October 31, 2008

NIEM NAMING AND DESIGN RULES

VERSION 1.3

URI: http://reference.niem.gov/niem/specification/naming-and-design-rules/1.3/

NATIONAL INFORMATION EXCHANGE MODEL | NIEM.gov

⟨NIEM⟩

Editors:

Webb Roberts, Georgia Tech Research Institute

Susan Liebeskind, Georgia Tech Research Institute

Mark Kindl, Georgia Tech Research Institute

Abstract:

This document specifies the data model, XML components, and XML data for use with the National Information Exchange Model (NIEM) version 2.0.

Status:

This document is a specification for NIEM-conformant XML Schema documents, components, and instances. It represents the design that has evolved from the collaborative work of the NIEM Business Architecture Committee (NBAC) and the NIEM Technical Architecture Committee (NTAC) and their predecessors.

This specification is a product of the NIEM Program Management Office (PMO).

Send comments on this specification via email to `nisshelp@ijis.org`.

Record of Changes

No.	Date	Reference: All, Page, Table, Figure, Paragraph	A = Add. M = Mod. D = Del.	Revised By	Change Description
1.0	06/08/2007	All	A	Webb Roberts, Susan Liebeskind, Mark Kindl	Initial version Internal draft
1.1	06/27/2007	All	M	Webb Roberts, Susan Liebeskind, Mark Kindl	Internal draft
1.2	08/07/2007	All	M	Webb Roberts, Susan Liebeskind, Mark Kindl	Public draft
1.3	10/31/2008	All	M	NTAC, Webb Roberts, Mark Kindl	Final

Contents

Figures

Tables

1 Introduction

This Naming and Design Rules (NDR) document specifies XML Schema documents for use with the National Information Exchange Model (NIEM). NIEM is an information sharing framework based on the World Wide Web Consortium (W3C) Extensible Markup Language (XML) Schema standard. In February 2005, the U.S. Departments of Justice (DOJ) and Homeland Security (DHS) signed a cooperative agreement to jointly develop NIEM by leveraging and expanding the Global Justice XML Data Model (GJXDM) into multiple domains. NIEM is a result of a combined government and industry effort to improve information interoperability and exchange within the United States at federal, state, tribal, and local levels of government.

NIEM specifies a set of reusable information components for defining standard information exchange messages, transactions, and documents on a large scale: across multiple communities of interest and lines of business. These reusable components are rendered in XML Schema documents as type, element, and attribute definitions that comply with the W3C XML Schema specification. The resulting reference schemas are available to government practitioners and developers at `http://niem.gov/`.

The W3C XML Schema standard enables information interoperability and sharing by providing a common language for describing data precisely. The constructs it defines are basic metadata building blocks — baseline data types and structural components. Users employ these building blocks to describe their own domain-oriented data semantics and structures, as well as structures for specific information exchanges and components for reuse across multiple information exchanges. Rules that profile allowable XML Schema constructs and describe how to use them help ensure that those components are consistent and reusable.

This document specifies principles and enforceable rules for NIEM data components and schemas. Schemas and components that obey the rules set forth here are considered to be **NIEM-conformant**.

1.1 Scope

This document was developed to specify NIEM 2.0. Later releases of NIEM may be specified by later versions of this document. The document covers the following issues in depth:

- The underlying NIEM data model
- Guiding principles behind the design of NIEM
- Rules for using XML Schema constructs in NIEM
- Rules for modeling and structuring NIEM-conformant schemas
- Rules for creating NIEM-conformant instances
- Rules for naming NIEM components
- Rules for extending NIEM-conformant components

This document does NOT address the following:

- A formal definition of the NIEM data model.

  Such a definition would focus on the Resource Definition Framework (RDF) and concepts not strictly required for interoperability. This document instead focuses on definition of schemas that work with the data model, to ensure translatability and interoperability.

- A detailed discussion of NIEM architecture and schema versioning.

  Such rules will be addressed in **[ARCH]**.

- The artifacts of the NIEM information exchange process.

  The artifacts of the NIEM information exchange process are discussed in **[IEPD]**.

This document is intended as a technical specification. It is not intended to be a tutorial or a user guide. A brief NIEM overview is provided in Appendix A: NIEM Overview.

1.2 Audience

This document targets practitioners and developers who employ NIEM for information exchange and interoperability. Such information exchanges may be between or within organizations. The NIEM reference schemas provide system implementers much content on which to build specific exchanges. However, there is a need for extended and additional content. The purpose of this document is to define the rules for such new content so that it will be consistent with the NIEM reference schemas. These rules are intended to establish and, more important, enforce a degree of standardization on a national level.

1.3 Document Conventions

This document uses formatting and syntactic conventions to clarify meaning and avoid ambiguity.

1.3.1 Document References

This document relies on references to many outside documents. Such references are noted by bold, bracketed inline terms. For example, a reference to RFC 2119 is shown as **[RFC2119]**. All reference documents are recorded in Appendix D: References.

1.3.2 Normative and Informative Content

This document includes a variety of content. Some content is normative (binding and enforceable in implementations), while other content is informative (explanatory, but not part of the NIEM specification). In general, the informative material appears as supporting text and specific rationales for the normative material.

Conventions used within this document include:

[Definition: *<term>*]

A formal definition of a term associated with NIEM.

Definitions are normative.

[Principle <*number*>]

A guiding principle for NIEM.

The principles represent the requirements, concepts, and goals that have helped shape the NIEM. Principles are informative, not normative, but act as the basis on which the rules are defined.

Accompanying each principle is a short discussion section that justifies the application of the principle to NIEM design.

Principles are numbered in the order in which they appear in the document.

[Rule <section>-<number>] (<applicability>)

An enforceable rule for NIEM.

Rules state specific requirements on artifacts, such as schemas and instances. Most rules apply to conformant schemas, while others apply to instances. The rules are normative.

Rules are stated using both XML InfoSet terminology (elements and attributes) and XML Schema terminology (schema components). The choice of terminology is driven by which standard best expresses the rule. Certain concepts are more clearly expressed using XML InfoSet information items, others using the XML Schema data model; still others are best expressed using a combination of terminology drawn from both standards.

Rules have rationales that justify the need for the rule. For clarity, there may be multiple rules that have the same rationale.

Rules and supporting text may use Extended Backus-Naur Form (EBNF) notation as defined by **[XML]**.

Rules are numbered according to the section in which they appear and the order in which they appear within that section. For example, Rule 5-1 is the first rule in Section 5.

Each rule is accompanied by a description of its applicability. This identifies the type of schema to which the rule applies or indicates whether the rule is applicable to XML documents or element information items. Each entry in the list is a code from Table 2-1: Codes Representing Conformance Targets. If a code appears in the applicability list for a rule, then the rule applies to the corresponding conformance target. The conformance targets are defined in Section 2, NIEM Conformance.

1.3.3 Formatting

In addition to special formatting for definitions, principles, and rules, this document uses consistent formatting to identify NIEM components.

`Courier`: All words appearing in `Courier` font are values, objects, keywords, or literal XML text.

Italics: All words appearing in *italics*, when not titles or used for emphasis, are special terms with definitions appearing in this document.

Keywords: Keywords reflect concepts or constructs expressed in the language of their source standard. Keywords have been given an identifying prefix to reflect their source. The following prefixes are used:

- `xsd:` identifies keywords from the W3C XML Schema Definition Language specification.
- `xsi:` identifies keywords from the W3C XML Schema's XML Schema Instance specification.
- `structures:` identifies keywords from the NIEM structures namespace.
- `appinfo`: identifies keywords from the NIEM appinfo namespace.

Throughout the document, fragments of XML Schema or XML instances are used to clarify a principle or rule. These fragments are specially formatted in `Courier` font and appear in text boxes. An example of such a fragment follows:

Figure 1-1: Example of an XML fragment

```
<xsd:complexType name="PersonType">
  ...
</xsd:complexType>
```

1.4 Terminology

This document uses standard terminology to explain the principles and rules that describe NIEM.

1.4.1 RFC 2119 Terminology

Within normative content (rules and definitions), the key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in **[RFC2119]**.

1.4.2 XML Information Set Terminology

This document uses the concepts of element information items (“element”), attribute information items (“attribute”), and their associated properties as defined by **[XMLInfoSet]** with clarifications as discussed below. Note that in the clarification that follows, the abstract property names appear in square brackets adjacent to the information items to which they belong. For example, “Element[parent]” discusses the abstract property “parent” of the element information item.

- parent of an element (Element[parent])

  child of an element (Element[children])

  Note that the InfoSet properties “Element[parent]” and “Element[children]” correspond to a direct, immediate relationship with an element. Children of an element and their

children, and so on, are collectively referred to as *descendants* of that element. Parents of an element and their parents, and so on, are collectively referred to as ancestors of that element.

- element owning an attribute (Attribute[owner element])

  The owner of an attribute is the element that possesses or contains the attribute.

The use of the term *document element* from **[XMLInfoSet]** to describe the root of all elements in an XML document is preferred over the informal and nonstandard term *root element*.

1.4.3 XML Schema Terminology

The terms *W3C XML Schema*, *XML Schema* (upper case "Schema"), and *XSD* all refer to the XML Schema definition language, as specified in the two-part XML Schema specification:

- XML Schema Part 1: Structures **[XMLSchemaStructures]**
- XML Schema Part 2: Datatypes **[XMLSchemaDatatypes]**

The term *XML schema* (lower case "schema") refers to specific XML schema documents that conform to the XML Schema specifications listed above.

The terms *XML instance* and *XML document* refer to an XML instance document, which is defined by and validates to a particular XML schema.

The term *schema component* is defined in **[XMLSchemaStructures]** as a building block for XML Schema. This document refers to, rather than restates, the definitions of the different schema components associated with the XML Schema Abstract Data Model, which are defined in the XML Schema specification. In this document, the name of the referenced schema component may appear without the suffix "schema component" (e.g., the term "complex type definition" may be used instead of "complex type definition schema component") to enhance readability of the text.

The term *NCName* is defined in **[XMLSchemaDatatypes]** and refers to XML *noncolonized* names, which are XML name strings that do not contain the ":" character.

1.4.4 XML Namespace Terminology

This document uses the concept of an *XML Namespace* as defined by **[XMLNamespaces]** and **[XMLNamespacesErrata]**.

1.5 Document Organization

This remainder of this document is organized into sections as follows:

- NIEM Conformance describes terminology, requirements, and artifacts related to NIEM conformance.
- The NIEM Conceptual Model discusses the underlying semantic model for NIEM.
- *Guiding Principles* discusses the principles that serve as the foundation of and guidelines for the rules.

- *Relation to Standards* discusses the use of the key standards used in the development of NIEM.
- *XML Schema Design Rules* discusses the rules for using XML Schema constructs in NIEM-conformant schemas.
- *Modeling Rules* discusses the rules for the additional structures and constraints needed to build NIEM-conformant schemas.
- XML Instance Rules discusses the rules for NIEM-conformant XML instance documents.
- *Naming Rules* discusses the rules used in naming NIEM-conformant data components.

NOTE: The ordering of the sections is intended to minimize the number of forward references in the document. For this reason, the naming rules appear as the last section of the document, so that the concepts being named have already been discussed.

This document also contains appendices of reference material as follows:

- A brief, non-normative overview of NIEM.
- Indexes of principles, rules, and definitions.
- Discussion and full listings of the NIEM 2.0 supporting schemas (`structures` and `appinfo`).
- An itemized listing of the NIEM 2.0 reference schemas.
- References to external standard documents.

2 NIEM Conformance

This Naming and Design Rules defines NIEM conformance. This definition is performed through terminology definitions and rules. Together, these define several classes of schemas, as well as defining conformance for XML instances of NIEM-conformant schemas. These classes of schemas are defined, along with the definition of NIEM conformance for XML documents, in Section 2.1, Conformance Targets, below. The schemas defined therein are NIEM-conformant schemas:

[Definition: NIEM-conformant schema]

> An XML Schema document is a **NIEM-conformant schema** if and only if it is a reference schema, a subset schema, an extension schema, an exchange schema, or a constraint schema.

