rarely possible. The only way to reach consensus on components is to include all obvious requirements and maintain relatively relaxed constraints.

To ensure interoperability, specific component requirements and constraints are determined on a per-exchange basis (in IEPDs). By creating a subset of NIEM Core, reference and code table schemas, the user can limit the components to only those he or she needs. In the future, a business component layer between IEPDs and NIEM will allow domains to apply consistent requirements and constraints for their exchanges.

The basic principle for a subset is that an instance that validates against a correct subset schema will always validate against the full reference NIEM schema set. The user may also adjust cardinality constraints, as desired, within the subset schemas.

Additional constraints may be handled in a constraint schema. A constraint schema is derived from a subset schema. However, it may contain other constraints (for example, additional types for specific constraints). The constraint schema provides an alternative *constraint validation* path that allows the user to reduce the possible set of allowable XML instances, independent of the NIEM schema or subset *conformance validation* path. This is done through multi-pass validation. A correctly constructed XML instance will validate through both the conformance and the constraint path.

3 NIEM IEPD Development Lifecycle - Overview

While this paper is focused on the NIEM extension techniques, it is helpful to understand where those techniques fit in the IEPD development lifecycle. This section briefly discusses the IEPD lifecycle, and shows where NIEM extension may be necessary.

Figure shows the high level outline of the IEPD lifecycle.

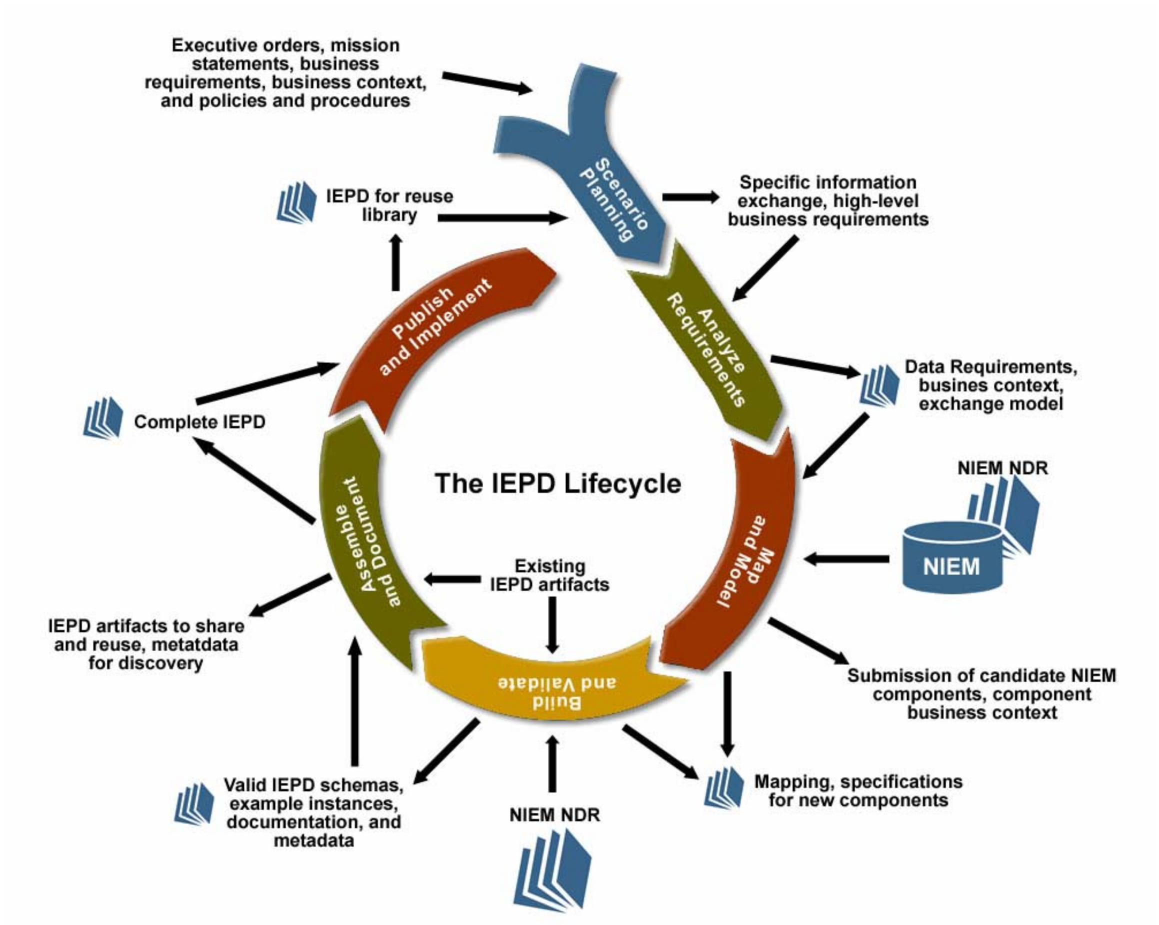

Executive orders, mission statements, business requirements, business context, and policies and procedures
Scenario Planning
IEPD for reuse library
Specific information exchange, high-level business requirements
Analyze Requirements
Publish and Implement
Data Requirements, busines context, exchange model
Complete IEPD
The IEPD Lifecycle
NIEM NDR
NIEM
Assemble and Document
Map and Model
Existing IEPD artifacts
IEPD artifacts to share and reuse, metatdata for discovery
Build and Validate
Submission of candidate NIEM components, component business context
Valid IEPD schemas, example instances, documentation, and metadata
NIEM NDR
Mapping, specifications for new components

Figure 2: IEPD Lifecycle

This section will briefly discuss each of the steps that appear in the figure. For more detailed information on the lifecycle, please refer to the IEPD Development Lifecycle section of the NIEM Concept of Operations, available on the NIEM Web site at http://www.niem.gov//files/NIEM_Concept_of_Operations.pdf While this paper focuses primarily on the XML creation aspects of the IEPD lifecycle, it should be stressed that all steps in the lifecycle are important in the creation of an IEPD.

Step 1: Scenario Planning and Business Taxonomies

Scenario planning and business taxonomies are two methodologies that can be used for identifying specific information exchanges. Scenario planning is a bottoms-up approach and depicts either current information exchange practices among involved parties, or potential future environments that envision broader and more expansive information sharing, as well as changes in business practices or processes. Business taxonomies employ a top-down approach which requires documenting the business operations of an organization using a common framework, such as that defined in the Federal Enterprise Architecture Business Reference Model (BRM), available at http://www.whitehouse.gov/omb/egov/a-3-brm.html. Regardless which approach a developer takes—scenario planning or business taxonomy analysis—the result is the identification of specific information exchanges that are the object of IEPD development.

Step 2: Analyze Requirements - Identify and Document Information Exchange Requirements

The IEPD developer selects one or more related exchanges identified during Step 1 to fully elaborate and build a domain model. The IEPD developer may decide to document multiple information exchanges inherent in an operational scenario or business requirement using the IEPD lifecycle, following the steps which are here described. To build a domain model, the IEPD developer can use information exchange modeling (IEM) tools to model the precise nature and content of the exchange. The outputs of this step include the business context, data requirements and a domain model for the exchange.

Step 3: Map Domain Model to NIEM

The IEPD developer maps NIEM components to the data components (or requirements) in the domain model using his preferred tools. During this mapping, the developer may find there are matches, partial matches, and no matches between the domain model and data components in the NIEM model.

> When there are partial matches or no matches, the extension techniques outlined in this document will be used to add local extensions to the IEPD. These may become candidates for later submission to NIEM.

Matching components can involve those where the component names may differ but where the data components themselves are semantically and structurally equivalent, i.e., there is a one-to-one mapping between the NIEM and the source component. Partial matches can arise when there are similarities, but also some differences between data components. These differences can include semantic or structural mismatches, element naming collisions, or mismatches at the value set, data type or lexical levels. For partial matches, it is necessary to document the need for extension or refinement of existing data components.

Data components with no matching NIEM data components comprise a set of additional element types that are candidates for insertion into the NIEM. Depending on the nature of the potential inclusion in the model, recommendations may include adding a new or subordinate type, adding an element, extending a value set, modifying a data type or lexical representation, renaming data components, or revising a definition. For components that do not match at all, a NIEM-conformant component must be created, following the rules specified in the NIEM Naming and Design Rules (NDR).

The output of this step is the component mapping between the domain model in Step 2 and NIEM, along with the newly modeled components. These newly modeled components become new candidate components for submission to NIEM.

Step 4: Build and Validate IEPDs

Based on the new data components and component mapping identified in the map and model step, the IEPD developer builds and generates IEPD schema artifacts including:

- *Subset schema:* Constructed by extracting from the reference schema set those types and elements needed for a specific information exchange. The NIEM Schema Subset Generation Tool (SSGT) can assist with this process.
- *Extension schema:* Defines an IEP-specific namespace that contains types, elements, and attributes needed for the IEP but which are not in NIEM.
- *Constraint schema:* Adds additional constraints (such as cardinality) or restrictions to the types and elements in the subset. Constraint schemas do not have to be NIEM-conformant.
- *Exchange schema:* Contains the document element (also known as the root element) and may also define basic IEP content.

The extension and exchange schemas will contain the XML representations of the local extension of the NIEM model. These representations are the artifacts of the extension techniques outlined in this document.

The subset and exchange schemas are mandatory for an IEPD. The extension and constraint schemas are optional.

As part of this step, the IEPD developer may also build one or more sample XML instances and eXtensible Stylesheet Language (XSL) stylesheets. These XML instances are examples of the data exchange documents defined by this IEPD and the payload documents that are actually exchanged. Not only do they serve as example artifacts for the IEPD, but they can also be used to validate the schemas for the IEPD. XSL stylesheets are used to consistently format the data within the XML instances to meet display or output requirements.

To further define the IEPD, additional documentation is also needed. This should include the domain model, business rules, change log (or the initial file for such), basic metadata, a catalog or manifest, and other artifacts as required by the NIEM IEPD Specification. The outputs of this step are the valid schemas, example instances, documentation artifacts, and metadata.

Step 5: Assemble IEPDs to IEPD Specification

Once all of the schemas, documentation, metadata, etc. have been captured, the IEPD can be generated based on the format defined in the NIEM IEPD Specification. The NIEM IEPD Tool can assist with this process. The output of this step is a complete IEPD, which provides reference for other users.

Step 6: Publish and Implement Exchanges

The final output of the IEPD lifecycle is an IEPD that is published and available for search, discovery and (re)use. The NIEM IEPD Specification defines a portable, self-contained, self-documented, machine readable package that enables IEPD registration and storage in virtually any location. IEPD developers have the option of publishing an

IEPD to their own repository, an industry-wide repository, or to register and publish an IEPD through the NIEM Program.