Neither constraint schemas nor subset schemas serve as the primary (cardinal) definitions for components they define. The primary definitions come from reference schemas, exchange schemas, and extension schemas. The XML Schema components defined by these schemas are NIEM-conformant components.

[Definition: NIEM-conformant component]

A **NIEM-conformant component** is an XML Schema component that is defined by a reference schema, an extension schema, or an exchange schema.

The NIEM support schemas, `structures` and `appinfo`, are considered part of the infrastructure of NIEM schemas and are not themselves considered to be NIEM-conformant schemas.

2.1 Conformance Targets Overview

The sections below define the conformance targets for this document. Each rule in this document is applicable to one or more of the conformance targets.

Throughout the document, each rule definition contains a list of applicable conformance targets (as described in Section 1.3.2, Normative and Informative Content, above). The rule is binding for the targets on this list. This list is normative. This list uses the following codes:

Table 2-1: Codes Representing Conformance Targets

Code	Conformance target
REF	Reference schemas
SUB	Subset schemas
EXT	Extension and exchange schemas
CON	Constraint schemas
INS	XML instance data

Each section below provides a list of rules that apply to its conformance target. These lists are informative, not normative. The applicability of a rule to a conformance target is normatively specified by the applicability list contained in the rule definition.

These conformance targets define the scope of the NDR. Anything not on this list of conformance targets is explicitly not addressed.

2.2 Reference Schemas

A NIEM reference schema is a schema that is intended to be the authoritative definition schema for a NIEM namespace. This includes the reference schemas for the NIEM Core schema and NIEM domain schemas.

[Definition: reference schema]

A **reference schema** is an XML Schema document that meets all of the following criteria:

- It is explicitly designated as a reference schema. This may be declared by an IEPD catalog or by a tool-specific mechanism outside the schema.
- It provides the broadest, most fundamental definitions of components in its namespace.

- It provides the authoritative definition of business semantics for components in its namespace.
- It is intended to serve as the basis for components in IEPD schemas, including subset schemas, constraint schemas, extension schemas, and exchange schemas.
- It satisfies all rules specified in the Naming and Design Rules for reference schemas.

Any schema that defines components that are intended to be incorporated into NIEM Core or a NIEM domain may be defined as a reference schema.

The rules for reference schemas are more stringent than are the rules for other classes of NIEM-conformant schemas. Reference schemas are intended to support the broadest reuse. They are very uniform in their structure. As they are the primary definitions for data components, they do not need to restrict other data definitions, and they are not allowed to use XML Schema's restriction mechanisms. Reference schemas are intended to be as regular and simple as possible.

The following rules apply to reference schemas:

- All rules in Section 5
- All rules in Section 6, except [Rule 6-20] through [Rule 6-22] and [Rule 6-57]
- All rules in Section 7, except [Rule 7-69] and [Rule 7-70]
- [Rule 8-7]
- All rules in Section 9

2.3 IEPD Subset Schemas

[Definition: subset schema]

A **subset schema** is an XML Schema document that meets all of the following criteria:

- It is explicitly designated as a subset schema. This may be declared by an IEPD catalog or by a tool-specific mechanism outside the schema.
- It has a target namespace previously defined by a reference schema. That is, it does not provide original definitions for schema components, but instead provides an alternate schema representation of components that are defined by a reference schema.
- It does not alter the business semantics of components in its namespace. The reference schema defines these business semantics.
- It is intended to express the limited vocabulary necessary for an IEPD and to support XML Schema validation for an IEPD.
- It satisfies all rules specified in the Naming and Design Rules for subset schemas.

A subset schema is based on another NIEM-conformant schema: a reference schema. A subset schema is defined such that any valid instance of the subset schema is also a valid instance of

the base (reference) schema. This means that a subset schema is not allowed to introduce new content, nor is it allowed to extend the data content defined by a component of the reference schema.

For example, a subset schema would not be allowed to introduce a new U.S. state (e.g., "West Michigan") into a list of states defined by the reference schema. Any XML instance that included the new state would validate against the supposed subset schema but would not validate against the reference schema. This would violate the basic premise underlying the use of subsets: subsets must be as restrictive as or more restrictive than the reference schema.

A subset schema may omit any construct of the base schema that has no effect on schema validation, including `xsd:documentation` and `xsd:appinfo` annotations. The reference schema on which a subset schema is based is considered the authoritative source of such annotations.

The following rules apply to subset schemas:

- All rules in Section 5, except [Rule 5-4]
- All rules in Section 6, except [Rule 6-16], [Rule 6-20] through [Rule 6-22], [Rule 6-26], [Rule 6-27], [Rule 6-46], [Rule 6-47], [Rule 6-49] through [Rule 6-51], [Rule 6-53], [Rule 6-55], and [Rule 6-57]
- In Section 7, [Rule 7-2], [Rule 7-3], [Rule 7-37], [Rule 7-38], [Rule 7-40], [Rule 7-42] through [Rule 7-44], [Rule 7-47], [Rule 7-48], [Rule 7-51] through [Rule 7-53], [Rule 7-55] through [Rule 7-59], [Rule 7-64], [Rule 7-65], [Rule 7-68] through [Rule 7-70]
- All rules in Section 9

2.4 IEPD Extension Schemas and Exchange Schemas

[Definition: extension schema]

An extension schema is an XML Schema document that meets all of the following criteria:

- It is explicitly designated as an extension schema. This may be declared by an IEPD catalog or by a tool-specific mechanism outside the schema.
- It provides the broadest, most fundamental definitions of components in its namespace.
- It provides the authoritative definition of business semantics for components in its namespace.
- It contains components that, when appropriate, use or are derived from the components in reference schemas or exchange schemas. When a reference schema contains relevant components, it is preferred to an exchange schema.
- It is intended to express the additional vocabulary required for an IEPD, above and beyond the vocabulary available from reference schemas, and to support XML Schema validation for an IEPD.

- It satisfies all rules specified in the Naming and Design Rules for extension schemas.

[Definition: exchange schema]

An exchange schema is an XML Schema document that meets all of the following criteria:

- It is explicitly designated as an exchange schema. This may be declared by an IEPD catalog or by a tool-specific mechanism outside the schema.
- It provides the broadest, most fundamental definitions of components in its namespace.
- It provides the authoritative definition of business semantics for components in its namespace.
- It contains components that use or are derived from the components in reference schemas or exchange schemas.
- It is intended to identify and define the document element information item for a particular information exchange that is described by an IEPD.
- It satisfies all rules specified in the Naming and Design Rules for exchange schemas.

An extension schema in an IEPD serves several functions. First, it defines new content within a new namespace, which may be an IEPD-specific namespace or a namespace shared by several IEPDs. This content is NIEM-conformant but has fewer restrictions on it than do NIEM reference schemas. Second, the extension schema bases its content on content from NIEM reference schemas, where appropriate. Methods of deriving content include using (by reference) existing components, as well as creating extensions and restrictions of existing components.

For example, an IEPD may create a type for an IEPD-specific phone number and base that type on a type defined by the NIEM Core reference schema. This IEPD-specific phone number type may restrict the NIEM Core type to limit those possibilities that are permitted of the base type.

IEPD extensions and restrictions must include annotations and documentation to be conformant, but they are allowed to use restriction, choice, and some other constructs that are not allowed in NIEM reference schemas.

Note that IEPDs may define schemas that meet the criteria of reference schemas for those components that the IEPD wishes to nominate for inclusion in NIEM Core or in domains.

The following rules apply to extensions and exchange schemas:

- All rules in Section 5
- All rules in Section 6, except [Rule 6-11], [Rule 6-18], [Rule 6-19], [Rule 6-29] through [Rule 6-31], [Rule 6-53], and [Rule 6-55]
- All rules in Section 7, except [Rule 7-69] and [Rule 7-70]
- [Rule 8-7]
- All rules in Section 9

2.5 IEPD Constraint Schemas

[Definition: constraint schema]

A constraint schema is an XML Schema document that meets all of the following criteria:

- It is explicitly designated as a constraint schema. This may be declared by an IEPD catalog or by a tool-specific mechanism outside the schema.
- It contains XML Schema components that exist for the purpose of (1) determining schema-validity of XML documents according to some criteria not easily expressed in other classes of schema documents, and (2) expressing those criteria in the XML Schema definition language.
- It has a target namespace previously defined by a reference schema, extension schema, or exchange schema, or it is intended to support a constraint schema that does have such a target namespace.
- It is intended to express business rules for a class of XML documents, not the semantics of those XML documents.
- It satisfies all rules specified in the Naming and Design Rules for constraint schemas.

Constraint schemas provide a mechanism within an IEPD by which the IEPD may use the XML Schema definition language to describe business rules for NIEM-conformant reference schemas. A constraint schema need not express the complete syntax for any class of XML documents. Schema-validity should be assessed using reference or subset schemas as well as constraint schemas.

A constraint schema is not assumed to be a definitive definition for the components it describes. Instead, a constraint schema uses the XML Schema definition language to add constraints and restrictions to components defined by other schemas.

A constraint schema may be used in tandem with a reference schema, extension schema, or exchange schema to enable validation of specific business rules. Or, a broader constraint schema, which adds constraints to the rules defined by the reference schemas, may be defined for an IEPD. Such a schema may be used as the sole yardstick for validation of the namespace, but combining IEPD constraints with the base schemas may make those constraints harder to understand and reuse later.

Constraint schemas have far fewer requirements than other forms of schema. As they are expected to work in tandem with normative schemas, they are allowed to use the XML Schema language however necessary to express business rules.

The following rules apply to constraint schemas:

- In Section 5, [Rule 5-1] through [Rule 5-3]
- In Section 6, [Rule 6-33], [Rule 6-34], and [Rule 6-35] through [Rule 6-38]
- In Section 7, [Rule 7-2] and [Rule 7-3]

2.6 NIEM-Conformant XML Documents and Elements

This document has specific rules about how NIEM content should be used in XML documents. As well as containing rules for XML Schema documents, this NDR contains rules for NIEM-conformant XML content at a finer granularity than the XML document.

[Definition: NIEM-conformant XML document]

A NIEM-conformant XML document is an XML document that satisfies all of the following criteria:

- The document element is locally schema-valid.
- Each element information item within the XML document that has a namespace name matching the target namespace of a reference schema, extension schema, or exchange schema is a NIEM-conformant element information item.

In this definition and the next definition below, the term *XML document* is as specified in **[XML]**. The terms *document information item, document element*, *element information item*, *namespace name*, and *local name* are as specified in **[XMLInfoSet]**. The term *valid* is as specified in **[XMLSchemaStructures]**.

Schema-validity may be assessed against a single set of schemas or against multiple sets of schemas. Assessment against schemas is as directed by an IEPD, other instructions, or tools.

Note that the document element (root element) of a NIEM-conformant XML document is not required to be a NIEM-conformant element information item. Other specifications, such as the IEPD specification, may add additional constraints to these to specify IEPD or exchange conformance.

[Definition: NIEM-conformant element information item]

A NIEM-conformant element information item is an element information item that satisfies all of the following criteria:

- Its namespace name and local name matches an element declared by a reference schema, extension schema, or exchange schema.
- It occurs within a NIEM-conformant XML document.
- It is locally schema-valid.
- It satisfies all rules specified in the Naming and Design Rules for NIEM-conformant element information items.

Because each NIEM-conformant element information item must be locally schema-valid, each element must validate against the schema definition of the element, even if the element information item is allowed within the document because of a wildcard with `processContents` of "`skip`". Within a NIEM-conformant XML document, each element that is from a NIEM namespace conforms to its schema specification.

NDR rules apply to element information items with respect to the reference schemas for the relevant namespaces. For example, when applying a rule concerning the applicability of an

augmentation element to a type, the definitions as specified in the reference schema are relevant, but definitions in other schemas, such as subset and constraint schemas, are not considered. Such applicability is likely not indicated by subset and constraint schemas, but extension schemas are required to contain sufficient definitions for proper validation of NIEM-conformant instances.

The following rules apply to NIEM-conformant element information items:

- In Section 7, [Rule 7-55]
- All rules in Section 8

3 The NIEM Conceptual Model

The NIEM provides a concrete data model, in the form of a set of XML Schema documents. These schemas may be used to build messages and information exchanges. The schemas spell out what kinds of objects exist and how those objects may be related. XML data that follows the rules of NIEM imply specific meaning. The varieties of XML Schema components used within NIEM-conformant schemas are selected to clarify the meaning of XML data. That is, schema components that do not have a clear meaning have been avoided. NIEM provides a framework within which XML data has a specific meaning.

One limitation of XML and XML Schema is that they do not describe the meaning of an XML document. The XML specification defines XML documents and defines their syntax but does not address the meaning of those documents. The XML Schema specification defines the XML Schema definition language, which describes the structure and constrains the contents of XML documents (schemas).

In a schema, the meaning of a schema component (e.g., element, attribute, or type) may be described using the `xsd:documentation` element. Or, additional information may be included via the `xsd:appinfo` element. Although this may enable humans to understand XML data, more information is needed to support the machine-understandable meaning of XML data. In addition, inconsistency among the ways that schema components may be put together may be a source of confusion.

The RDF Core Working Group of the World Wide Web consortium has developed a simple, consistent conceptual model, the RDF model. The RDF model is described and specified through a set of W3C Recommendations, the Resource Description Framework (RDF) specifications, making it a very well-defined standard. The NIEM model and the rules contained in this NDR are based on the RDF model. This provides numerous advantages:

- NIEM's conceptual model is defined by a recognized standard.
- NIEM's conceptual model is very well defined.
- NIEM's conceptual model provides a consistent basis for relating attributes, elements, types, and other XML Schema components.

- NIEM's use of the RDF model defines what a set of NIEM data means. The RDF specification provides a detailed description of what a statement means (see **[RDFSemantics]**), and this is leveraged by NIEM.
- NIEM's use of the RDF model provides a basis for inferencing and reasoning about XML data that uses NIEM. That is, using the rules defined for the RDF model, programs can determine implications of relationships between NIEM-defined objects.

With the exception of Section 2, NIEM rules are explained in this document without reference to RDF or RDF concepts. Understanding RDF is not required to understand NIEM-conformant schemas or data based on NIEM. However, understanding RDF concepts may deepen understanding of NIEM.

The goal of this section is to clarify the meaning of XML data that is NIEM-conformant and to outline the implications of various modeling constructs in NIEM. The rules for NIEM-conformant schemas and instances are in place to ensure that a specific meaning can be derived from data. That is, the data makes specific assertions, which are well understood since they are derived from the rules for NIEM.

The key concepts underpinning the NIEM conceptual model are discussed in the remainder of this section:

- NIEM and the RDF Model
- NIEM Properties
- Unique Identification of Data Objects
- NIEM Data Model Is Explicit, Not Implicit
- NIEM Data Model Implementation in XML Schema

3.1 NIEM and the RDF Model

NIEM has its foundation in the RDF model. This helps to ensure that NIEM-conformant data has precise meaning. The RDF view of what data means is clarified by **[RDFSemantics]**:

> . . . asserting a sentence makes a claim about the world . . . an assertion amounts to stating a constraint on the possible ways the world might be.

The RDF view of the meaning of data carries into NIEM: NIEM elements form statements that make claims about the world: that a person has a name, a residence location, a spouse, etc. The assertion of one set of facts does not necessarily rule out other statements: A person could have multiple names, could have moved, or could be divorced. Each statement is a claim asserted to be true by the originator of the statement.

This NDR discusses NIEM data in terms of objects, a term more accessible than the word used by RDF, resources. RDF defines the world in terms of resources. **[RDFSemantics]** describes what may constitute a resource:

> . . . no assumptions are made here about the nature of resources; "resource" is treated here as synonymous with "entity," i.e., as a generic term for anything in the universe of discourse.

RDF resources coincide with NIEM objects and associations. That is, both objects and associations in NIEM are RDF resources with the additional constraints:

- A NIEM object or association is an instance of a complex type defined by an XML Schema document.
- The XML Schema document that defines a NIEM object is a NIEM-conformant schema.

NIEM associations are defined as n-ary properties as described in **[N-ary]**, use case 3. NIEM object types are defined in Section 7.4.1, Object Types. NIEM associations are defined in Section 7.4.3, Association Types. Assertions are made via NIEM-conformant XML data, described by Section 8, XML Instance Rules.

The XML Schema types that define NIEM objects and associations are related to each other via elements and attributes. That is, a type contains elements and attributes, and an element or attribute has a value that is an instance of an XML Schema type. In NIEM, these elements and attributes are XML Schema representations of RDF properties, which are described by **[RDFPrimer]**, "2.1 Basic Concepts":

> "RDF is based on the idea that the things being described have properties which have values, and that resources can be described by making statements . . . that specify those properties and values."

This describes how NIEM works: schemas describe things and their properties. NIEM-conformant data specifies objects, the values of their properties, and the relationships between them.

There are several kinds of assertions that may be made with NIEM-conformant data. Examples include:

- An assertion that **an object exists**. An occurrence of an element commonly establishes the existence of an object. Such an object may be tangible or intangible. For example, the element `nc:Person` in an exchange implies that a person does or did exist. An element may also express that an object does not exist (e.g., the license plate ABC123 was never issued), but this is an uncommon case.

  Descriptions of objects may carry an implicit assumption that objects exist. Such an assumption is dependent on the message in which such descriptions are made. If an object that is described does not exist, it should be made explicit in the definition of an element containing or referring to the object.

- An assertion that **an object has a characteristic**. A feature or quality of an object is commonly represented by an element appearing within the element that establishes the object. For example, the height of a person is described by the `nc:PersonHeightMeasure` element. The `nc:PersonHeightMeasure`

element occurs as XML content of the `nc:Person` element. In some cases, a characteristic may be represented by an attribute owned by an element.

- An assertion that **an object participates in a relationship**. A relationship between objects may be established in any of several ways:
  - Both objects may be referenced from an association that establishes the relationship. Associations are also useful for expressing n-ary relationships, as well as relationships supported by additional data.
  - An element may occur within one object that indicates the relationship with the other object. This element may be either a content element or a reference element.

    The NIEM Core schema and some domain schemas have been normalized such that a minimum number of reference or content elements establish relationships. In these cases, use of an association is the more common method for establishing a relationship. However, in an exchange, using a reference or content element to express a relationship may be the simpler, preferred method for expressing a relationship.

3.2 NIEM Properties

NIEM-conformant data describes characteristics of objects and relationships between objects. In RDF, these characteristics and relationships are called **properties** of objects, which is also how NIEM refers to them. NIEM represents properties with element declarations and attribute declarations.

Within data, a property relates XML data much as a verb relates nouns in a sentence: a verb has a subject and an object.

- The **property** itself: What relationship is being asserted? For example, the property may say that a weapon has a user, or that someone has hair of a particular color.
- The **subject**: About what object is the property being asserted? This would be the weapon that has the user, or the person whose hair is being described.
- The **object**: What is the value of the property, or with what other object does the relationship exist? This would be the person who is the user of the weapon or the person whose hair has the color brown.

A property relates *two* objects. Data will describe an object having a characteristic with a specific value or will describe an object with a particular relationship to another object. All properties are pair-wise: between two objects, or between an object and a value.

In theory, any relationship that involves more than two objects may be modeled as a set of binary properties. In NIEM, such relationships may be expressed either as a set of properties (i.e., as element and attribute declarations) or as a complex type defining an association.

3.3 Unique Identification of Data Objects

In NIEM, an exchange is generally ad hoc. That is, a message may be generated without any persistence. It exists only to exchange data and may not have any universal meaning beyond that specific exchange. As such, a message may or may not have a URI as an identifier. NIEM was designed with the assumption that a given exchange need not have any unique identifier; NIEM does not require a unique identifier. NIEM also does not require any object (data instance) to be identified by a URI. This differs from RDF, in which all entities (other than literal values) are identified by globally meaningful URIs.

A NIEM-conformant instance uses XML IDs to identify objects within an XML document; The NIEM XML ID is an attribute `structures:id` of type `xsd:ID`. These IDs are not assumed by NIEM to have any universal significance; they need only be unique within the XML document. The use of an ID is required only when an object must be referenced within the document. NIEM recognizes no correlation between these local IDs and any URI.

Any given implementation, message, or IEPD may be defined to apply a URI or other universally meaningful identifier to an object or message. However, NIEM has no such requirement.

3.4 NIEM Data Model Is Explicit, Not Implicit

In NIEM data, that which is not stated is not implied. If data says a person's name is "John," it is not implicitly saying that he does not have other names, or that “John” is his legal name, or that he is different from a person known as “Bob.” The only assertion being made is that one of the names by which this person is known is "John."

This is one reason that definitions of NIEM content are so important. The definitions must state exactly what any given statement implies. The concept of "legal name" may be defined that makes additional assertions about a name of a person. Such assertions must be made explicit in the definition of the relationship.

3.5 NIEM Data Model Implementation in XML Schema

NIEM defines rules for XML Schema documents that enforce the NIEM conceptual model. The schemas that follow these rules are referred to as **NIEM-conformant schemas**.

As discussed above, NIEM classes and properties are mapped onto XML Schema components. The following is an example of how a NIEM class for “Person” is rendered as an XML Schema complex type definition:

Figure 3-1: Conceptual class rendered as XML Schema complex type

```
<xsd:complexType name="PersonType">
  ...
</xsd:complexType>
```

The following is an example of how a NIEM property for “ImageOperator” is rendered as an element declaration:

Figure 3-2: Conceptual property rendered as element declaration

```
<xsd:element name="ImageOperator" type="nc:PersonType" nillable="true">
  ...
</xsd:element>
```

NIEM also defines rules for XML documents that enforce the NIEM conceptual model. An XML document is called a **NIEM-conformant XML document** if it follows the rules specified by the NIEM-conformant schema, as well as additional rules that are NIEM-specific. For example, in a NIEM-conformant XML document, a reference element must refer to a data element that is of an appropriate XML Schema type. If this is not the case, the document may be valid according to the schema, but it will not be NIEM-conformant.

Figure 3-3: Sample fragment of NIEM-conformant data

```
<nc:Person>
  <nc:PersonHairColorCode>BRN</nc:PersonHairColorCode>
</nc:Person>
```

Based on an element declaration from NIEM Core, the following example illustrates a valid XML instance that does not conform to NIEM. Per the `appinfo:ReferenceTarget` element in the schema declaration, `nc:ActivityReference` may ONLY refer to an `nc:ActivityType`. However, within the instance, `my:ActivityList/nc:ActivityReference` refers to “Bill,” which is an `nc:PersonType`.

Figure 3-4: Schema declaration for element `nc:ActivityReference`

```
<xsd:element name="ActivityReference" type="structures:ReferenceType">
  <xsd:annotation>
    <xsd:documentation>
      A single or set of related actions, events, or process steps.
    </xsd:documentation>
    <xsd:appinfo>
      <appinfo:ReferenceTarget appinfo:name="ActivityType"/>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element
```

Figure 3-5: Valid instance for above schema that does NOT conform to NIEM rules

```
<nc:Person structures:id="Bill">
  <nc:PersonFullName>William Tell</nc:PersonFullName>
  <nc:PersonSexCode>M</nc:PersonSexCode>
</nc:Person>

<nc:Activity structures:id="Pie">
  <nc:ActivityDescriptionText>
    County fair pie-eating contest
  </nc:ActivityDescriptionText>
</nc:Activity>

<my:ActivityList>
  <nc:ActivityReference structures:ref="Pie"/>
  <nc:ActivityReference structures:ref="Bill"/>
</my:ActivityList>
```

4 Guiding Principles

Principles in this specification provide a foundation for the rules. These principles are generally applicable in most cases. They should not be used as a replacement for common sense or appropriate special cases.