4 NIEM Data Model Concepts

The NIEM data model is based on two types of data model constructs

- NIEM types
- NIEM properties

Together, these two data model concepts are used to model generic and domain-specific concepts in a way that maximizes reuse and extension within the NIEM framework.

4.1 NIEM Types

A *NIEM type* corresponds to the abstraction of either a real world object, such as a name, or a conceptual object, such as a relationship. In object-oriented terminology, a NIEM type corresponds to a "class." NIEM types are represented in XML Schema as XML Schema complex and simple types.

All XML Schema types associated with NIEM types must incorporate one of the NIEM base types that are defined in the NIEM Structures schema. These NIEM base types contain the hooks for adding NIEM-compliant constructs to the NIEM data model, such as new NIEM types, or custom metadata and/or augmentations for existing NIEM types. The details of these local extension techniques will be discussed in Section 5.

The process of representing specific objects of the NIEM types in an XML document is called "instantiation". The XML fragment that is created based on the NIEM type is considered to be an "instance" of the NIEM type. In object-oriented terminology, an instance of a NIEM type corresponds to an "object."

Instances of NIEM types appear in XML exchange documents relevant to a particular information exchange. These documents obey the rules set forth in the XML Schemas generated from the NIEM data model, and in the exchange-specific NIEM document or extension schemas.

Note that the unmodified term "type" is ambiguous in the context of discussing the NIEM data objects – it could either refer to a NIEM type or an XML Schema type that represents the NIEM type in an XML Schema. In this paper, we will always refer to NIEM types or XML Schema types so it is clear from the context which kind of type is being discussed, the data model concept or its XML Schema representation. In addition, a reference to a NIEM type appears in quotes, like this

"ComplexObjectType"

whereas the XML Schema representation of that type appears in a `Courier` font, and is preceded with the standard XML namespace prefix associated with the containing schema's namespace, like this:

`s:ComplexObjectType`

4.2 NIEM Properties

A *NIEM property* describes a pair-wise relationship between instances of NIEM types. NIEM properties have two variations – a pair-wise relationship between a NIEM component and a value, and a pair-wise relationship between two NIEM components. The way NIEM types and properties work together, and are represented in XML Schema, are described as follows:

- A NIEM type has one or more NIEM properties.
  - This rule is represented in XML Schema by defining the properties as XML Schema elements within an XML Schema complex type.
- A NIEM property has a value.
  - This rule is trivially represented in XML Schema by the assignment of values to XML Schema components.
- The NIEM property's value is an instance of another NIEM type.
  - This rule is represented in XML Schema by assigning values that are instances of XML Schema simple or complex types.

The association of a NIEM property with a NIEM type means that an instance of a NIEM type has a characteristic, a relationship, or a subpart represented by an instance of that NIEM property. For example:

- A NIEM type "PersonType" may have the property "BirthLocation" to indicate a relationship, namely the place where a person was born.
- A NIEM type "PersonType" may have the property "EyeColor", to indicate a characteristic of the person.
- A NIEM type "VehicleType" may have the property "Cargo", a subpart which represents the contents of the vehicle.

The NIEM data model does not make concrete distinctions between kinds of NIEM properties. All NIEM properties are represented as XML Schema elements and attributes in an XML Schema type, regardless of how the property is used in the NIEM type. However, naming conventions for a NIEM property can provide semantic information on how NIEM types use the property. The use of naming conventions in naming properties is discussed in more detail in the NIEM NDR.

The relationship between a NIEM property and a NIEM type may be semantically strong, such as the birth location of a person, or semantically weak, with its exact meaning left unstated. In NIEM, the property involved in the semantically weak relationship is commonly referred to as a *container element*.

The name of the container element is usually based on the NIEM type that defines it. The appearance of a container element within a NIEM type carries no additional semantics about the relationship between the property and the containing type. Use of container elements indicate only that there is a relationship, but does not provide any semantics for interpreting that relationship.

For example, a NIEM container element `nc:Person` would be associated with the NIEM type `nc:PersonType`. The use of the NIEM container element `nc:Person` in a containing NIEM type indicates that a person has some association with the instances of the containing NIEM type. But because the `nc:Person` container element is used, there is no additional meaning about the association of the person and the instance containing it. While there is a person associated with the instance, nothing is known about the relationship except its existence.

The use of the Person container element is in contrast to a NIEM property named `nc:AssessmentPerson`, also of NIEM type `nc:PersonType`. When the NIEM property `nc:AssessmentPerson` is contained within an instance of a NIEM type, it is clear that the person referenced by this property was responsible for an assessment of some type, relevant to the exchange being modeled. The more descriptive name, `nc:AssessmentPerson`, gives more information about the relationship of the person with the containing instance, as compared to the semantic-free implications associated with less descriptive name of the container element `nc:Person.`

Finally, because of the syntax provided by XML Schema, there are two representations of NIEM properties that are included in a NIEM type: *content elements* and *reference elements*. A content element is an XML element that contains its value inline. A reference element is an XML element that refers to an XML construct outside the containing XML fragment rather than inline. Examples of these two alternate representations appear in the next section.

4.3 NIEM 2.0 Conformant Namespaces

A NIEM-conformant namespace is associated with an XML Schema that obeys all rules specified in the NIEM NDR. NIEM types and properties are associated with NIEM-conformant namespaces. The standard NIEM-conformant namespaces are assigned standard namespace prefixes.

In NIEM 2.0, there are three namespace prefixes associated with the NIEM Core:

- `i` – bound to the appinfo namespace. The appinfo namespace contains components which provide additional semantics and syntactic guidelines for components built by NIEM schemas.
- `s` – bound to the structures namespace. The structures namespace contains structures for organizing NIEM components.
- `nc` – bound to the NIEM Core namespace. The NIEM Core namespace contains the universal and common components used by NIEM domains. Components in this namespace are marked with metadata to distinguish universal components (used by all or nearly all of the NIEM domains) from common components (used by two or more NIEM domains.)

In NIEM 2.0, there are seven domains. Their standard namespaces prefixes are:

- `em` – Emergency Management
- `infra` – Infrastructure Protection (in DHS)

- `im` – Immigration (in DHS)
- `intel` – Intelligence
- `it` – International Trade (in DHS)
- `j` – Justice
- `scr` – Person Screening

NIEM 2.0 includes a number of code table schemas with their own namespace prefixes. These schemas contain the type definitions and code values for NIEM elements that are enumerated types. The complete list of code table schema namespace prefixes and descriptions of the code table schemas appears in Appendix C.

Finally, NIEM currently includes the schemas for several external standards related to emergency management and geospatial information exchanges. NIEM profiles these schemas for use in IEPDs through adapter types (discussed in Section 6.3). The non-conforming external standard schemas are contained in the NIEM reference schema set within a special subdirectory labeled `external`.

For the emergency management standards, NIEM provides two schemas that contain adapter types for two standard Emergency Management messages. The adapter types in these schemas wrap components from the standard Common Alerting Protocol and Distribution Element schemas contained in the `external` subdirectory. The namespace prefixes and subdirectory labels for these are:

- `edxl-cap` (where cap = Common Alerting Protocol)
- `edxl-de` (where de = Distribution Element)

For the geospatial external standards, NIEM provides a single schema in the `geospatial` subdirectory that contains a large number of adapter types for geospatial components used from external non-conformant schemas in the `external` subdirectory. Within the `external` subdirectory there are 68 geospatial external standard schemas that are partitioned into 18 namespaces. The namespace prefix for the schema that contains the geospatial adapter types is:

- `geo` – Geospatial

NIEM types and properties from any of the namespaces in NIEM 2.0 may be used in developing custom types and properties for use in an IEPD. Additional namespaces may be added to future releases of NIEM, and assigned standard namespace prefixes. Appendix B provides a complete directory tree for the NIEM 2.0 release.

4.4 NIEM Type and Properties Example

An example serves to clarify the rules governing the interaction of NIEM types and properties, and how those rules are represented in NIEM-conformant XML Schema.

Note that in these and other examples in this paper, namespace prefixes are used in the schema fragments, but not in the instance fragments. The default namespace is assigned to the schema or instance being defined. This approach to use of namespace prefixes and default namespace is not mandated by NIEM – the IEPD developer may use namespaces and namespace prefixes as appropriate for the specific tasks in keeping with any rules for their use specified in the NIEM NDR.

In the following XML Schema fragment from the NIEM Core schema, we see a portion of the definition of the NIEM Core Type "PersonType" and its contained property "PersonName".

XML Schema fragment for nc:PersonType

```
<xsd:schema xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
targetNamespace="http://niem.gov/niem/niem-core/2.0"
…>

<xsd:complexType name="PersonType">
    <xsd:complexContent>
       <xsd:extension base="s:ComplexObjectType">
         <xsd:sequence>
           <xsd:element ref="nc:PersonAgeMeasure" minOccurs="0"
maxOccurs="unbounded"/>
           <xsd:element ref="nc:PersonBirthDate" minOccurs="0"
maxOccurs="unbounded"/>
           <xsd:element ref="nc:PersonName" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
       </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

The association of the "PersonName" NIEM property with the "PersonType" NIEM type is represented by the existence of an XML Schema complex type `PersonType` which contains an XML Schema element `PersonName`.

The `PersonName` element is a reference to an existing element, rather than defined inline. So how is that existing element defined? That element is an XML Schema complex type, `PersonNameType`, which happens to be the XML Schema representation of the NIEM type "PersonNameType".

Here is the XML Schema fragment that shows the definition of the “PersonNameType” property:

XML Schema fragment for nc:PersonNameType

```
<xsd:schema xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
targetNamespace=”http://niem.gov/niem/niem-core/2.0"
…>

<xsd:complexType name="PersonNameType">
  <xsd:complexContent>
    <xsd:extension base="s:ComplexObjectType">
      <xsd:sequence>
        <xsd:element ref="nc:PersonNamePrefixText" minOccurs="0"
maxOccurs="unbounded"/>
        <xsd:element ref="nc:PersonGivenName" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="nc:PersonMiddleName" minOccurs="0"
maxOccurs="unbounded"/>
        <xsd:element ref="nc:PersonSurName" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="nc:PersonNameSuffixText" minOccurs="0"
maxOccurs="unbounded"/>
        <xsd:element ref="nc:PersonMaidenName" minOccurs="0"
maxOccurs="unbounded"/>
        <xsd:element ref="nc:PersonFullName" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute ref="nc:personNameCommentText" use="optional"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

As we can see, the “PersonNameType” NIEM type itself contains NIEM properties, represented as XML Schema elements and attributes. These contained NIEM properties may be of NIEM complex types, or of XML Schema simple types, as in the case of a NIEM property represented as an XML attributes. NIEM types may also have simple content of XML Schema simple types, and not be defined in terms of other NIEM properties at all.