The principles are not operationally enforceable; they do not specify constraints on XML Schema documents and instances. The rules are the normative and enforceable manifestation of the principles.

The principles discussed in this section are categorized as follows:

- Specification Guidelines
- XML Schema Design Guidelines
- Modeling Design Guidelines
- Implementation Guidelines

4.1 Specification Guidelines

The principles in this section address what material should be included in this NDR and how it should be represented.

4.1.1 Keep Specification to a Minimum

This specification should state what is required for interoperability, not all that could be specified. Certain decisions (such as normative XML comments) could create roadblocks for interoperability, making heavy demands on systems for very little gain. The goal is not standardization for standardization's sake. The goal is to maximize interoperability and reuse.

[Principle 1]

> This specification SHOULD specify what is necessary for semantic interoperability and no more.

The term **semantic interoperability** is here defined as "the ability of two or more computer systems to exchange information and have the meaning of that information automatically interpreted by the receiving system accurately enough to produce useful results."

4.1.2 Focus on Rules for Schemas

This specification should try, as much as is possible, to specify schema-level content. This is a specification for schemas, and so it should specify schemas. It should avoid specifying complex data models or data dictionaries.

[Principle 2]

> This specification SHOULD focus on providing rules for specifying schemas.

4.1.3 Use Specific, Concise Rules

A rule should be as precise and specific as possible to avoid broad, hard-to-modify rules. Putting multiple clauses in a rule makes it harder to enforce. Using separate rules allows specific conditions to be clearly stated.

[Principle 3]

> This specification SHOULD feature rules that are as specific, precise, and concise as possible.

4.2 XML Schema Design Guidelines

The principles in this section address how XML Schema technology should be used in designing NIEM-conformant schemas and instances.

4.2.1 Disallow Content Modification With XML Processors

XML Schema has constructs that can make the data provided by XML processors different before and after schema processing. An example of this is the use of XML Schema attribute declarations with default values. Before schema validation, there may be no attribute value, but after processing, the attribute value exists.

Within NIEM, the purpose of processing instances against schemas is solely validation: testing that data instances match desired constraints and guidelines. It should not be used to change the content of data instances.

[Principle 4]

> The content of a NIEM-conformant data instance SHOULD NOT be modified by processing against XML Schema documents.

4.2.2 Use XML Validating Parsers for Content Validation

NIEM is designed for XML Schema validation. A primary goal is to maximize the amount of validation that may be performed by XML Schema-validating parsers.

XML Schema validates content using content models: descriptions of what elements and attributes may be contained within an element, and what values are allowable. It is the XML element hierarchy (elements with attributes and unstructured content, contained by other elements) that the XML Schema definition language specifies and that XML Schema validating parsers can validate.

Mechanisms involving linking using attribute and element values are useful, but they should only be relied on when absolutely necessary, as XML Schema-validating parsers cannot readily validate them. For example, if a link is established via attribute values, an XML Schema-validating parser cannot determine that participants have appropriate type definitions. Whenever possible, NIEM content should rely on XML syntax that can be validated with XML Schema.

[Principle 5]

> NIEM-conformant schemas and NIEM-conformant XML documents SHOULD use XML Schema validating parsers for validation of XML content.

4.2.3 Validate for Conformance to Reference Schemas

Systems that operate on XML data have the opportunity to perform multiple layers of processing. Middleware, XML libraries, schemas, and application software may process data. The primary purpose of XML Schema validation is to restrict processed data to that data that conforms to agreed-upon rules. This restriction is achieved by marking as invalid that data that does not conform to the rules defined by the schema.

[Principle 6]

> Systems that use NIEM-conformant data SHOULD mark as invalid data that does not conform to the rules defined by applicable XML Schema documents.

4.2.4 Allow Multiple Schemas for XML Constraints

The NIEM does not attempt to create a one-size-fits-all schema to perform all validation. Instead, it creates a set of reference schemas, on which additional constraints may be placed. It also does not focus on language-binding XML Schema implementations, which convert XML Schema definitions into working programs. It is, instead, focused on normalizing language and preserving the meaning of data.

[Principle 7]

> Constraints on XML instances MAY be validated by multiple schema validation passes, using multiple schemas for a single namespace.

4.2.5 Define One Reference Schema Per Namespace

NIEM uses the concept of a *reference schema*, which defines the structure and content of a namespace. For each NIEM-conformant namespace, there is exactly one NIEM reference schema. A user may use a subset schema or constraint schema in place of a reference schema, but all NIEM-conformant XML documents must validate against a single reference schema for each namespace.

[Principle 8]

> Each NIEM-conformant namespace SHOULD be defined by exactly one reference schema.

4.2.6 Disallow Mixed Content

XML data that use mixed content are difficult to specify and complicate the task of data processing. Much of the payload carried by mixed content is unchecked and does not facilitate data standardization or validation.

[Principle 9]

> NIEM-conformant schemas SHOULD NOT specify data that uses mixed content.

4.2.7 Specify Types for All Constructs

Schema components within NIEM all have names. This means that there are no anonymous types, elements, or other components defined by NIEM. Once an application has determined the name (i.e., namespace and local name) of an attribute or element used in NIEM-conformant instances, it will also know the type of that attribute or element.

There are no local attributes or elements defined by NIEM, only global attributes and elements. This maximizes the ability of application developers to extend, restrict, or otherwise derive definitions of local components from NIEM-conformant components. Using named global components in schemas maximizes the capacity for reuse.

[Principle 10]

> NIEM-conformant schemas SHOULD NOT use or define local or anonymous components, as they adversely affect reuse.

4.2.8 Avoid Wildcards in Reference Schemas

Wildcards in NIEM-conformant schemas work in opposition to standardization. The goal of creating harmonized, standard schemas is to standardize definitions of data. The use of wildcard mechanisms (such as `xsd:any`, which allows insertion of an arbitrary number of elements from any namespace) allows nonstandard data to be passed via otherwise standardized exchanges.

Avoidance of wildcards in the standard schemas encourages the separation of standardized and nonstandardized data. It encourages users to incorporate their data into NIEM in a standardized way. It also encourages users to extend in a way that may be readily incorporated into NIEM.

[Principle 11]

> NIEM-conformant components SHOULD NOT incorporate wildcards unless absolutely necessary, as they hinder standardization by encouraging use of nonstandardized data rather than standardized data.

4.2.9 Provide Default Reference Schema Locations

[XMLSchemaStructures] provides three ways to specify the physical location of an XML Schema document: `schemaLocation`, an attribute of the element `xsd:import`, along with `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation`, attributes of an XML Schema document element. In all of these uses, the specification explicitly maintains that the schema location specified is a hint, which may be overridden by applications.

[Principle 12]

> Schema locations specified within NIEM-conformant reference schemas SHOULD be interpreted as hints and as default values by processing applications.

4.2.10 Use Open Standards

The cooperative efforts of many knowledgeable individuals have resulted in many important published information standards. Where appropriate and applicable, NIEM ought to leverage these standards.

[Principle 13]

> NIEM standards and schemas SHOULD leverage and enable use of other open standards.

4.3 Modeling Design Guidelines

The principles in this section address the design philosophy used in designing the NIEM conceptual model.

4.3.1 Namespaces Enhance Reuse

NIEM is designed to maximize reuse of namespaces and the schemas that define them. When referring to a concept defined by NIEM, a user should ensure that instances and schemas refer to the namespace defined by NIEM. User-defined namespaces should be used for specializations and extension of NIEM constructs but should not be used when the NIEM structures are sufficient.

[Principle 14]

> NIEM-conformant instances and schemas SHOULD reuse components from NIEM distribution schemas when possible.

NIEM relies heavily on XML namespaces to prevent naming conflicts and clashes. Reuse of any component is always by reference to both its namespace and its local name. All NIEM component names have global scope. Therefore, validation always occurs against the reference schemas or subsets thereof.

Example:

Figure 4-1: Example of the use of a namespace

```
<xsd:element ref="nc:BinaryCaptureDate"
    minOccurs="0"
    maxOccurs="unbounded"/>
```

In this example, `nc:BinaryCaptureDate` is reused by referencing its element declaration through both its namespace (which is bound to the prefix `nc:`) and its local name (`BinaryCaptureDate`). If an element named `BinaryCaptureDate` is declared in another namespace, it is an entirely different element than `nc:BinaryCaptureDate`. There is no implicit relationship to `nc:BinaryCaptureDate`.

From a business perspective, the two elements are likely to be *related* in the sense that they may have very similar semantic meanings. They may have essentially the same meaning, but slightly different properties. Such a relationship may commonly exist. However, any relationship between the two elements must be made explicit using methods outlined in this document.

[Principle 15]

> A component SHOULD be identified by its local name together with its namespace. A namespace SHOULD be a required part of the name of a component. A component's local name SHOULD NOT imply a relationship to components with similar names from other namespaces.

4.3.2 Design NIEM for Extensibility

NIEM is designed to be extended. Numerous methods are considered acceptable in creating extended and specialized components.

[Principle 16]

> NIEM-conformant schemas and standards SHOULD be designed to encourage and ease extension and augmentation by users and developers outside the standardization process.

4.4 Implementation Guidelines

The principles in this section address issues pertaining to the implementation of applications that use NIEM.

4.4.1 Avoid Displaying Raw XML Data

XML data should be made human-understandable when possible, but it is not targeted at human consumers. HTML is intended for browsers. Browsers and similar technology provide human interfaces to XML and other structured content. As such, structured XML content does

not belong in places targeting humans. Human-targeted information should be of a form suitable for presentation.

[Principle 17]

> XML data SHOULD be designed for automatic processing. XML data SHOULD NOT be designed for literal presentation to people. NIEM standards and schemas SHOULD NOT use literal presentation to people as a design criterion.

4.4.2 Leave Implementation Decisions to Implementers

NIEM is intended to be an open specification supported by many diverse implementations. It was designed from data requirements and not from or for any particular system or implementation. Use of NIEM should not depend on specific software, other than XML Schema-validating parsers.

[Principle 18]

> NIEM SHOULD NOT depend on specific software packages, software frameworks, or software systems for interpretation of XML instances.

[Principle 19]

> NIEM schemas and standards SHOULD be designed such that software systems that use NIEM may be built with a variety of off-the-shelf and free software products.

4.5 Modeling Guidelines

The NIEM Naming and Design Rules (NDR) specify NIEM-conformant components, schemas, and instances. These guidelines influence and shape the more-specific principles and rules in this document. They are derived from best practices and from discussions within the NIEM Business Architecture Committee (NBAC) and the NIEM Technical Architecture Committee (NTAC). This list may grow and evolve as NIEM matures.

The principles in this section address decisions that data modelers must face when creating NIEM-conformant schema representations of domain data. These guidelines are not absolute (the key word is SHOULD). It may not be possible to apply all guidelines in every case. However, they should always be considered.

4.5.1 Documentation

As will be described in later sections of this document, all NIEM components are documented through their definitions and names. Although it is often very difficult to apply, a data component definition should be drafted before the data component name is finalized.

Drafting the definition for a data component first ensures that the author understands the exact nature of the entity or concept that the data component represents. The component name should subsequently be composed to summarize the definition. Reversing this sequence often results in data definitions that very precisely describe the component name but do not adequately describe the entity or concept that the component is designed to represent. This can lead to the ambiguous use of such components.

[Principle 20]

A data component definition SHOULD be drafted before the associated data element name is composed.

4.5.2 Consistent Naming

Components in NIEM should be given names that are consistent with names of other NIEM components. Having consistent names for components has several advantages:

1. It is easier to determine the nature of a component when it has a name that conveys the meaning and use of the component.
2. It is easier to find a component when it is named predictably.
3. It is easier to create a name for a component when clear guidelines exist.

[Principle 21]

Components in NIEM SHOULD be given names that are consistent with names of other NIEM components. Such names SHOULD be based on simple rules.