Here is a fragment of an XML instance that contains an instance of the NIEM type “PersonType”. In this fragment, the XML element name `Person` is an instance of the XML Schema Type `nc:PersonType`:

XML instance fragment using nc:PersonType as content element

```
<nc:Person xmlns:nc="http://niem.gov/niem/niem-core/2.0">
 <nc:PersonName>
     <nc:PersonGivenName>John</nc:PersonGivenName>
     <nc:PersonMiddleName>Q</nc:PersonMiddleName>
     <nc:PersonSurName>Public</nc:PersonSurName>
  </nc:PersonName>
  <nc:PersonBirthDate>1970-01-01</nc:PersonBirthDate>
</nc:Person>
```

In the previous instance fragment, the XML Schema element `PersonName` is a content element, because the value is an instance of the XML Schema type, `PersonNameType`. By contrast, in the following fragment, the same information appears, but is represented using a reference element, rather than a content element. As shown below, `PersonNameReference` is a reference element, because the value is defined as an instance of the XML Schema base type `s:ReferenceType`, instead of `PersonNameType`. The XML instance of `s:ReferenceType` contains a reference to a "PersonName" instance whose XML representation has been assigned the identifier "`A`". The value pointed to by the reference element contains the same information as was kept inline by the content element example. However, by pulling the element information into a separate component, it is now capable of being shared by multiple NIEM properties using reference elements, rather than being used inline exactly once by a content element.

XML instance fragment using reference element

```
<nc:Person xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:nc="http://niem.gov/niem/niem-core/2.0" >
  <nc:PersonNameReference s:reference="A"/>
  <nc:PersonBirthDate>1970-01-01</nc:PersonBirthDate>
</nc:Person>

<nc:PersonName s:id="A" xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:nc="http://niem.gov/niem/niem-core/2.0" >
  <nc:PersonGivenName>Robert</nc:PersonGivenName>
  <nc:PersonSurName>Smith</nc:PersonSurName>
</nc:PersonName>
```

Whether to represent NIEM properties as content or reference elements is a decision determined by the use and complexity of the information being modeled. This topic will be discussed in more detail in Section 5.4.5.

5 Extension Techniques

There are two approaches for extending the NIEM data model for use in information exchange schemas and documents.

- Creating new NIEM types to represent new concepts
- Adding new data to existing NIEM types, to extend existing concepts

The end result of the data model extensions is a collection of new XML Schema types and elements. These new components will reside in either a NIEM exchange schema (if the extensions are specific to a given exchange) or in a NIEM extension schema (if the extensions could potentially be used by more than one exchange through XML schema import facilities.)

5.1 Designing the Exchange Schema Document Element

As discussed in Section 3, the exchange schema defines the document element (also referred to as the root element) of an exchange. This document element defines the top-level structure of the IEPD instance.

The IEPD drives the design of this top-level element. Although it is called a "document element", this top level element need not be a traditional document, if the exchange is better modeled with a message-passing paradigm. Since XML instances can be document-oriented or data-oriented, the exchange document element should be designed to support document or data-oriented exchange as appropriate.

In the case of document-oriented IEPDs, NIEM provides the NIEM Core Type `nc:DocumentType` as a building block. By deriving types from the "DocumentType" instances of those derived types are distinguishable as documents. In addition, the "DocumentType" contains a large number of NIEM properties which can be used to describe data about the document itself (document metadata.) This collection of properties includes properties such as "DocumentAuthor", "DocumentCreationDate" and so on. In domains that are document-centric, it is recommended that the document element of the exchange derive from the NIEM "DocumentType", so as to clearly mark the IEP as a document, and to inherit this metadata collection for marking documents.

In the case of message-oriented IEPDs, there is no NIEM 'MessageType" provided for derivation purposes. At the time this paper was published, a standard NIEM messaging framework was not available. That said, a typical message-style IEP might have a message header component, followed by a message payload component containing the actual data being exchanged. The payload might be followed by other components for handling exceptions, providing digital signatures and so on. Regardless of how the document element of a message-oriented IEPD exchange schema is designed, it is recommended that any documents that appear in the payload of an IEPD message be derived from the "DocumentType". The reason for this derivation recommendation is the same as for document-oriented IEPDs – to mark this portion of the payload as a document, and to reuse the collection of properties that describe NIEM documents.

5.2 Designing New NIEM Types

After reviewing the NIEM data model, users may find that the concept they wish to represent in their information exchange does not exist in NIEM. In this case, NIEM provides two techniques for creating new NIEM types to represent the new concept:

- composing a new NIEM type from a collection of NIEM properties,
- specializing an existing NIEM type to create a new NIEM type

The addition of a new NIEM type to the NIEM data model results in a local extension of the NIEM data model, with a new data component. This extension is represented as the creation of additional XML Schema types to represent the new data model components.

The new concept may be very simple to model: perhaps all that is needed is a new NIEM type which reuses existing NIEM properties from the core data model, in a new composition or specialization. At the other end of the spectrum, modeling a new concept may trigger the creation of several new NIEM types and properties, all associated with the particular information exchange.

5.2.1 NIEM Type Composition

The basic method for creating NIEM types is by composition of different parts. As discussed earlier, the parts of a NIEM type are NIEM properties. The parts are composed ("put together") as a sequence of NIEM properties.

In the corresponding XML Schema representation, this means the NIEM type is represented as an XML Schema complex type. That XML Schema type is composed of an ordered sequence of XML Schema elements and attributes that correspond to the NIEM properties.

Here is a simple XML Schema fragment that shows the definition of a composite type “MyCompositeType” with two NIEM properties, “MyName” and “MyText.” Those two properties are instances of the NIEM type “TextType” and are not shown in the schema fragment below.

XML Schema example of a composite type

```
<xsd:schema xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:exch="urn:examples.com:techniques:exchange-example"
targetNamespace=”urn:examples.com:techniques:exchange-example"
…>

<xsd:complexType name="MyCompositeType">
  <xsd:complexContent>
    <xsd:extension base="s:ComplexObjectType">
      <xsd:sequence>
        <xsd:element ref="exch:MyName" ... />
        <xsd:element ref="exch:MyText" ... />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

An XML instance of that new composite type would look like this:

XML Instance example of a composite type

```
<exch:MyComposite xmlns:exch="urn:examples.com:techniques:exchange-example" >
   <exch:MyName>George P. Burdell</exch:MyName>
   <exch:MyText>Some text here</exch:MyText>
</exch:MyComposite>
```

5.2.2 NIEM Type Specialization

Specialization is a method that creates a new NIEM type from an existing NIEM type, called the *derived NIEM type*. Wikipedia (http://en.wikipedia.org/wiki/Specialization) defines specialization as follows:

> Concept B is a specialization of concept A if and only if:
>
> - every instance of concept B is also an instance of concept A; and
> - there are instances of concept A which are not instances of concept B.
>
> For instance, 'Bird' is a specialization of 'Animal' because every bird is an animal, and there are animals which are not birds (dogs, for instance).

Derived NIEM types must also obey two additional rules:

- Specializations represent permanent, time-independent characteristics for an instance. For example, it is incorrect to design a specialized type to characterize instances of people who are National Guardsman. A person may be on activity duty for a number of years and then discharged, at which point the person is no longer a National Guardsman.

- Specializations are mutually exclusive – there is no overlap between the instances of different derived types. To put it another way, the intersection of instances of one derived type with instances of any other derived types must be null (empty). For example, it is incorrect to design two specialized types, one which characterizes instances of people with black hair and one which characterizes instances of people with brown eyes. Those people with black hair and brown eyes would be instances of both of these specialized types, which is a violation of this mutually exclusive rule.

Specialization in the NIEM data model is represented in XML Schema as XML Schema complex type extension.

As mentioned above, a case is a special form of a NIEM activity and demonstrates the correct use of specialization within the NIEM data model. The concept of a case is modeled by the NIEM type “CaseType”. It is represented by an XML Schema complex type in the NIEM core namespace, `nc:CaseType`. The fact that a case is a specialization of an activity is represented in XML Schema by the `nc:CaseType` (the representation of the concept “CaseType”) extending the `nc:ActivityType` (the representation of the concept “ActivityType”) with the addition of NIEM properties (represented as XSD Schema elements) to the definition of `nc:CaseType`.

Here is a XML Schema fragment that defines the specialized NIEM type “CaseType,” based on the NIEM type “ActivityType” with additional NIEM properties added to it:

XML Schema example of a specialized type

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
targetNamespace=”http://niem.gov/niem/niem-core/2.0”
…>

<xsd:complexType name="CaseType">
  <xsd:complexContent>
    <xsd:extension base="nc:ActivityType">
      <xsd:sequence>
        <xsd:element ref="nc:CaseTitleText" ... />
        <xsd:element ref="nc:CaseCategoryText" ... />
        ...
        <xsd:element ref="nc:CaseResolutionText" ... />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

An XML instance of that specialized NIEM type “CaseType” would look like this:

XML Instance example of a specialized type

```
<nc:Case xmlns:nc="http://niem.gov/niem/niem-core/2.0">
   <!-- instances of properties associated with base nc:ActivityType -->
   <nc:ActivityID>1923</nc:ActivityID>
        ...
   <!-- instances of properties associated with derived type -->
   <nc:CaseTitleText>Murder of Roger Ackroyd<nc:CaseTitleText>
   <nc:CaseCategoryText>Whodunnit</nc:CaseCategoryText>
        ...
   <nc:CaseResolutionText>Suspect everyone.</nc:CaseResolutionText>
</nc:Case>
```

Specialized types must be carefully designed to avoid violating the rules for being permanent and mutually exclusive.

5.3 Adding to Existing NIEM Types

After reviewing the NIEM data model, users may find that the basic concept they need is already part of the NIEM data model, but does not carry all the information needed for a particular information exchange. In this case, NIEM provides two techniques for adding to the existing concept without the overhead of adding new NIEM types,

- adding metadata to an existing NIEM type
- augmenting an existing NIEM type

With the approaches outlined in this section, the concepts in the core data model are reused, but in a way that allows the exchange specific information to be incorporated with the core concepts.

5.3.1 Adding Metadata

Metadata is defined loosely as “data about data”, information that describes the information stored in a database or data model. NIEM metadata can be thought of as a pedigree for the data, storing information about how the data was gathered, who gathered it, when it was gathered, and so on. All of the base NIEM types in the NIEM Structures schema contain a reference to a structure that holds optional metadata for NIEM objects. Since the base NIEM types are the root for all NIEM types in the NIEM data model (whether said types are provided in the data model or custom to an information exchange), all instances of all NIEM types are capable of carrying metadata describing the data in those instances.

NIEM metadata is represented in XML Schema as separate, reusable sets of XML Schema fragments. The association of metadata with NIEM types is represented by adding metadata “hooks”, or empty placeholders, to all the NIEM base types. The representation of these metadata hooks in XML Schema is through the use of one or more XML attributes. The value of these XML attributes is the identifier for a previously defined set of metadata.

NIEM provides two categories of metadata:

- metadata specific to an object
- metadata specific to a relationship(link) between two objects

These two categories of metadata are implemented through the use of two XML Schema attributes defined in the NIEM Structures schema. The XML attribute `s:metadata` assigns a set of metadata specific to the object. The XML attribute `s:linkMetadata` assigns a set of metadata specific to a relationship(link) between two objects. NIEM types may have one or both of these types of metadata available, depending on the NIEM base type it is derived from:

- `s:ComplexObjectType (s:metadata` and `s:linkMetadata)`
- `s:ReferenceType (s:linkMetadata)`
- `s:AugmentationType (s:metadata)`

Multiple sets of metadata, both object-specific and link-specific, depending on the NIEM base type in use, may be applied to a NIEM type. This is handled by supplying multiple values to the XML Schema attribute(s) in the XML schema representation of the NIEM type.

The NIEM metadata design does not enforce the use of particular pieces of metadata – the user is free to design metadata sets to represent his exchange and those metadata sets can be arbitrarily complex, using all the facilities available in XML Schema. However, as a convenience, and to encourage interoperability, a predefined set of object metadata attributes is defined in the NIEM Core schema, using the XML Schema type `nc:MetadataType.`

Here is an XML Schema fragment that shows the definition of metadata in the NIEM structures schema type “ComplexObjectType”. It demonstrates that NIEM types represented as extensions of the XML SchemaType `s:ComplexObjectType` can optionally contain either object or link metadata or both:

XML Schema fragment demonstrating NIEM metadata hooks

```
<xsd:schema xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://niem.gov/niem/structures/2.0"
…>

<xsd:complexType name="ComplexObjectType" abstract="true">
  <xsd:attribute ref="s:id"/>
  <xsd:attribute ref="s:metadata"/>
  <xsd:attribute ref="s:linkMetadata"/>
</xsd:complexType>
```

Looking back at the PersonType example in Section 4.4, we see that the XML Schema type `nc:PersonType` is a specialization of `s:ComplexObjectType`. Therefore, instances of the NIEM type “PersonType”, by virtue of inheritance from the NIEM Structures schema type “ComplexObjectType” can have metadata attached to them without having to compose or specialize another variant of a person.

The following XML Schema fragment demonstrates the use of a metadata set for capturing metadata about the lab that analyzed a piece of DNA evidence.

XML Schema fragment for new metadata set (`DNATestingLabMetadataType`)

```
<xsd:schema xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
xmlns:exch="urn:examples.com:techniques:exchange-example"
targetNamespace="urn:examples.com:techniques:exchange-example"
…>

<xsd:element name="DNATestingLabTechnician" type="nc:PersonType" nillable="true"/>
<xsd:element name="DNATestingLab" type="nc:OrganizationType" nillable="true"/>

<xsd:complexType name="DNATestingLabMetadataType">
     <xsd:complexContent>
         <xsd:extension base="s:MetadataType">
            <xsd:sequence>
               <xsd:element ref="exch:DNATestingLab"/>
               <xsd:element ref="exch:DNATestingLabTechnician"/>
            </xsd:sequence>
          </xsd:extension>
     </xsd:complexContent>
</xsd:complexType>

<xsd:element name="DNATestingLabMetadata" type="exch:DNATestingLabMetadataType"
nillable="true"/>
```

The following XML instance fragment takes advantage of the metadata hooks to add metadata describing the lab technicians analyzing some pieces of DNA evidence.

XML instance example using new metadata set

```

<nc:DNA s:metadata="LabAlice" xmnls="urn:examples.com:techniques:exchange-example"
 xmlns:s="http://niem.gov/niem/structures/2.0"
 xmlns:nc="http://niem.gov/niem/niem-core/2.0">
   <nc:DNAImage>
      <!-- image detail omitted -->
      <nc:ImageLocation>
         <nc:LocationDescriptionText>Cabinet ABC123</nc:LocationDescriptionText>
      </nc:ImageLocation>
   </nc:DNAImage>
</nc:DNA>

<nc:DNA s:metadata="LabAlice" xmnls="urn:examples.com:techniques:exchange-example"
 xmlns:s="http://niem.gov/niem/structures/2.0"
 xmlns:nc="http://niem.gov/niem/niem-core/2.0">
   <nc:DNAImage>
      <!-- image detail omitted -->
      <nc:ImageLocation>
         <nc:LocationDescriptionText>Cabinet ABC456</nc:LocationDescriptionText>
      </nc:ImageLocation>
   </nc:DNAImage>
</nc:DNA>
```

```
<nc:DNA s:metadata="LabBob"
  xmnls="urn:examples.com:techniques:exchange-example"
 xmlns:s="http://niem.gov/niem/structures/2.0"
 xmlns:nc="http://niem.gov/niem/niem-core/2.0">
   <nc:DNAImage>
      <!-- image detail omitted -->
      <nc:ImageLocation>
         <nc:LocationDescriptionText>Cabinet ABC789</nc:LocationDescriptionText>
      </nc:ImageLocation>
   </nc:DNAImage>
</nc:DNA>

<DNATestingLabMetadata s:id="LabAlice"
  xmnls="urn:examples.com:techniques:exchange-example"
 xmlns:s="http://niem.gov/niem/structures/2.0"
 xmlns:nc="http://niem.gov/niem/niem-core/2.0">
   <nc:DNATestingLab>
      <nc:OrganizationName>Dan's DNA Lab</nc:OrganizationName>
   </nc:DNATestingLab>
   <nc:DNATestingLabTechnician>
      <nc:PersonName>
         <nc:PersonFullName>Alice Smith</nc:PersonFullName>
      </nc:PersonName>
   </nc:DNATestingLabTechnician>
</DNATestingLabMetadata>

<DNATestingLabMetadata s:id="LabBob" xmnls="urn:examples.com:techniques:examples"
 xmlns:s="http://niem.gov/niem/structures/2.0"
 xmlns:nc="http://niem.gov/niem/niem-core/2.0">
   <nc:DNATestingLab>
      <nc:OrganizationName>Dan's DNA Lab</nc:OrganizationName>
   </nc:DNATestingLab>
   <nc:DNATestingLabTechnician>
      <nc:PersonName>
         <nc:PersonFullName>Bob Jones</nc:PersonFullName>
      </nc:PersonName>
   </nc:DNATestingLabTechnician>
</DNATestingLabMetadata>
```

5.3.2 Adding Augmentations

Augmentation of a NIEM data type allows the addition of domain- or model-specific information to the concept embodied in the NIEM type, without creating a new NIEM type. It would be impractical and unwieldy to include all possible domain model-specific properties in NIEM Core schemas for general use. Instead, domain modelers need to be able define data for their use, independently from common definitions. Furthermore, that data needs to be applicable to the NIEM data object itself, and reusable in multiple exchanges. The augmentation approach built into NIEM utilizes XML Schema constructs to reuse the existing XML schema representations for the data model, by allowing them to be augmented with the new information.

NIEM augmentations are represented in XML Schema as a collection of XML Schema type extensions, built according to a set of rules governing the augmentation process. An augmentation requires the creation of the following XML Schema entities:

- an XML Schema type derived from the NIEM Structures abstract base type `s:AugmentationType` (i.e. an extension of this base type)

- an XML Schema element whose name has the suffix “Augmentation” and is defined to belong in the substitution group `s:Augmentation`.

The XML Schema fragments from the Structures schema that support augmentation are:

XML Schema fragment containing augmentation support

```
<xsd:schema xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace=”http://niem.gov/niem/structures/2.0"
…>

<xsd:complexType name="AugmentationType" abstract="true">
  <xsd:attribute ref="s:metadata"/>
  <xsd:attribute ref="s:linkMetadata"/>
</xsd:complexType>

<xsd:element name="Augmentation" type="s:AugmentationType"/>
```

Note that the “Augmentation” substitution group, `s:Augmentation`, provides a base for element substitution. The name of this substitution group provides meaningful information since it defines elements using this substitution group as being augmentations.

Augmentations generally contain domain-specific information, and thus, are usually associated with the NIEM domains and domain specific IEPDs, not with the NIEM Core. While augmentation components (elements and their associated types) are considered extensions to the NIEM model and are contained in NIEM, the augmentation components are applied to (i.e. extend) the types they are designed to supplement only within an IEPD schema (not within NIEM itself). This subtle restriction is one reason augmentations are reusable in combinations.

A simple example calling for an augmentation is an IEPD that exchanges information involving commercial vehicles. The augmentation contains additional information about the commercial vehicle’s history with the company that owns it. This augmentation is associated with the base type `nc:CommercialVehicleType`, through the use of the appinfo annotation. Since this is an extension to the NIEM model for a particular domain, the new augmentation container definitions appear within an IEPD extension schema, where it could be reused by other IEPDs as needed.

IEPD extension schema fragment describing a new augmentation type

```
<xsd:schema xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
xmlns:ext="urn:example.com:techniques:extension-example"
targetNamespace=”urn:example.com:techniques:extension-example”
…>

<xsd:complexType name="CommercialVehicleAugmentationType">
   <xsd:complexContent>
      <xsd:extension base="s:AugmentationType">
         <xsd:sequence>
            <xsd:element ref="ext:VehicleCompanyVIN"/>
            <xsd:element ref="ext:VehicleOwningCompany"/>
             <xsd:element ref="ext:VehicleCompanyPurchaseDate"/>
         </xsd:sequence>
      </xsd:extension>
 </xsd:complexContent>

 <xsd:element name="CommercialVehicleAugmentation"
     type="ext:CommercialVehicleAugmentationType"
     substitutionGroup="s:Augmentation">
    <xsd:annotation>
       <xsd:appinfo>
          <i:appliesTo
             i:namespace=”http://niem.gov/niem/niem-core/2.0"
             name="CommercialVehicleType"/>
       </xsd:appinfo>
   </xsd:annotation>
</xsd:element>