4.5.3 Reflect the Real World

NIEM provides a standard for data exchange. To help facilitate unambiguous understanding of NIEM reusable components, the names and structures should represent and model the informational aspects of objects and concepts that users are most familiar with. Types should not simply model collections of data.

[Principle 22]

Component definitions in NIEM-conformant schemas SHOULD reflect real-world concepts.

4.5.4 Be Consistent

There should be no conflicts of meaning among types. This holds for types within a namespace, as well as types in different namespaces. A type should be used consistently in similar situations for similar purposes. Types should be defined for clear understanding and ease of intended use.

[Principle 23]

Component definitions in NIEM-conformant schemas SHOULD have semantic consistency.

4.5.5 Reserve Inheritance for Specialization

Specialization should not be applied simply for the sake of achieving property inheritance. Specialization should be applied only where it is meaningful and appropriate to model permanent sibling subclasses of a base class that are mutually exclusive of one another.

[Principle 24]

> Complex type definitions in NIEM-conformant schemas SHOULD use type inheritance only for specialization.

Note that application of type augmentations is a well-defined exception to this guideline.

4.5.6 Do Not Duplicate Definitions

A real-world entity should be modeled in only one way. The definition of a type or element should appear once and only once. Multiple components of identical or closely similar semantics hinder interoperability because too many valid methods exist for representing the same data. For each data concept that must be represented, there should be only one component (and associated type) to represent it.

Components with very similar semantics may exist in different contexts. For example, a complex type created for a particular exchange may appear to have identical or closely similar semantics to a complex type defined in the NIEM Core schema. However, the type defined at the exchange level will have much more precise business requirements and syntax, compared with the broad definitions that are heavily reused. Specific contextual definitions should be considered semantic changes. This includes the application of augmentations to create a specialized type for a specific use.

Two components may have the same definition while having different representations. For example, a string may hold the complete name of a person, or the name may be represented by a structure that separates the components of the name into first, last, etc. The definition of alternative representations should not be considered duplication.

[Principle 25]

> Multiple components with identical or undifferentiated semantics SHOULD NOT be defined. Component definitions SHOULD have clear, explicit distinctions.

4.5.7 Keep It Simple

All NIEM content and structure is fundamentally based on business requirements for information exchange. To encourage adoption and use in practice, NIEM must implement business requirements in simple, consistent, practical ways.

[Principle 26]

> NIEM-conformant schemas SHOULD have the simplest possible structure, content, and architecture consistent with real business requirements.

4.5.8 Be Aware of Scope

The scope of components defined in NIEM-conformant schemas should be carefully considered. Some components represent simple data values, while others represent complex objects with many parts and relationships. Components should exist in layers. Components should exist as small, narrowly scoped, atomic entities that are used to consistently construct more broadly scoped, complex components (and so on).

[Principle 27]

Components defined by NIEM-conformant schemas SHOULD be defined appropriate for their scope.

4.5.9 Be Mindful of Namespace Cohesion

Namespaces should maximize cohesion. The namespace methodology helps prevent name clashes among communities or domains that have different business perspectives and may choose identical data names to represent different data concepts. A namespace should be designed so that its components are consistent, may be used together, and may be updated at the same time.

[Principle 28]

XML namespaces defined by NIEM-conformant schemas SHOULD encapsulate data components that are coherent, consistent, and internally related as a set. A namespace SHOULD encapsulate components that tend to change together.

5 Relation to Standards

This section specifies the standards and specifications to which NIEM conforms. Where NIEM differs from public standards, the rationale for those differences is discussed in this section. The complete list of standards and specifications referenced in this section appears in Appendix D: References.

5.1 XML 1.0

[Rule 5-1] (REF, SUB, EXT, CON)

The schema MUST conform to XML as specified by **[XML]**.

Rationale

XML is a well-known, commonly used W3C Recommendation. It is supported by a large number of commercial and open-source software tools. It is a simple, well-defined, semi-structured data format that is flexible enough to allow for easy extension. XML works with many other powerful associated technologies such as XML Schema, XSLT, and XPath. Artifacts of NIEM conform to the most recent recommendation for XML.

5.2 XML Namespaces

[Rule 5-2] (REF, SUB, EXT, CON)

The schema MUST conform to the specification for namespaces in XML, as defined by **[XMLNamespaces]** and **[XMLNamespacesErrata]**.

Rationale

NIEM is designed to facilitate cross-domain data exchanges and interoperability. The ultimate scope of NIEM is anticipated to be quite large. The primary purpose of

namespaces is to avoid naming conflicts, which for NIEM could become quite common, since NIEM stakeholders and IEPD developers define and name many of their own data components independently. Therefore, in NIEM, XML namespaces are employed both to avoid name clashes and to provide a level of independence to participating domains.

5.3 XML Schema

[Rule 5-3] (REF, SUB, EXT, CON)

The schema MUST conform to the W3C XML Schema Recommendations: XML Schema Part 1: Structures and XML Schema Part 2: Datatypes, as specified by **[XMLSchemaStructures]** and **[XMLSchemaDatatypes]**.

Rationale

XML Schema has become the generally accepted schema language and is experiencing the most widespread adoption. Although other schema languages exist that offer their own advantages and disadvantages, the current approach is to base NIEM on XML Schema.

5.4 ISO 11179, Part 4

Good data definitions are fundamental to data interoperability. You cannot effectively exchange what you cannot understand. NIEM employs the guidance of **[ISO 11179 Part 4]** as a baseline for its data component definitions. All NIEM components are documented.

[Definition: documented component]

In a NIEM-conformant schema, a **documented component** is an XML Schema component that has an associated data definition. These schema components have a textual definition, so that the component may be well-understood. Schemas that do not document their components accordingly are not NIEM-conformant.

[Definition: data definition]

The **data definition** of a documented component is the content of the first occurrence of the element `xsd:documentation`, which is an immediate child of an occurrence of the element `xsd:annotation`, which is an immediate child of the element that defines the component.