<xsd:element name="VehicleCompanyVIN" . . .
<xsd:element name="VehicleOwningCompany" . . . />
<xsd:element name="VehicleCompanyPurchaseDate" . . . />
```

The use of this augmentation container appears within types defined in the IEPD exchange schema:

IEPD exchange schema fragment describing a new augmentation type

```
<xsd:schema xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
xmlns:ext="urn:example.com:techniques:extension-example"
xmlns:exch="urn:example.com:techniques:exchange-example"
targetNamespace=”urn:example.com:techniques:exchange-example”
…>

<xsd:complexType name="MyCommercialVehicleType">
      <xsd:complexContent>
          <xsd:extension base="nc:CommercialVehicleType">
              <xsd:sequence>
                  <xsd:element ref="ext:CommercialVehicleAugmentation"/>
              </xsd:sequence>
          </xsd:extension>
       </xsd:complexContent>
</xsd:complexType>

<xsd:element name="MyCommercialVehicle"
      type="exch:MyCommercialVehicleType"
      substitutionGroup="nc:CommercialVehicle"/>
```

An IEPD XML instance fragment that demonstrates the use of this augmentation would look like this:

IEPD instance example using new augmentation type

```
<exch:MyCommercialVehicle
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
xmlns:ext="urn:example.com:techniques:extension-example"
xmlns:exch="urn:example.com:techniques:exchange-example"
..>
   <!-- associated with base type -->
   <nc:VehicleBrand>Chevrolet</nc:VehicleBrand>
   <nc:VehicleDoorQuantity>6</nc:VehicleDoorQuantity>
   <nc:VehicleAxleQuantity>3</nc:VehicleAxleQuantity>
   <!-- associated with augmentation -->
   <ext:CommercialVehicleAugmentation>
        <ext:VehicleCompanyVIN> . . . </ext:VehicleCompanyVIN>
        <ext:VehicleOwningCompany> . . . </ext:VehicleOwningCompany>
        <ext:VehicleCompanyPurchaseDate> . . . </ext:VehicleCompanyPurchaseDate>
   </ext:CommercialVehicleAugmentation>
</exch:MyCommercialVehicle>
```

5.3.3 Using Element Substitution

NIEM uses several techniques from XML Schema to allow as-needed element substitutions for pre-existing NIEM properties and into pre-existing NIEM types. Element substitution techniques allow the substitution of new XML Schema elements, representing derived NIEM properties that can be used where the parent properties are expected.

There are three XML Schema techniques that support the NIEM use of element substitutions:

- use of substitution groups
- creation of abstract, type-less elements, and
- use of abstract elements in reference schemas.

Substitution groups allow elements to be derived from other elements. The attribute "`substitutionGroup`" appears on element definitions and indicates an element for which the element being defined may be substituted.

An abstract, type-less element represents a specific NIEM concept which can have multiple representations. Because the element has no type, it can carry any content, meaning that any kind of representation may be used, without restriction. However, only concrete elements may be used in XML instances, not abstract type-less elements which have no restriction on content. Therefore, the use of a substitution group, in conjunction with an abstract element, allows concrete elements to be substituted for the abstract element in XML instances.

Element substitution techniques are often used to implement managed lists (a/k/a code lists). This allows for the substitution of different code sets to represent the same enumerated concept. By providing a substitutable component in the schema, the schema designer has provided a placeholder for information that the IEPD creator must supply.

This defers the decision on which codes to use to a particular information exchange, rather than setting it in the schema.

For example, in the NIEM Core namespace, there is a NIEM type, `nc:JurisdictionType`, which contains references to geopolitical areas (country, state, county). An IEPD using this type may need the freedom to decide how to represent those references, as either text, or with an appropriate managed list of codes. Accordingly, the country, county and state references are setup as abstract, type-less components, to represent the geopolitical references conceptually, while not restricting the representation of those references.

This type is defined in the Core schema as follows:

XML schema example demonstrating element substitution in reference schema

```
<xsd:schema xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
targetNamespace="http://niem.gov/niem/niem-core/2.0"
…>

<xsd:complexType name="JurisdictionType">
    <xsd:complexContent>
        <xsd:extension base="s:ComplexObjectType">
             <xsd:sequence>
                  <xsd:element ref="nc:LocationCityName"
                       minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element ref="nc:LocationCountry"
                       minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element ref="nc:LocationCounty"
                       minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element ref="nc:LocationState"
                       minOccurs="0" maxOccurs="unbounded"/>
             </xsd:sequence>
         </xsd:extension>
     </xsd:complexContent>
</xsd:complexType>

<xsd:element name="LocationCountry" abstract="true"/>
<xsd:element name="LocationCounty" abstract="true"/>
<xsd:element name="LocationState" abstract="true"/>
```

For those IEPDs that wish to use a textual description of the geopolitical area, rather than a standard code table, an appropriate element is defined (`LocationCountryName`) in the Core namespace and placed in the substitution group `nc:LocationCountry`:

XML schema example demonstrating element substitution with substitution group (1)

```
<xsd:schema xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:i="http://niem.gov/niem/appinfo/2.0"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
targetNamespace="http://niem.gov/niem/niem-core/2.0"
…>

<xsd:element name="LocationCountryName" type="nc:ProperNameTextType"
substitutionGroup="nc:LocationCountry" nillable="true">
     <xsd:annotation>
         <xsd:appinfo>
             <i:Base i:name="LocationCountry"/>
          </xsd:appinfo>
     </xsd:annotation>
</xsd:element>
```

For those IEPDs that wish to use one of several managed lists available in NIEM for country codes, other (enumerated) elements are defined in the Core namespace and placed in the substitution group `LocationCountry`:

XML schema example demonstrating element substitution with substitution group (2)

```
<xsd:schema xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:i="http://niem.gov/niem/appinfo/2.0"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
xmlns:fips_10-4="http://niem.gov/niem/fips_10-4/2.0"
xmlns:iso_3166="http://niem.gov/niem/iso_3166/2.0"
targetNamespace="http://niem.gov/niem/niem-core/2.0"
…>

<xsd:element name="LocationCountryFIPS10-4Code" type="fips_10-4:CountryCodeType"
substitutionGroup="nc:LocationCountry" nillable="true">
    <xsd:annotation>
        <xsd:appinfo>
            <i:Base i:name="LocationCountry"/>
        </xsd:appinfo>
    </xsd:annotation>
</xsd:element>

<xsd:element name="LocationCountryISO3166Alpha2Code"
type="iso_3166:CountryAlpha2CodeType" substitutionGroup="nc:LocationCountry"
nillable="true">
    <xsd:annotation>
        <xsd:appinfo>
            <i:Base i:name="LocationCountry"/>
        </xsd:appinfo>
    </xsd:annotation>
</xsd:element>

<xsd:element name="LocationCountryISO3166Alpha3Code"
type="iso_3166:CountryAlpha3CodeType" substitutionGroup="nc:LocationCountry"
nillable="true">
    <xsd:annotation>
        <xsd:appinfo>
            <i:Base i:name="LocationCountry"/>
        </xsd:appinfo>
    </xsd:annotation>
</xsd:element>
```

```
<xsd:element name="LocationCountryISO3166NumericCode"
type="iso_3166:CountryNumericCodeType" substitutionGroup="nc:LocationCountry"
nillable="true">
    <xsd:annotation>
        <xsd:appinfo>
            <i:Base i:name="LocationCountry"/>
        </xsd:appinfo>
    </xsd:annotation>
</xsd:element>
```

As a result, an IEPD is allowed to use any of the following representations for the United States in its exchange documents:

XML instance examples demonstrating different country representations

```
<!-- as text -->
<nc:Jurisdiction xmlns:nc="http://niem.gov/niem/niem-core/2.0" >
   . . .
   <nc:LocationCountryName>The United States</nc:LocationCountryName>
   . . .
</nc:Jurisdiction>

<!-- as a FIPS10-4 code -->
<nc:Jurisdiction xmlns:nc="http://niem.gov/niem/niem-core/2.0" >
    . . .
   <nc:LocationCountryFIPS10-4Code>US</nc:LocationCountryFIPS10-4Code>
   . . .
</nc:Jurisdiction>

<!-- as an ISO1366 2 letter code -->
<nc:Jurisdiction xmlns:nc="http://niem.gov/niem/niem-core/2.0">
   . . .
   <nc:LocationCountryISO3166Alpha2Code>US</nc:LocationCountryISO3166Alpha2Code>
   . . .
</nc:Jurisdiction>

<!-- as an ISO1366 3 letter code -->
<nc:Jurisdiction xmlns:nc="http://niem.gov/niem/niem-core/2.0">
   . . .
   <nc:LocationCountryISO3166Alpha3Code>USA</nc:LocationCountryISO3166Alpha3Code>
   . . .
</nc:Jurisdiction>

<!-- as an ISO1366 numeric code -->
<nc:Jurisdiction xmlns:nc="http://niem.gov/niem/niem-core/2.0">
    . . .
  <nc:LocationCountryISO3166NumericCode>840</nc:LocationCountryISO3166NumericCode>
    . . .
</nc:Jurisdiction>
```

In addition, if an IEPD needs to use yet another managed list for countries not in NIEM, based on another code standard, it is free to do so. For example, suppose the exchange is limited to contain information about jurisdictions in South American countries and should use the 2-letter ISO3166 country codes for South America. In this case, the IEPD could create its own type for that managed list, as well as an element of that type. Then it can define that element (`LocationSouthAmericaCountryCode`) as a member of the `nc:LocationCountryISO3166Alpha2Code` substitution group.

Note that the substitution group need not be an abstract type – in this case, we are restricting a known concrete type, and using it in the substitution group.

The relevant fragment of this local managed list schema could look like this:

XML schema example demonstrating element substitution for concrete types

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
xmlns:exch=”urn:example.com:techniques:exchange-example”
targetNamespace=”urn: example.com:techniques:exchange-example”
…>

<xsd:element name="LocationSouthAmericaCountryCode"
type="exch:SouthAmericaCountryCodeType"
substitutionGroup="nc:LocationCountryISO3166Alpha2Code" nillable="true"/>
```

A schema containing the type definition for this local managed list schema would be included in the IEPD for this information exchange. As a result, IEP instances may use `exch:LocationSouthAmericaCountryCode` anywhere that the NIEM element `iso_3166:LocationCountryISO3166Alpha2Code` is expected.

It is worth pointing out in this example that an IEPD developer could also subset just the South American country codes (either manually or using the Schema Subset Generation Tool). However, the use of a named substitution group makes explicit the concept that South American country codes should be used and no others.

5.4 Choosing an Extension Technique

This section discusses some decision points for selecting an appropriate NIEM extension technique, and offers some advice on which extension technique is appropriate for some common scenarios.

5.4.1 Create New NIEM Types or Add to Existing Types?

Creation of new NIEM types through specialization or composition creates a new type of data object. It is not appropriate to create a NIEM type just to define additional properties of the base type, as that would hinder reuse. When new properties must be added to a base type, use the augmentation approach rather than creating a new NIEM type. Otherwise, when required and appropriate, use specialization or composition to create the new NIEM type.

5.4.2 NIEM Type Composition or Specialization?

Creation of new NIEM types with the composition technique allows for new concepts to be created out of smaller, possibly unrelated, building blocks. Creation of new types with the specialization technique allows refinement of an existing concept. Specialization allows dynamic type substitution within an instance. An element of the new derived type can be used anywhere an element of the base class was expected.

5.4.3 NIEM Type Specialization or Augmentation?

Specialized types must be carefully designed to adhere to the rules for types being permanent and mutually exclusive. In many cases, using augmentations or roles (as will be discussed in Section 6) are good alternatives when the rules for specialized types cannot be satisfied.

5.4.4 Augmentation or Metadata?

The sort of information that is to be added to the domain model determines whether to use augmentation or metadata. If the additional information describes characteristics of the data itself, such as who gathered the data, when it was gathered, then a metadata block should be added. Metadata blocks should be used to contain only data about the data. Otherwise, an augmentation container should be used to add the extra information, which represents data about objects and relationships, not data about the data itself.

5.4.5 Content or Reference Elements?

The choice of content or reference elements to represent NIEM properties depends on the information exchange being modeled. For example, a NIEM property representing a birth date would probably best be represented as a content element. Even though lots of people could have the same birth date, and the date could be used for many purposes, it is generally easier to just use copies of the date, when it is used in multiple places.

By comparison, a NIEM property that represents a person will often take the form of a reference element. As the definition of a person may be complicated, it makes little sense to copy its value when it is needed in multiple places. In many cases, the person may exist outside its use in the NIEM property and may appear in several other NIEM properties, within other NIEM types. Therefore, it may make sense for the person to be a standalone entity that can be reused through properties represented as reference elements.

In general, when the data is simple, or will only be used in one particular context or property, a content element representation is a good choice. If the data is complex, or is likely to appear in multiple contexts, a reference element representation should be used. The large, complex properties in the NIEM Core tend to use reference elements, while the small, simpler properties in NIEM tend to use content elements.

In many cases, an IEPD developer must decide whether a NIEM property can be used either as a reference or a content element. However, not all properties have both representations. There are some NIEM properties which are constrained to be used as reference elements only, whereas other NIEM properties are constrained to be used as content elements only.

6 Standard NIEM Extensions

In this section, we cover three common NIEM extensions that practitioners may use in their information exchanges. These extensions are common enough that there are special structures in NIEM to facilitate their addition.

6.1 Roles

A *role* represents a particular context or activity for a data object. A role may be specific to time, incident, employment, or other aspects of an activity or context. The object to which the role applies is called the *base object*.

We do not want to create NIEM role types for every possible use of a particular NIEM type. Instead, we create them as the situation warrants their use. Where the role has specific data associated with it, and the data has its own life cycle, we create a new NIEM type to capture that data. When the role has no data specific to the context or activity tied to the base object, no new NIEM type needs to be created to represent the role.

For example, one person might pick up an object, like a tire iron, and hit another person with it. In this case, the tire iron will take on the role of “Weapon” the wielder of the tire iron the role of “CriminalSuspect", and the person hit with the tire iron the role of "Victim". On the other hand, if a person picks up the tire iron and steals it, rather than hitting someone with it, the tire iron will take on a different role. Now the role of the tire iron is "StolenProperty" and the person who owned the tire iron is assigned the role of “Victim”. In these situations, where the role has specific data associated with it, and the data has its own life cycle, we create a new NIEM type.

In other situations, a role may or may not have data associated with it. For example, a vehicle may be used as a getaway car from a robbery. If we only know that a robbery had a getaway car associated with it, then we could define a “RobberyType” which has a property “GetawayCar” that is of “VehicleType”. There is no need to create a new NIEM type to represent the getaway car in the robbery, so long as the existing NIEM type “VehicleType” contains all the necessary information properties.

But now, suppose we want to record information about the use of the getaway car (e.g. driver, violations, max speed, and origination point), expanding the data we gather about the robbery. We would want to extend the data model to create a new NIEM role type to store that information about the use of the vehicle in that activity, “RoleOfVehicle”. To do so, we define “RobberyType” with a property “GetawayCar” that is of this new NIEM role type “RoleOfVehicle.”

Any single data object instance may have multiple roles for a given context or activity. For example, a single person may take the role of "ArrestingOfficial", "Victim", and "Witness" in the same instance.

In XML Schema, a new role is represented as a XML Schema complex type. The type should contain a particular "RoleOf<BaseObject>” XML element. This element represents the base object to which this role applies. Several “RoleOf<BaseObject>” properties are provided in the NIEM data model, but others may be added as needed. The XML Schema type also includes any other elements representing the other data associated with the role.

Here is an example describing the role type representing a weapon:

XML Schema fragment for a weapon, a role of an object.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
targetNamespace=”http://niem.gov/niem/niem-core/2.0”
…>

<xsd:complexType name="WeaponType">
  <xsd:sequence>
    <xsd:element ref="nc:RoleOfItemReference" ... />
    <xsd:element ref="nc:WeaponUserReference" ... />
    <xsd:element ref="nc:WeaponInvolvedInActivityReference" .../>
    <xsd:element ref="nc:WeaponUsageText" ... />
  </xsd:sequence>
</xsd:complexType>
```

The element "`nc:RoleOfItemReference`" refers to the object used as a weapon. In a corresponding XML instance, the value of that element is a reference to the tire iron that was used as a weapon:

XML instance example of a weapon, a role of an object

```
<nc:Property s:id="ExhibitA" xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:nc="http://niem.gov/niem/niem-core/2.0">
   <nc:PropertyDescriptionText>Tire iron</nc:PropertyDescriptionText>
</nc:Property>

<nc:Person s:id="TMD051395" xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:nc="http://niem.gov/niem/niem-core/2.0">
   <nc:PersonFullName>Rod Rage</nc:PersonFullName>
</nc:Person>

<nc:Weapon xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:nc="http://niem.gov/niem/niem-core/2.0">
  <nc:RoleOfItemReference s:ref="ExhibitA"/>
  <nc:WeaponUserReference s:ref="RR051395"/>
  <nc:WeaponUsageText>Swung like a club</nc:WeaponUsageText>
</nc:Weapon>
```

This represents a weapon, which is a role taken by object "ExhibitA", the tire iron; and that weapon was used by person "RR051395” to commit the crime.

6.2 Associations

An *association* represents a specific relationship between objects. Associations are used when a simple NIEM property is insufficient to model the relationship clearly and when properties of the relationship exist that are not attributable to the objects being related.

For example, a parent-child relationship could be represented as simple properties:

- The parent object has a "child" property. The value of the property is the child of the parent.
- The child object has a "parent" property. The value of the property is the parent of the child.

These two options create concerns:

- For a given relationship, which method do we use? Do we link from the child, or from the parent, or both?
- If these are represented as content elements, what about the circular reference?
- Where do we put additional information about the relationship?

To resolve these issues, we create an *association type*. An association type is a NIEM type that represents the relationship between the parent and the child, and which captures the additional information about the relationship, not the objects involved in the relationship.

There are two types of properties in an association type

- Characteristics, describing the context and particulars of the relationship
- Participants, describing the objects involved in the relationship.

In XML Schema, type composition is used to create the XML Schema complex type for an association. The new NIEM association type contains reference elements that refer to the objects (participants) it associates. Characteristics of the relationship should be maintained in the NIEM association type, not in the objects it associates.

For example, here is a possible marriage association between two people:

XML schema fragment showing a marriage association

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
xmlns:i="http://niem.gov/niem/appinfo/2.0"
xmlns:exch=”urn:example.com:techniques:exchange-example”
targetNamespace=”urn:example.com:techniques:exchange-example”
…>

<xsd:complexType name="MarriageAssociationType">
   <xsd:sequence>
      <xsd:element ref="exch:SpouseReference" minOccurs="2" maxOccurs="2" />
      <xsd:element name="nc:MarriageDate" type="nc:DateType"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="SpouseReference" type="s:ReferenceType">
    <xsd:annotation>
        <xsd:appinfo>
            <i:ReferenceTarget i:name="nc:PersonType"/>
        </xsd:appinfo>
    </xsd:annotation>
</xsd:element>

<xsd:element name="MarriageAssociation" type="exch:MarriageAssociationType"
nillable="true"/>
```

And here is an instance of that marriage association:

XML instance fragment showing a marriage association

```
<nc:Person s:id="GPB" xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:nc="http://niem.gov/niem/niem-core/2.0" >
    <nc:PersonName>
        <nc:PersonFullName>George P. Burdell</nc:PersonFullName>
    </nc:PersonName>
</nc:Person>

<nc:Person s:id="GZB" xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"  >
    <nc:PersonName>
        <nc:PersonFullName>Georgia Z. Burdell</nc:PersonFullName>
    </nc:PersonName>
</nc:Person>

<exch:MarriageAssociation xmlns:s="http://niem.gov/niem/structures/2.0"
xmlns:exch="urn:example.com:techniques:exchange-example">
     <exch:SpouseReference s:ref="GPB" />
     <exch:SpouseReference s:ref="GZP" />
     <exch:MarriageDate>1989-04-01</exch:MarriageDate>
</exch:MarriageAssociation>
```

Association types may be reused in multiple contexts – there is no need to define a new association type for a particular association, if an existing type represents the relationship accurately. For example, the Core namespace contains an association type `nc:PersonLocationAssociationType`, which is used in the following associations in the NIEM Core:

XML schema fragment showing shared association types

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nc="http://niem.gov/niem/niem-core/2.0"
targetNamespace="http://niem.gov/niem/niem-core/2.0"
…>

<xsd:element name="LocationNeighboringPersonAssociation"
type="nc:PersonLocationAssociationType" nillable="true"/>

<xsd:element name="PersonCurrentLocationAssociation"
type="nc:PersonLocationAssociationType" nillable="true"/>

<xsd:element name="PersonDetainmentLocationAssociation"
type="nc:PersonLocationAssociationType" nillable="true"/>

<xsd:element name="PersonEmploymentLocationAssociation"
type="nc:PersonLocationAssociationType" nillable="true"/>

<xsd:element name="PersonKnownPreviousLocationAssociation"
type="nc:PersonLocationAssociationType" nillable="true"/>