Figure 5-1: Example of data definition of `MeasureMetadataType`

```
<xsd:complexType name="MeasureMetadataType">
  <xsd:annotation>
    <xsd:documentation>
      A data type for metadata about a measurement.
    </xsd:documentation>
    <xsd:appinfo>
      <appinfo:Base
          appinfo:namespace="http://niem.gov/niem/structures/2.0"
          appinfo:name="MetadataType"/>
      <appinfo:AppliesTo appinfo:name="MeasureType"/>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="s:MetadataType">
      <xsd:sequence>
        <xsd:element ref="nc:MeasureDate"
            minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="nc:Measurer"
            minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

[Rule 5-4] (REF, EXT)

Within a NIEM-conformant schema, the data definition provided for each documented component SHALL follow the requirements and recommendations for data definitions given by **[ISO 11179 Part 4]**.

Rationale

To advance the goal of creating semantically rich NIEM-conformant schemas, it is necessary that data definitions be descriptive, meaningful, and precise. **[ISO 11179 Part 4]** provides standard structure and rules for defining data definitions. NIEM uses this standard for component definitions.

Note that the metadata maintained for each NIEM component contains additional details, including domain-specific usage examples and keywords. Such metadata is used to enhance search and discovery of components in a registry, and therefore, is not included in schemas.

For convenience and reference, the summary requirements and recommendations in **[ISO 11179 Part 4]** are reproduced here:

ISO 11179 Requirements

A data definition SHALL:

- Be stated in the singular.
- State what the concept is, not only what it is not.
- Be stated as a descriptive phrase or sentence(s).
- Contain only commonly understood abbreviations.
- Be expressed without embedding definitions of other data or underlying concepts.

ISO 11179 Recommendations

A data definition SHOULD:

- State the essential meaning of the concept.
- Be precise and unambiguous.
- Be concise.
- Be able to stand alone.
- Be expressed without embedding rationale, functional usage, or procedural information.
- Avoid circular reasoning.
- Use the same terminology and consistent logical structure for related definitions.
- Be appropriate for the type of metadata item being defined.

In addition to the requirements and recommendations of **[ISO 11179 Part 4]**, NIEM applies additional rules to data definitions. These rules are detailed in Section 7.2.1, Human-Readable Documentation.

5.5 ISO 11179, Part 5

Names are a simple but incomplete means of providing semantics to data components. Data definitions, structure, and context help to fill the gap left by the limitations of naming. The goals for data component names should be syntactic consistency, semantic precision, and simplicity. In many cases, these goals conflict and it is sometimes necessary to compromise or to allow exceptions to ensure clarity and understanding. To the extent possible, NIEM applies **[ISO 11179 Part 5]** to construct NIEM data component names.

The set of NIEM data components is a collection of data representations for real-world objects and concepts, along with their associated properties and relationships. Thus, names for these components would consist of the terms (words) for object classes or that describe object classes, their characteristic properties, subparts, and relationships.

[Rule 5-5] (REF, SUB, EXT)

> A NIEM component name SHALL be formed by applying the informative guidelines and examples detailed in Annex A of **[ISO 11179 Part 5]**, with exceptions as specified in this document, most notably those specified in Section 9, Naming Rules.

Rationale

> The guidelines and examples of **[ISO 11179 Part 5]** provide a simple, consistent syntax for data names that captures context and thereby imparts a reasonable degree of semantic precision.

NIEM uses the guidelines and examples of **[ISO 11179 Part 5]** as a baseline for normative naming rules. However, some NIEM components require bending of these rules. Special naming rules for these classes of components are presented and discussed in Section 9. In spite of these exceptions, most NIEM component names can be disassembled into their **[ISO 11179 Part 5]** constituent words or terms.

Example:

The NIEM component name `AircraftFuselageColorCode` disassembles as follows:

- Object class term = "`Aircraft`"
- Qualifier term = "`Fuselage`"
- Property term = "`Color`"
- Representation term = "`Code`"

Section 9, Naming Rules, details the specific rules for each kind of term and how to construct NIEM component names from it. Exceptions for special components are also described in Section 9.

6 XML Schema Design Rules

The W3C XML Schema Language provides many features that allow a developer to represent a logical data model many different ways. This section establishes rules for the use of XML Schema constructs within NIEM-conformant schemas. Because the XML Schema specifications are flexible, comprehensive rules are needed to achieve a balance between establishing uniform schema design and providing developers flexibility to solve novel data modeling problems.

Note that external schemas (non-NIEM-conformant schemas) do not need to obey the rules set forth in this section. So long as schema components from external schemas are adapted for use with NIEM, according to the modeling rules in Section 7.7, they may be used as they appear in the external standard, even if the schema components violate the rules for NIEM-conformant schemas.

The XML Schema design rules in this section fall into the following categories:

- Restrictions on XML Schema Constructs
- `xsd:schema` Document Element
- Namespace Imports
- Annotations
- Type Definitions
- Additional Definitions and Declarations

6.1 Restrictions on XML Schema Constructs

A number of XML Schema constructs are not used within NIEM-conformant schemas. Many of these constructs provide capability that is not currently needed within NIEM. Some of these constructs create problems for interoperability, with tool support, or with clarity or precision of data model definition.

6.1.1 No Mixed Content

[Rule 6-1] (REF, SUB, EXT)

Within the schema, an element `xsd:complexType` SHALL NOT own the attribute `mixed` with the value `true`.

[Rule 6-2] (REF, SUB, EXT)

Within the schema, an element declaration that is of complex content SHALL NOT own the attribute `mixed` with the value `true`.

Rationale

Mixed content allows the mixing of data tags with text. Languages such as XHTML use this syntax for markup of text. NIEM-conformant schemas define XML that is for data exchange, not text markup. Mixed content creates complexity in processing, defining, and constraining content.

Well-defined markup languages exist outside NIEM and may be used with NIEM data. External schemas may include mixed content and may be used with NIEM. However, mixed content must not be defined by NIEM-conformant schemas in keeping with [Principle 9].

6.1.2 No Notations

[Rule 6-3] (REF, SUB, EXT)

The schema SHALL NOT contain a reference to the type definition `xsd:NOTATION` or to a type derived from that type.

[Rule 6-4] (REF, SUB, EXT)

The schema SHALL NOT contain the element `xsd:notation`.

Rationale

XML Schema notations allow the attachment of system and public identifiers on fields of data. The notation mechanism does not play a part in validation of instances and is not supported by NIEM.

6.1.3 No Schema Inclusion

[Rule 6-5] (REF, SUB, EXT)

The schema SHALL NOT contain the element `xsd:include`.

Rationale

Element `xsd:include` brings schemas defined in separate files into the current namespace. It breaks a namespace up into arbitrary partial schemas, which needlessly complicates the schema structure, making it harder to reuse and process, and also increases the likelihood of conflicting definitions.

Inclusion of schemas that do not have namespaces also complicates schema understanding. This inclusion makes it difficult to find the realization of a specific schema artifact and create aliases for schema components that should be reused. Inclusion of schemas also violates [Principle 8], as it uses multiple schemas to construct a namespace.

6.1.4 No Schema Redefinition

[Rule 6-6] (REF, SUB, EXT)

The schema SHALL NOT contain the element `xsd:redefine`.

Rationale

The `xsd:redefine` element allows an XML Schema document to restrict and extend components from a namespace, in that very namespace. Such redefinition introduces duplication of definitions, allowing multiple definitions to exist for components from a single namespace. This violates [Principle 8] that a single reference schema defines a NIEM-conformant namespace.

6.1.5 Wildcard Restrictions

There are many constructs within XML Schema that act as wildcards. That is, they introduce buckets that may carry arbitrary or otherwise nonvalidated content. Such constructs violate [Principle 11], and as such provide implicit workarounds for the difficult task of agreeing on the content of data models. Such workarounds should be made explicitly, outside the core data model.

6.1.5.1 No Unconstrained Type Substitution

[Rule 6-7] (REF, SUB, EXT)

The schema SHALL NOT reference the type `xsd:anyType`.

Rationale

XML Schema has the concept of the "ur-type," a type that is the root of all other types. This type is realized in schemas as `xsd:anyType`.

NIEM-conformant schemas must not use `xsd:anyType`, because this feature permits the introduction of arbitrary content (i.e., untyped and unconstrained data) into an XML instance. NIEM intends that the schemas describing that instance describe all constructs within the instance.

6.1.5.2 No Unconstrained Text Substitution

[Rule 6-8] (REF, SUB, EXT)

The schema SHALL NOT reference the type `xsd:anySimpleType`.

Rationale

XML Schema provides a restriction of the "ur-type," which contains only simple content. This provides a wildcard for arbitrary text. It is realized in XML Schema as `xsd:anySimpleType`.

NIEM-conformant schemas must not use `xsd:anySimpleType` because this feature is insufficiently constrained to provide a meaningful starting point for content definitions. Instead, content should be based on one of the more specifically defined simple types defined by XML Schema.

6.1.5.3 Untyped Elements Must Be Abstract

[Rule 6-9] (REF, SUB, EXT)

Within the schema, an element declaration with the attribute `name` and without the attribute `type` MUST carry the attribute `abstract` with the value `true`.

Rationale

Untyped element declarations act as wildcards that may carry arbitrary data. By declaring such types abstract, NIEM allows the creation of type independent semantics without allowing arbitrary content to appear in XML instances.

6.1.5.4 No Untyped Attributes

[Rule 6-10] (REF, SUB, EXT)

Within the schema, an attribute declaration with attribute `name` MUST carry the attribute `type`.

Rationale

Untyped XML Schema attributes allow arbitrary content, with no semantics. Attributes must have a type so that specific syntax and semantics will be provided.

6.1.5.5 No Unconstrained Element Substitution

[Rule 6-11] (REF, SUB)

The schema SHALL NOT contain the element `xsd:any`.

Rationale

The `xsd:any` particle (see Model Group Restrictions for an informative definition of particle) provides a wildcard that may carry arbitrary content. The particle `xsd:any` may appear within constraint schemas, extension schemas, and exchange schemas.

6.1.5.6 No Unconstrained Attribute Substitution

[Rule 6-12] (REF, SUB, EXT)

The schema SHALL NOT contain the element `xsd:anyAttribute`.

Rationale

The `xsd:anyAttribute` element provides a wildcard, where arbitrary attributes may appear. The element `xsd:anyAttribute` may appear within constraint schemas or within other schemas that are not NIEM-conformant, but it is prohibited in NIEM-conformant schemas.

6.1.6 Component Naming Restrictions

All NIEM components must be named. That is, type definitions, and element and attribute declarations must be given explicit names — local and anonymous component definition is not allowed. Note that XML Schema enforces the placement of attribute group and model group definitions as top-level components, which forces the components to be named.

6.1.6.1 No Anonymous Type Definitions

[Rule 6-13] (REF, SUB, EXT)

Within the schema, any occurrence of the element `xsd:complexType` or `xsd:simpleType` MUST appear as an immediate child of the element `xsd:schema`.

Rationale

NIEM does not support anonymous types in NIEM-conformant schemas. All XML Schema "top-level" types (children of the document element) are required by XML Schema to be named. By requiring NIEM type definitions to be top level, they are forced to be named and are therefore globally reusable.

6.1.6.2 No Local Element Declarations

[Rule 6-14] (REF, SUB, EXT)

Within the schema, any element declaration carrying the attribute `name` MUST appear as an immediate child of the document element `xsd:schema`.

Rationale

All schema components defined by NIEM-conformant schemas must be named, accessible from outside the defining schema, and reusable across schemas. Local element definitions provide named elements that are not reusable outside the context in which they are defined. Requiring named NIEM elements to be top level ensures that they are globally reusable.

6.1.6.3 No Local Attribute Definitions

[Rule 6-15] (REF, SUB, EXT)

Within the schema, any attribute declaration owning the attribute `name` MUST appear as an immediate child of the document element `xsd:schema`.

Rationale

All schema components defined by NIEM-conformant schemas are named, accessible from outside the defining schema, and reusable across schemas. Local attribute definitions provide named attributes that are not reusable outside the context in which they are defined. Requiring named NIEM attributes to be top level ensures that they are globally reusable.

6.1.7 No Uniqueness Constraints

[Rule 6-16] (REF, EXT)

The schema SHALL NOT contain any of the elements `xsd:unique`, `xsd:key`, `xsd:keyref`, `xsd:selector`, or `xsd:field`.

Rationale

XML Schema provides NIEM with the ability to apply uniqueness constraints to schema-validated content. These mechanisms, however, establish relationships in a way that is very difficult to understand, extend, and keep consisent through schema reuse. These elements may be used in subset schemas and constraint schemas.

6.1.8 Model Group Restrictions

Complex content definitions in XML Schema use model group schema components. These schema components, `xsd:all`, `xsd:choice` and `xsd:sequence`, also called compositors, provide for ordering and selection of particles within a model group.

XML Schema defines a **particle** as an occurrence of `xsd:element`, `xsd:sequence`, `xsd:choice`, `xsd:any` (wildcard) and `xsd:group` (model group) within a content model. For example, an `xsd:sequence` within an XML Schema complex type is a particle. An `xsd:element` occurring within an `xsd:sequence` is also a particle.

6.1.8.1 Restrictions on Particle Ordering

[Rule 6-17] (REF, SUB, EXT)

The schema SHALL NOT contain the element `xsd:all`.

Rationale

The element `xsd:all` provides a set of particles (e.g., elements) that may be included in an instance, in no particular order. This can greatly complicate processing and may be difficult to comprehend and satisfy.

[Rule 6-18] (REF)

The schema SHALL NOT contain the element `xsd:choice`.

Rationale

The element `xsd:choice` provides an exclusive set of particles, one of which may appear in an instance. This can greatly complicate processing and may be difficult to comprehend, satisfy, and reuse.

The element `xsd:choice` may be used in extension and exchange schemas, as it presents a simple way for a schema writer to represent a set of optional content. It may also be used in subset schemas and constraint schemas to represent syntactic alternatives.

6.1.8.2 No Recursively Defined Model Groups

[Rule 6-19] (REF, SUB)

Within the schema, any immediate child of a model group `xsd:sequence` element MUST be one of `xsd:annotation` or `xsd:element`

[Rule 6-20] (EXT)

Within the schema, any immediate child of a model group `xsd:sequence` element MUST be one of `xsd:annotation`, `xsd:element`, `xsd:choice`, or `xsd:any`.

[Rule 6-21] (EXT)

Within the schema, any immediate child of a model group `xsd:choice` element MUST be one of `xsd:annotation` or `xsd:element`.

[Rule 6-22] (EXT)

The use of `xsd:choice` SHALL define syntax, structure, grouping, and cardinality of instances, but SHALL NOT define semantics. The semantics of a property within an `xsd:choice` SHALL be identical to the semantics of the property within an `xsd:sequence`.

Rationale

XML Schema provides the capability for model groups to be recursively defined. This means that a sequence may contain a sequence, and a choice may contain a choice. These rules are designed to keep content models simple, comprehensive, and reusable: The content of an element should boil down to a simple list of elements, defined in as straightforward a manner as is possible to meet requirements.

6.1.8.3 Restrictions on Named Groups

[Rule 6-23] (REF, SUB, EXT)

The schema SHALL NOT contain the element `xsd:group`.

Rationale

NIEM does not allow groups of elements to be named other than as named complex types. A group in XML Schema creates a named entity that may be included in multiple

types, and which consists of a sequence of or choice between element particles. The NIEM has not developed a semantic model for these components, and they are not integrated into NIEM's design.

6.1.8.4 Particle Cardinality Restrictions

[Rule 6-24] (REF, SUB, EXT)

Within the schema, if the element `xsd:sequence` carries the attribute `minOccurs`, it MUST set the value for the attribute to `1`.

[Rule 6-25] (REF, SUB, EXT)

Within the schema, if the element `xsd:sequence` carries the attribute `maxOccurs`, it MUST set the value of the attribute to `1`.

Rationale

XML Schema allows each particle to specify cardinality (how many times the particle may appear in an instance). NIEM restricts the cardinality of `xsd:sequence` particles to exactly one, to ensure that content model definitions are defined in as straightforward a manner as possible.

Discussion

Note that the particle `xsd:any` is not allowed in reference schemas or subset schemas by [Rule 6-11]

Note also that element declarations acting as a particle (particles formed by `xsd:element`) may have any cardinality; they are not restricted by this rule. Should a user desire the behavior that would be obtained from the use of special cardinalities on these particles, he or she should define them within explicitly named elements.

6.1.9 Block Substitution Restrictions

XML Schema provides a mechanism that will prevent substitution for an element declaration or type definition. That is, an element declaration may declare one or more of the following:

1. An instance of this element declaration may not substitute an extended type.