<xsd:element name="PersonLastSeenLocationAssociation"
type="nc:PersonLocationAssociationType" nillable="true"/>

<xsd:element name="PersonLocationAssociation"
type="nc:PersonLocationAssociationType" nillable="true"/>
```

Associations should be created between objects only if the objects in the relationship meet the following criteria:

- The related objects are peers, meaning one is not logically a subpart of the other. Peers have their own set of characteristic properties independently of one another.

- Objects can exist outside of the relationship with another object. In other words, none of the objects lose meaning if separated from the others. (Note that the very simple properties, such as Date, usually tend to lose semantic meaning when taken out of context).

6.3 Adapting External Standards

In addition to adding new NIEM types and properties to NIEM, it is possible to adapt existing external (non-NIEM) namespaces for use in the NIEM framework. This allows the use of external standards within NIEM IEPDs, without requiring that the external standards themselves be NIEM-conformant. The intent here is to allow use of external standard components exactly as they were defined.

The basic technique for adapting external standards is to wrap the non-conformant XML Schema types and elements in NIEM-conformant components, maintained in a NIEM-conformant schema. These wrapper components effectively shadow as much or as little of the external standard as deemed appropriate, depending on how the wrapper components are designed. This allows the use of the standard within the NIEM framework at any granularity, while preserving the semantics and original structure of the external standard.

External standards do NOT need to be remodeled or placed directly into NIEM – instead their adapting components can be used within NIEM-compliant IEPDs without requiring translation on the part of the IEPD designer. However, profiles of the external standards must be included in the IEPD package.

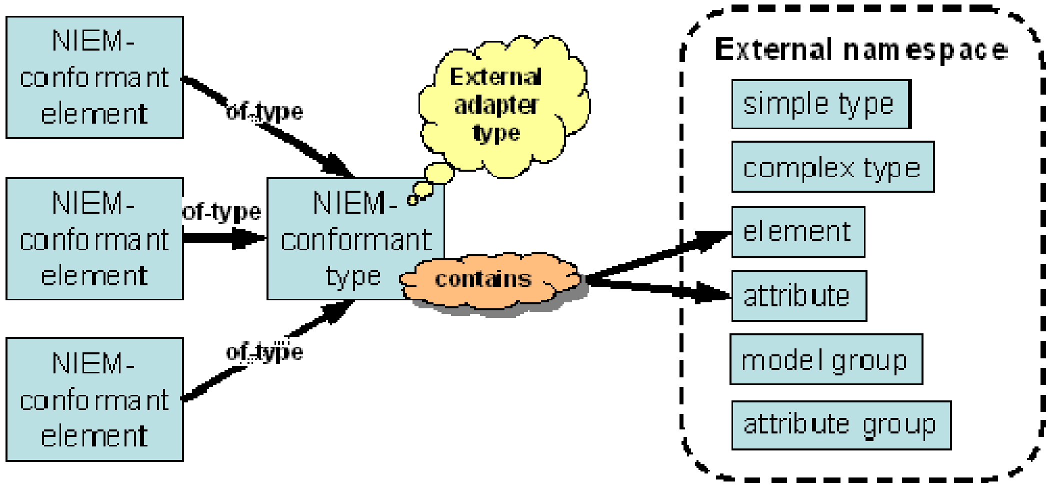
The main construct available in NIEM 2.0 for use with external standards are *external adapter types*, necessary when the external standard provides reusable elements defined as non-NIEM-conforming types.

The external adapter type is a NIEM-conformant type that contains

- attributes from external namespaces
- elements from external namespaces

The subparts of that adapter type should correspond to a semantically meaningful concept – in other words, an adapter type should wrap concepts, not just unrelated external content. The adapter type may reference content from more than one external namespace, but all content must be from external namespaces.

The picture below shows the relationship between the NIEM wrappers and the external content they adapt:



There are some special importing and packaging requirements for an IEPD that accesses external adapter types. An IEPD that uses an external namespace through adapter components will require the import of both a schema that contains the NIEM-conformant components (adapter types) and the non-NIEM conformant external schemas. All the relevant schemas must be included in the IEPD. But aside from these requirements, external adapter types can be used in an IEPD just like standard NIEM types – nothing special is required for designing schemas or instances that use external adapter types.

Here is an example of an external adapter type from the Geospatial external standard in NIEM 2.0. The adapter type is `geo:SingleSiteLandmarkAddressType`. Note that the appinfo information states that this is an external adapter type.

Adapter schema fragment describing an external adapter type

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:i="http://niem.gov/niem/appinfo/2.0"
xmlns:addr="http://niem.gov/niem/external/urisa-street-address/draft-0.2.0/dhs-
gmo/1.0.0"
xmlns:geo="http://niem.gov/niem/geospatial/2.0"
targetNamespace=”http://niem.gov/niem/geospatial/2.0”
…>

<xsd:complexType name="SingleSiteLandmarkAddressType">
    <xsd:annotation>
        <xsd:appinfo>
            <i:Base i:namespace="http://niem.gov/niem/structures/2.0"
             i:name="Object"/>
            <i:ExternalAdapterTypeIndicator>true</i:ExternalAdapterTypeIndicator>
        </xsd:appinfo>
    </xsd:annotation>
    <xsd:complexContent>
        <xsd:extension base="s:ComplexObjectType">
            <xsd:sequence>
                 <xsd:element ref="addr:SingleSiteLandmarkAddress"
                  maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Here is the external content, from the URISA Street address namespace, adapted by this particular external adapter type:

External schema fragment for external element being adapted

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:addr="http://niem.gov/niem/external/urisa-street-address/draft-0.2.0/dhs-
gmo/1.0.0"
targetNamespace="http://niem.gov/niem/external/urisa-street-address/draft-
0.2.0/dhs-gmo/1.0.0"
 …>

<xsd:complexType name="SingleSiteLandmarkAddress_type">
    <xsd:sequence>
        <xsd:element name="LandmarkName" type="addr:LandmarkName_type"/>
        <xsd:element name="CompleteOccupancyIdentifier”
            type="addr:CompleteOccupancyIdentifier_type" minOccurs="0"/>
        <xsd:element name="PlaceName" type="addr:PlaceName_type"/>
        <xsd:element name="StateName" type="addr:StateName_type"/>
        <xsd:element name="ZipCode" type="addr:ZipCode_type"/>
        <xsd:element name="ZipPlus4" type="addr:ZipPlus4_type" minOccurs="0"/>
        <xsd:element name="NationName" type="addr:NationName_type" minOccurs="0"/>
        <xsd:element name="AddressAttributes"
            type="addr:AddressAttributes_type" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="action" type="addr:Action_type" use="optional"/>
</xsd:complexType>

<xsd:element name="SingleSiteLandmarkAddress"
     <type="addr:SingleSiteLandmarkAddress_type"/>
```

Finally, here is a fragment of an XML instance that uses this external adapter type and the external content it adapts:

XML instance example using adapted external standard

```
<geo:SingleSiteLandmarkAddress xmlns:geo="http://niem.gov/niem/geospatial/2.0"
xmlns:addr="http://niem.gov/niem/external/urisa-street-address/draft-0.2.0/dhs-
gmo/1.0.0">
    <addr:SingleSiteLandmarkAddress>
        <addr:LandmarkName>Statue of Liberty</addr:LandmarkName>
             <addr:PlaceName>
                  <addr:MunicipalJurisdiction>New York</addr:MunicipalJurisdiction>
             </addr:PlaceName>
        <addr:StateName>NY</addr:StateName>
    <addr:ZipCode>10004</addr:ZipCode>
    </addr:SingleSiteLandmarkAddress>
</geo:SingleSiteLandmarkAddress>
```

7 Conclusion

This paper has outlined the techniques that are available to information practitioners to extend the NIEM to handle new information exchanges. By proper use of these techniques within the IEPD development lifecycle, domain modelers have the ability to leverage the extensive constructs and capabilities of the NIEM data model, while extending it as necessary, to meet the needs of their particular information exchanges.

Appendix A An IEPD Example

This paper is supplemented with an IEPD example called “Commercial Vehicle Collision” that illustrates extending types using type augmentation in a couple of different ways as well as the use of associations. The IEPD was constructed without a set of valid requirements. This example IEPD is strictly for illustrative or training purposes. It bears no relationship to any official IEPD from any agency or organization.

The IEPD contains most but not necessarily all of the files required by the NIEM IEPD Specifications (http://www.niem.gov//files/NIEM_IEPD_Requirements_v2_1.pdf). It contains the following files:

- catalog.html
- exchangeSchema.xsd
- extensionSchema.xsd
- metadata.xml
- sampleInstance.xml
- subset.zip
- wantlist.xml

To view the files, unzip the package CommercialVehicleCollision.zip and open catalog.html in a browser. Once extracted from the archive (zip file), all files can be accessed through the catalog.

Appendix B NIEM 2.0 Release Directory Tree

The following is an alphabetical listing of the directory tree for the NIEM 2.0 Release of reference schemas. These directories are all under the *niem* subdirectory.

```
niem
│
├───ansi-nist
│   └───2.0
│           ansi-nist.xsd
│
├───ansi_d20
│   └───2.0
│           ansi_d20.xsd
│
├───apco
│   └───2.0
│           apco.xsd
│
├───appinfo
│   └───2.0
│           appinfo.xsd
│
├───atf
│   └───2.0
│           atf.xsd
│
├───census
│   └───2.0
│           census.xsd
│
├───dea
│   └───2.0
│           dea.xsd
│
├───dod_jcs-pub2.0-misc
│   └───2.0
│           dod_jcs-pub2.0-misc.xsd
│
├───domains
│   ├───emergencyManagement
│   │   └───2.0
│   │           emergencyManagement.xsd
│   │
│   ├───immigration
│   │   └───2.0
│   │           immigration.xsd
│   │
│   ├───infrastructureProtection
│   │   └───2.0
│   │           infrastructureProtection.xsd
│   │
│   ├───intelligence
│   │   └───2.0
│   │           intelligence.xsd
│   │
│   ├───internationalTrade
│   │   └───2.0
│   │           internationalTrade.xsd
│   │
│   ├───jxdm
│   │   └───4.0
│   │           jxdm.xsd
│   │
│   └───screening
│       └───2.0
│               screening.xsd
│
├───edxl
│   └───2.0
│           edxl.xsd
│
├───edxl-cap
│   └───2.0
│           edxl-cap.xsd
│
├───edxl-de
│   └───2.0
│           edxl-de.xsd
│
├───external
│   ├───cap
│   │   └───1.1
│   │           cap.xsd
│   │
│   ├───de
│   │   └───1.0
│   │           de.xsd
│   │
│   ├───dhs-gmo
│   │   └───AS
│   │       ├───mobileObject
│   │       │   └───1.0.0
│   │       │           mobileObject.xsd
│   │       │
│   │       └───multiModalRoute
│   │           └───1.0.0
│   │                   multiModalRoute.xsd
│   │
│   ├───iai-ifc
│   │   └───rc2
│   │       └───dhs-gmo
│   │           └───1.0.0
│   │                   IFC2X2_FINAL.xsd
│   │
│   ├───iso-10303-step
│   │   └───2
│   │       └───dhs-gmo
│   │           └───1.0.0
│   │                   configuration.xsd
│   │                   ex.xsd
│   │
│   ├───iso-19139-gmd
│   │   └───draft-0.1
│   │       ├───gco
│   │       │   └───dhs-gmo
│   │       │       └───1.0.0
│   │       │               basicTypes.xsd
│   │       │               gco.xsd
│   │       │               gcoBase.xsd
│   │       │
│   │       ├───gmd
│   │       │   └───dhs-gmo
│   │       │       └───1.0.0
│   │       │               applicationSchema.xsd
│   │       │               citation.xsd
│   │       │               constraints.xsd
│   │       │               content.xsd
│   │       │               dataQuality.xsd
│   │       │               distribution.xsd
│   │       │               extent.xsd
│   │       │               freeText.xsd
│   │       │               gmd.xsd
│   │       │               identification.xsd
│   │       │               maintenance.xsd
│   │       │               metadataApplication.xsd
│   │       │               metadataEntity.xsd
│   │       │               metadataExtension.xsd
│   │       │               portrayalCatalogue.xsd
│   │       │               referenceSystem.xsd
│   │       │               spatialRepresentation.xsd
│   │       │
│   │       ├───gmx
│   │       │   └───dhs-gmo
│   │       │       └───1.0.0
│   │       │               catalogues.xsd
│   │       │               codelistItem.xsd
│   │       │               crsItem.xsd
│   │       │               extendedTypes.xsd
│   │       │               gmx.xsd
│   │       │               gmxUsage.xsd
│   │       │               uomItem.xsd
│   │       │
│   │       ├───gsr
│   │       │   └───dhs-gmo
│   │       │       └───1.0.0
│   │       │               gsr.xsd
│   │       │               spatialReferencing.xsd
│   │       │
│   │       ├───gss
│   │       │   └───dhs-gmo
│   │       │       └───1.0.0
│   │       │               geometry.xsd
│   │       │               gss.xsd
│   │       │
│   │       └───gts
│   │           └───dhs-gmo
│   │               └───1.0.0
│   │                       gts.xsd
│   │                       temporalObjects.xsd
│   │
│   ├───landxml
│   │   └───1.1
│   │           LandXML-1.1.xsd
│   │
│   ├───ogc-context
│   │   └───1.1.0
│   │       └───dhs-gmo
│   │           └───1.0.0
│   │                   context.xsd
│   │
│   ├───ogc-filter
│   │   └───1.1.0
│   │       └───dhs-gmo
│   │           └───1.0.0
│   │                   filter.xsd
│   │
│   ├───ogc-gml
│   │   └───3.1.1
│   │       └───dhs-gmo
│   │           └───1.0.0
│   │                   gml.xsd
│   │
│   ├───ogc-observation
│   │   └───draft-0.14.5
│   │       ├───om
│   │       │   └───dhs-gmo
│   │       │       └───1.0.0
│   │       │               commonObservation.xsd
│   │       │               event.xsd
│   │       │               observation.xsd
│   │       │               observationSpecializations.xsd
│   │       │               om.xsd
│   │       │               procedure.xsd
│   │       │               procedureSpecializations.xsd
│   │       │
│   │       ├───st
│   │       │   └───dhs-gmo
│   │       │       └───1.0.0
│   │       │               simpleTypeDerivation.xsd
│   │       │
│   │       └───swe
│   │           └───dhs-gmo
│   │               └───1.0.0
│   │                       discreteCoverage.xsd
│   │                       phenomenon.xsd
│   │                       record.xsd
│   │                       recordType.xsd
│   │                       swe.xsd
│   │                       SWE_basicTypes.xsd
│   │                       temporalAggregates.xsd
│   │
│   ├───ogc-openls
│   │   └───1.1.0
│   │       └───dhs-gmo
│   │           └───1.0.0
│   │                   ols.xsd
│   │
│   ├───ogc-ows
│   │   └───1.0.0
│   │       └───dhs-gmo
│   │           └───1.0.0
│   │                   ows.xsd
│   │
│   ├───ogc-sld
│   │   └───1.0.20
│   │       └───dhs-gmo
│   │           └───1.0.0
│   │                   sld.xsd
│   │
│   ├───ogc-swe-common
│   │   └───1.0.0
│   │       └───dhs-gmo
│   │           └───1.0.0
│   │                   data.xsd
│   │                   parameters.xsd
│   │                   positionData.xsd
│   │                   sweCommon.xsd
│   │
│   ├───ogc-wfs
│   │   └───1.1.0
│   │       └───dhs-gmo
│   │           └───1.0.0
│   │                   wfs.xsd
│   │
│   ├───urisa-street-address
│   │   └───draft-0.2.0
│   │       └───dhs-gmo
│   │           └───1.0.0
│   │                   StreetAddressDataStandard.xsd
│   │
│   ├───w3c-xlink
│   │   └───1.0
│   │       └───dhs-gmo
│   │           └───1.0.0
│   │                   xlinks.xsd
│   │
│   └───w3c-xml
│       └───1998
│               xml.xsd
│
├───fbi
│   └───2.0
│           fbi.xsd
│
├───fips_10-4
│   └───2.0
│           fips_10-4.xsd
│
├───fips_5-2
│   └───2.0
│           fips_5-2.xsd
│
├───fips_6-4
```

```
│   └───2.0
│           fips_6-4.xsd
│
├───geospatial
│   └───2.0
│           geospatial.xsd
│
├───have
│   └───2.0
│           have.xsd
│
├───hazmat
│   └───2.0
│           hazmat.xsd
│
├───iso_3166
│   └───2.0
│           iso_3166.xsd
│
├───iso_4217
│   └───2.0
│           iso_4217.xsd
│
├───iso_639-3
│   └───2.0
│           iso_639-3.xsd
│
├───itis
│   └───2.0
│           itis.xsd
│
├───lasd
│   └───2.0
│           lasd.xsd
│
├───mmucc_2
│   └───2.0
│           mmucc_2.xsd
│
├───mn_offense
│   └───2.0
│           mn_offense.xsd
│
├───nga
│   └───2.0
│           nga.xsd
│
├───niem-core
│   └───2.0
│           niem-core.xsd
│
├───nlets
│   └───2.0
│           nlets.xsd
│
├───nonauthoritative-code
│   └───2.0
│           nonauthoritative-code.xsd
│
├───post-canada
│   └───2.0
│           post-canada.xsd
│
├───proxy
│   └───xsd
│       └───2.0
│               xsd.xsd
│
├───sar
│   └───2.0
│           sar.xsd
│
├───structures
│   └───2.0
│           structures.xsd
│
├───twpdes
│   └───2.0
│           twpdes.xsd
│
├───ucr
│   └───2.0
│           ucr.xsd
│
├───unece_rec20-misc
│   └───2.0
│           unece_rec20-misc.xsd
│
├───usps_states
│   └───2.0
│           usps_states.xsd
│
└───ut_offender-tracking-misc
    └───2.0
            ut_offender-tracking-misc.xsd

```

Appendix C NIEM 2.0 Code Table Namespaces

The following is an alphabetical listing of namespace prefixes and descriptions for all the code table namespaces in NIEM 2.0:

- ansi_d20 - Motor vehicle administration codes from ANSI D20, the Data Dictionary for Traffic Record Systems, maintained by AAMVA, the American Association of Motor Vehicle Administrators.
- ansi-nist - ANSI/NIST Fingerprint and Biometric standard.
- apco - Association of Public-Safety Communications Officials (APCO) - International, Inc.
- atf - Bureau of Alcohol, Tobacco, and Firearms.
- can - Province codes for Canada.
- census - Employment codes from the U.S. Census Bureau.
- dea - Drug Enforcement Administration.
- dod_jcs-pub2.0-misc - Intelligence discipline codes from the U.S. Department of Defense (DoD) Joint Publication 2.01.
- edxl - Emergency Data Exchange Language.
- fbi - FBI code lists for National Crime and Information Center (NCIC-2000), National Incident-Based Reporting System (NIBRS), and National Law Enforcement Data Exchange (N-DEx)..
- fips_10-4 - Countries, dependencies, areas of special sovereignty, and their principal administrative divisions from the Federal Information Processing Standards (FIPS) 10-4.
- fips_5-2 - Codes for the identification of the states, the District of Columbia and the outlying areas of the U.S., and associated areas from the Federal Information Processing Standards (FIPS) 5-2.
- fips_6-4 - Counties and equivalent entities of the U.S., its possessions, and associated areas from the Federal Information Processing Standards (FIPS) 6-4.
- have - EDXL Hospital AVailability Exchange (HAVE).
- hazmat - Pipeline and Hazardous Materials Safety Administration - Office of Hazardous Materials Safety.
- iso_3166 - Codes for the representation of names of countries and their subdivisions from the International Organization for Standardization (ISO) 3166-1:1997.
- iso_4217 - Codes for the representation of currencies and funds from the International Organization for Standardization (ISO) 4217:2001.

- iso_639-3 - Codes for the representation of names of languages - Part 3: Alpha-3 code for comprehensive coverage of languages.
- itis - Integrated Transportation Information System.
- lasd - Los Angeles County Sheriff’s Department.
- mmucc_2 - Model Minimum Uniform Crash Criteria.
- mn_off - Statute and offense codes from the state of Minnesota.
- nga - National Geospatial Agency.
- nlets - NLETS - The International Justice and Public Safety Information Sharing Network
- nonauth - Non-authoritative codes for the direction of a person's pose in an image.
- sar - Suspicious Activity Reporting.
- twpdes - Terrorist Watchlist Person Data Exchange Standard.
- ucr - Crime reporting codes from Uniform Crime Reporting.
- unece - Miscellaneous unit of measure codes from the United Nations Economic Commission for Europe Recommendation No. 20, Codes for Units of Measure used in International Trade.
- usps - U.S. state and possession abbreviations from the U.S. Postal Service (USPS).
- ut_offender - Plea and military discharge codes from the Utah Offender Tracking Database, version 2.03.

NIEM™