2. An instance of this element declaration may not substitute a restricted type.
3. An instance of this element declaration may not substitute another element.

These restriction mechanisms are very useful in instances; they allow restriction of content models down to exact types and elements. However, in shared data models, they limit reuse and customization options, in opposition to [Principle 14].

[Rule 6-26] (REF, EXT)

Within the schema, if an element declaration carries the attribute `block`, it MUST set the value for the attribute to the empty string.

[Rule 6-27] (REF, EXT)

Within the schema, if a complex type definition carries the attribute `block`, it MUST set the value for the attribute to the empty string.

[Rule 6-28] (REF, SUB, EXT)

Within the schema, if the document element `xsd:schema` carries the attribute `blockDefault`, it MUST set the value for the attribute to the empty string.

Rationale

Restriction of substitution options reduces capacity for reuse; thus, it is forbidden within NIEM-conformant schemas In particular, setting the `block` value at the schema level complicates understanding of component definitions.

6.1.10 Final Value Restrictions

XML Schema provides the capability for type definitions and elements to declare a **final** value. This value prevents the creation of derived components. In shared data models, this capability limits reuse and customization options, in opposition to [Principle 14].

[Rule 6-29] (REF, SUB)

Within the schema, if a simple type definition carries the attribute `final`, it MUST set the value for the attribute to the empty string.

[Rule 6-30] (REF, SUB)

Within the schema, if a complex type definition carries the attribute `final`, it MUST set the value for the attribute to the empty string.

[Rule 6-31] (REF, SUB)

Within the schema, if an element declaration carries the attribute `final`, it MUST set the value for the attribute to the empty string.

[Rule 6-32] (REF, SUB, EXT)

Within the schema, if the document element `xsd:schema` carries the attribute `finalDefault`, it MUST set the value for that attribute to the empty string.

Rationale

Restriction of derivation options reduces capacity for reuse and so is forbidden within reference and subset schemas. As well, the use of finalDefault complicates understanding of schemas, so it is only allowed in constraint schemas.

6.1.11 Default Value Restrictions

XML Schema provides the capability for element and attribute declarations to provide default values when XML instances using those components do not provide values.

[Rule 6-33] (REF, SUB, EXT, CON)

Within the schema, any element `xsd:element` SHALL NOT carry the attribute `default`.

[Rule 6-34] (REF, SUB, EXT, CON)

Within the schema, any element `xsd:attribute` SHALL NOT carry the attribute `default`.

Rationale

The use of default values means that the act of validating a schema will insert a value into an XML instance where none existed prior to schema validation. Schema validation is for rejection of invalid instances, not for modifying instance content, as specified in [Principle 4].

6.2 `xsd:schema` Document Element

The features of XML Schema allow for flexibility of use for many different and varied types of implementation. NIEM requires consistent use of these features.

[Rule 6-35] (REF, SUB, EXT, CON)

Within the schema, the document element `xsd:schema` MUST carry the attribute `targetNamespace`.

[Rule 6-36] (REF, SUB, EXT, CON)

Within the schema, the value of the required attribute `targetNamespace` on the document element `xsd:schema` MUST match the production `<absolute-URI>` as defined by **[RFC3986]**.

Rationale

Schemas without defined namespaces provide definitions that are ambiguous, in that they are not universally identifiable.

Absolute URIs are the only universally meaningful URIs. URIs include both URLs and URNs. Finding the target namespace using standard XML Base technology is complicated and not specified by XML Schema. Relative URIs are not universally identifiable, as they are context-specific.

Discussion

The document element `xsd:schema` may contain optional attributes `attributeFormDefault` and `elementFormDefault`. The values of these attributes are immaterial to a NIEM-conformant schema, as each attribute defined by a NIEM-conformant schema must be defined at the top level and so must be qualified with the target namespace of its declaration.

[Rule 6-37] (REF, SUB, EXT, CON)

Within the schema, the document element `xsd:schema` MUST carry the attribute `version`.

[Rule 6-38] (REF, SUB, EXT, CON)

Within the schema, the value of the required attribute `version` on the document element `xsd:schema` MUST NOT be an empty string.

Rationale

It is very useful to be able to tell one version of a schema from another. Apart from the use of namespaces for versioning, it is sometimes necessary to release multiple versions of schema documents. Such use might include:

- Subset schemas and constraint schemas
- Error corrections or bug fixes
- Documentation changes
- Contact information updates

In such cases, a different value for the `version` attribute implies a different version of the schema. No specific meaning is assigned to specific version identifiers.

Note that some of the above uses for the version attribute are not employed in management of NIEM Core and domain schemas. An author of an application schema or exchange may use the `version` attribute for these purposes within their schemas.

6.3 Namespace Imports

XML Schema requires that namespaces used in external references be imported using the `xsd:import` element. The `xsd:import` element appears as an immediate child of the `xsd:schema` element. A schema must import any namespace which

1. Is not the local namespace, and
2. Is referenced from the schema.

The behavior of import statements is not necessarily intuitive. In short, the import introduces namespace into the schema in which the import appears; it has no transitive effect. If the namespaces of an import statement are not referenced from the schema, then the import statement has no effect. The import statement cannot be used to direct schema locations for schemas not referenced from the schema performing the import. The schema location directed by the import element may be overridden by user directive at the parser, or by being overridden by import elements from other schemas.

Imports of namespaces should be made as uniform as possible; all schemas in a schema set should agree on what schema location goes with a particular namespace. Otherwise, behavior may be dependent on the behavior of the parser and the order of components in instance documents.

6.3.1 `xsd:import` Element Restrictions

[Rule 6-39] (REF, SUB, EXT)

Within the schema, the element `xsd:import` MUST carry the attribute `namespace`.

[Rule 6-40] (REF, SUB, EXT)

Within the schema, the value of the required attribute `namespace` owned by the element `xsd:import` MUST match the production `<absolute-URI>` as defined by **[RFC3986]**.

Rationale

An import that does not specify a namespace is enabling reference to non-namespaced components. NIEM requires that all components have a defined namespace. It is important that the namespace declared by a schema be universally defined and unambiguous. Use of the standard XML Base for processing is not specified by XML Schema; thus it is not supported here.

[Rule 6-41] (REF, SUB, EXT)

Within the schema, the element `xsd:import` MUST carry the attribute `schemaLocation`.

Rationale

An import that does not specify a schema location gives no clue to processing applications as to where to find an implementation of the namespace. Even though such a provided schema location may be overridden, it is important that an initial default be provided for processing.

[Rule 6-42] (REF, SUB, EXT)

Within the schema, the value of the required attribute `schemaLocation` carried by the element `xsd:import` MUST match either the production `<absolute-URI>` or the definition of "*relative-path reference*," as defined by **[RFC3986]**.

Rationale

The default value may be specified either as absolute or relative URIs. Since URNs are not resolvable, they are inappropriate for use in `schemaLocation`. The requirement for conformance to "*relative-path reference*" is required to avoid the more obscure syntax of "*network-path reference*" and the system-specific "*absolute-path reference*."

[Rule 6-43] (REF, SUB, EXT)

Within the schema, the value of the required attribute `schemaLocation` carried by the element `xsd:import` MUST be resolvable to a XML schema document file that is valid according to **[XMLSchemaStructures]** and **[XMLSchemaDatatypes]**.

Rationale

The XML Schema specification requires that the object imported via `xsd:import` must be a schema document. This rule reinforces that requirement.

Discussion

Note that relative URI references are dereferenced from the location of the schema document performing the import, not from the location of an instance or other schema. Although NIEM distribution schemas use only relative URI references, that need not be the case for other NIEM-conformant schemas.

6.3.2 Including XML Content From Other Namespaces

Within an XML Schema document, there are several mechanisms to include XML content that is not from the XML or XML Schema namespaces. Those mechanisms are:

1. Carrying attributes from other than the XML or XML Schema namespaces on an element in the XML Schema namespace.

   By the rules of XML Schema, any element may have attributes that are from other namespaces. These attributes do not participate in validation but may carry information useful to tools that process schemas.

2. Adding content to the elements `xsd:appinfo` and `xsd:documentation`.

   XML Schema allows arbitrary XML content to be included within annotations. Such XML does not participate in validation but may communicate useful information to schema readers or processors.

NIEM requires all such XML content to be "schema-valid." That is, it must have a schema, and it must validate against that schema. The schemas must be introduced via `xsd:import` elements within the schema in which the content is used. This is for two reasons:

1. Some tools require imports of namespaces used within schemas and validate against those schemas.
2. The definition and the validity of content within schemas should be clear.

[Rule 6-44] (REF, SUB, EXT)

Within the schema, when a namespace other than the XML namespace or the XML Schema namespace is used, it MUST be imported into the schema using the `xsd:import` element.

Rationale

This rule ensures that used namespaces have recognizable defining sources and that they will cooperate with existing tools.

[Rule 6-45] (REF, SUB, EXT)

Within the schema, when a namespace other than the XML namespace or the XML Schema namespace is used, its content MUST be valid with respect to the schema imported for that namespace.

Rationale

XML Schema does not address the schema-validity of content used for annotations or attributes on schema components. This rule ensures that content used in such a manner is schema-valid. This encourages interoperable data definitions and schema documents.

6.4 Annotations

Annotations in XML Schema "provide for human- and machine-targeted annotations of schema components." **[XMLSchemaStructures]** The two types: human-targeted and machine-targeted, are kept separate by the use of two separate container elements defined by XML Schema: `xsd:documentation` and `xsd:appinfo`.

[Rule 6-46] (REF, EXT)

Within the schema, an element SHALL have at most one instance of an element `xsd:annotation` as an immediate child.

Rationale

XML Schema allows annotations to be added to components in a fairly loose manner: there may be multiple annotations, each of which may have multiple `documentation` or `appinfo` elements. This flexibility in the syntax provides no additional expressivity but does complicate processing, so it is forbidden in NIEM.

6.4.1 Human-Readable Documentation

XML Schema describes the content of `xsd:documentation` elements as "user information." This information is targeted for reading by humans. The XML Schema specification does not say what form human-targeted information should take. Within NIEM, user information is plain text with no formatting or XML structure.

[Rule 6-47] (REF, EXT)

Within the schema, the content of the `xsd:documentation` element that constitutes the data definition of a component MUST be character information items as specified by **[XMLInfoSet]**.

Rationale

According to the XML Schema specification, the content of `xsd:documentation` elements is intended for human consumption, whereas other structured XML content is intended for machine consumption. Therefore, the `xsd:documentation` element MUST NOT contain structured XML data. As such, any XML content appearing within a

documentation element is in the context of human-targeted examples and should be escaped using `<` and `>`. This rule also prohibits comments within documentation elements.

See **[SchemaForXMLSchema]**, the schema for XML Schema, as an example of documentation elements containing properly escaped XML elements.

XML comments are not XML Schema constructs and are not specifically associated with any schema-based components. As such, comments are not considered semantically meaningful by NIEM and may not be retained through processing of NIEM schemas.

[Rule 6-48] (REF, SUB, EXT)

XML comments SHALL not be used for persistent information about constructs within the schema.

Rationale

Since XML comments are not associated with any specific XML Schema construct, there is no standard way to interpret comments. As such, comments should be reserved for internal use, and XML Schema annotations should be preferred for meaningful information about components. NIEM specifically defines how information should be encapsulated in NIEM-conformant schemas via `xsd:annotation` elements.

6.4.2 Machine-Readable Annotations

XML Schema provides special annotations for support of automatic processing. The XML Schema specification provides the element `xsd:appinfo` to carry such content and does not specify what style of content they should carry. In NIEM, `xsd:appinfo` elements carry structured XML content.

[Rule 6-49] (REF, EXT)

Within the schema, any immediate child of an `xsd:appinfo` element SHALL be an element information item or a comment information item.

Rationale

Application information elements are intended for *automatic processing*; thus they should contain machine-oriented data, XML.

[Rule 6-50] (REF, EXT)

Within the schema, any element that is an immediate child of an `xsd:appinfo` element SHALL be in a namespace.

Rationale

Use of default namespace is allowed, but content has to have a real namespace, and namespaces must be declared. The XML namespaces specification includes the concept of content not in a namespace. Non-namespaced data runs counter to the principle of distinctly identifiable data definitions.

[Rule 6-51] (REF, EXT)

Within the schema, an element in the XML Schema namespace MUST NOT occur as a descendant of any element `xsd:appinfo`.

Rationale

NIEM-conformant schemas are designed to be very easily processed. Although uses of XML Schema elements as content of `xsd:appinfo` elements could be contrived, it is not current practice and could seriously complicate the authoring of schema validators and processors, such as XSLT, which may evaluate XML elements by their namespaces and names. Forbidding the use of XML Schema elements outside valid uses of schema will simplify such processing.

6.5 Type Definitions

XML Schema provides a variety of ways to define new types. This section covers the NIEM restrictions on defining complex types, with both simple and complex content.

6.5.1 Complex Type Definitions

XML Schema provides a large amount of flexibility in the creation of complex types. NIEM narrows the schema capability to a smaller set of constructs.

Note that rules on prohibited constructs (Section 6.1.6.1: No Anonymous Type Definitions, above) forbid defining complex types as local types. All complex type definitions must be top-level, named components.

XML Schema makes a distinction between complex types with simple content versus complex types with complex content. Complex types with simple content (CSCs) have content that is not allowed to contain XML elements. Complex types with complex content (CCCs) have content that does contain XML elements. Since mixed content is prohibited in NIEM by [Rule 6-1], all NIEM-conformant complex types are either CSCs or CCCs.

[Rule 6-52] (REF, SUB, EXT)

Within the schema, the element `xsd:complexType` MUST have as an immediate child either the element `xsd:complexContent` or the element `xsd:simpleContent`.

Rationale

XML Schema provides shorthand to defining complex content of a complex type, which is to define the complex type with immediate children that specify elements, or other groups, and attributes. In the desire to normalize schema representation of types and to be explicit, NIEM forbids the use of that shorthand.

6.5.2 Simple Content (CSC) Restrictions

Within a NIEM-conformant schema, a complex type with simple content (CSC) can be created in one of two ways:

1. By extension of an existing CSC.
2. By extension of an existing simple type.

Both of these methods use the element `xsd:extension`.

[Rule 6-53] (REF)

Within the schema, the element `xsd:simpleContent` MUST have as an immediate child the element `xsd:extension`.

Rationale

This rule ensures that the definition of a CSC will use the XML Schema extension facility. This allows for the above cases while disallowing much more complicated syntactic options available in XML Schema.

Note that the applicability of the above rule allows for use of `xsd:restriction` within `xsd:simpleContent` in subset schemas, extension schemas, and exchange schemas.

Although the two above methods have similar syntax, there are subtle differences. NIEM's conformance rules ensure that any complex type has the necessary attributes for representing IDs, metadata, and link metadata. So case 1 does not require adding these attributes, as they are guaranteed to occur in the base type.

However, in case 2, in which a new complex type is created from a simple type, the attributes for complex types must be added. This is done by reference to the attribute group `structures:SimpleObjectAttributeGroup`.

[Rule 6-54] (REF, SUB, EXT)

Within the schema, given an element `xsd:simpleContent` with a child `xsd:extension` owning an attribute `base`, if the attribute `base` has a value that resolves to the name of a simple type, then the element `xsd:extension` MUST have an immediate child element `xsd:attributeGroup`.

[Rationale]

This rule ensures that a CSC that is created as an immediate extension of a simple type adds the attributes required for specific NIEM linking mechanisms. The attribute group is required to be `structures:SimpleObjectAttributeGroup` by [Rule 6-59].

This creates a pattern for CSC definition as follows:

Figure 6-1: Example of CSC derived from a simple type

```
<xsd:complexType name="PercentageType">
  ...
  <xsd:simpleContent>
    <xsd:extension base="nc:PercentageSimpleType">
      <xsd:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

6.5.3 Complex Content (CCC) Restrictions

Within a reference schema, a complex type with complex content (CCC) can be created in one of two ways:

1. By extension of an existing complex type (CCC or CSC).
2. By extension of the type `structure:ComplexObjectType`.

Both of these methods use the element `xsd:extension`. Within extension schemas, exchange schemas, and subset schemas, the use of `xsd:restriction` to create complex types with complex content is also allowed.

[Rule 6-55] (REF)

Within the schema, the element `xsd:complexContent` MUST have as an immediate child the element `xsd:extension`.

Rationale

NIEM does not support, as conformant, the use of complex type restriction. NIEM defines a language, in which specific content is allowed. It does not specify messages that forbid content. Such restrictions may be performed in nonconformant schemas or within constraint schemas or other artifacts of constraint.

Note that XML Schema requires use of the attribute `base` on `xsd:extension`.

Note also that the applicability allows for the use of restriction in subset schemas, extension schemas, exchange schemas, and constraint schemas.

The `xsd:extension` element says that the type under definition is an extension of another type. That type must be limited to those used with NIEM.

[Rule 6-56] (REF, SUB, EXT)

Within the schema, given an element `xsd:complexContent` with a child `xsd:extension` owning an attribute `base`, the attribute `base` MUST have a value that resolves to the name of one of the following:

1. The type `structures:ComplexObjectType`.
2. The type `structures:MetadataType`.
3. The type `structures:AugmentationType`.

4. A complex type that is a NIEM-conformant component.

[Rationale]

This rule ensures that a CCC has well-defined ancestry. In turn, this ensures that every CCC has well-defined semantics.

[Rule 6-57] (EXT)

Within the schema, given an element `xsd:complexContent` with a child `xsd:restriction` owning an attribute `base`, the attribute `base` MUST have a value that resolves to the name of a complex type that is a NIEM-conformant component.

[Rationale]

This ensures that a CCC defined through restriction has well-defined semantics.

6.6 Additional Definitions and Declarations

XML Schema provides a variety of ways to declare and define elements and attributes.

6.6.1 Element Declarations

Within NIEM-conformant schemas, elements may be declared as abstract. Element declarations must be at the top level, as rules in other sections prohibit the use of local elements. Elements may be defined without a type, but any element declaration that has no type must be declared abstract by [Rule 6-9], which forbids anonymous type definitions.

Within an element declaration, the attributes `fixed`, `nillable`, and `substitutionGroup` may be used as per the XML Schema specification. The attribute `form` is irrelevant to NIEM, as NIEM-conformant schemas may not contain local element definitions by [Rule 6-14].

Element uses (element declarations acting as particles) must reference top-level named elements. In an element use, NIEM allows any values for the XML Schema properties “max occurs” and “min occurs.”

Based on a variety of user requirements, all elements in the NIEM 2.0 schemas are defined to allow a nil value. For example, the following XML instances are permitted in NIEM-conformant instances:

```
<nc:ActivityDate></nc:ActivityDate>
```

OR

```
<nc:ActivityDate/>
```

Nil value allowance or restriction is only significant to elements of nontextual types (e.g., dates and numeric values) and elements of text types that have restricted value space (e.g., code). This is because an unrestricted text typed element always contains the empty string ("") in its value space. However, for numeric values and restricted text type elements, NIEM allows users to tighten constraints as required in IEPDs by resetting `nillable="false"`.

6.6.2 Attribute Declarations

Attribute declarations must be declared with a type by [Rule 6-10], which forbids anonymous type definitions for attributes.

Within an attribute declaration, the attribute `fixed` may be used as per the XML Schema specification. Within an attribute declaration, the attribute `form` is irrelevant to NIEM, as NIEM-conformant schemas may not contain local attribute declarations.

Attribute uses (attribute declarations acting as particles) must be uses of top-level named attributes. NIEM-conformant schemas may not define local named attributes within type definitions. Within an attribute use, the attributes `fixed` and `use` may be used as per the XML Schema specification.

6.6.3 Attribute Group Definitions

In NIEM-conformant schemas, use of attribute groups is restricted. The only attribute group that plays a part in NIEM-conformant schemas is `structures:SimpleObjectAttributeGroup`. This attribute group provides the attributes necessary for IDs, metadata, and link metadata.

[Rule 6-58] (REF, SUB, EXT)

> Within the schema, any occurrence of the element `xsd:attributeGroup` MUST own an attribute `ref`.

[Rationale]

> The only attribute group used in NIEM-conformant schemas is `structures:SimpleObjectAttributeGroup`, as established by rules [Rule 6-59] and [Rule 7-39]. Therefore, NIEM-conformant schemas do not define additional attribute groups.

[Rule 6-59] (REF, SUB, EXT)

> Within the schema, the attribute `ref` owned by any element `xsd:attributeGroup` MUST have a value of a qualified name (possibly using the default namespace) that SHALL resolve to the namespace for the NIEM `structures` namespace and the local name `SimpleObjectAttributeGroup`.

[Rationale]

> The only attribute group used within NIEM-conformant schemas is `structures:SimpleObjectAttributeGroup`. Therefore, within a NIEM-conformant schema, only this attribute group can be referenced.

7 Modeling Rules

NIEM provides a framework for modeling concepts and relationships as XML artifacts. The data model is implemented via XML Schema. However, XML Schema does not provide sufficient structure and constraint to enable translating from a conceptual model to a schema and then to

instances of the concepts. NIEM provides additional support for modeling concepts as schemas and provides rules for creating and connecting data that realizes those concepts.

Underlying the NIEM data model are two namespaces: the `structures` namespace and the `appinfo` namespace. These two namespaces provide schema components that serve two functions:

1. They provide support for connecting structural definitions to concepts.
2. They provide base components from which to derive structural definitions.

These namespaces are distributed with the NIEM data model content but are not themselves considered to be content of the data model. They are, instead, part of the structure on which the data model is built.

7.1 `xsd:schema` Document Element Restrictions

[Rule 7-1] (REF, EXT)

Within the schema, the document element `xsd:schema` MUST have application information `appinfo:ConformantIndicator`, with text content "`true`".

Rationale

The `appinfo:ConformantIndicator` element is how NIEM-conformant schemas indicate that they are, in fact, NIEM-conformant. Without such an indicator, conformance would have to be "guessed" by readers and processors.

[Rule 7-2] (REF, SUB, EXT, CON)

Two XML Schema documents SHALL have the same value for attribute `targetNamespace` carried by the element `xsd:schema`, if and only if they represent the same set of components.

[Rule 7-3] (REF, SUB, EXT, CON)

Two XML Schema documents SHALL have the same value for attribute `targetNamespace` carried by the element `xsd:schema`, and different values for attribute `version` carried by the element `xsd:schema` if and only if they are different views of the same set of components.

Rationale

These rules embody the basic philosophy behind NIEM's use of namespaced components: A component is uniquely identified by its class (e.g. element, attribute, type), its namespace (a URI), and its local name (an unqualified string). Any two matching component identifiers refer to the same component, even if the versions of the schemas containing each are different.

7.2 Annotations

NIEM-conformant schemas define data models for the purpose of information exchange. A major part of defining data models is the proper definition of the contents of the model. What does a component mean, and what might it contain? How should it be used? NIEM-conformant schemas contain the invariant part of the definitions for the data model. The set of definitions includes:

1. A text definition of each component. This describes what the component means. The term used in this specification for such a text definition is *data definition*.
2. The structural definition of each component. This is made up of XML Schema component definitions, along with certain application information (`appinfo`).

When possible, meaning is expressed via XML Schema mechanisms: type derivation, element substitution, specific types and structures, as well as names that are trivially parseable. Beyond that, NIEM-specific syntax must be used, as discussed in this section.

7.2.1 Human-Readable Documentation

By other rules, a schema component must contain at most one element `xsd:annotation`. An element `xsd:annotation`, in turn, contains at most elements `xsd:documentation` and `xsd:appinfo`. The content of the first element `xsd:documentation` on a component is the data definition for the component.

[Rule 7-4] (REF, EXT)

Within the schema, any element `xsd:complexType` MUST be a documented component.

[Rule 7-5] (REF, EXT)

Within the schema, any element `xsd:simpleType` MUST be a documented component.

[Rule 7-6] (REF, EXT)

Within the schema, any element `xsd:element` that is an immediate child of an element `xsd:schema` MUST be a documented component.

[Rule 7-7] (REF, EXT)

Within the schema, any element `xsd:attribute` that is an immediate child of an element `xsd:schema` MUST be a documented component.

[Rule 7-8] (REF, EXT)

Within the schema, any element `xsd:enumeration` MUST be a documented component.

[Rule 7-9] (REF, EXT)

Within the schema, the document element `xsd:schema` MUST be a documented component.

Note that [Rule 5-4] applies **[ISO 11179 Part 4]** definition rules to documented components.

[Rule 7-10] (REF, EXT)

Words or synonyms for the words within a data element definition SHALL NOT be reused as terms in the corresponding component name if those words dilute the semantics and understanding of, or impart ambiguity to, the entity or concept that the component represents.

[Rule 7-11] (REF, EXT)

An object class SHALL have one and only one associated semantic meaning (i.e., a single word sense) as described in the definition of the component that represents that object class.

[Rule 7-12] (REF, EXT)

An object class SHALL NOT be redefined within the definitions of the components that represent properties or subparts of that entity or class.

Rationale

Data definitions should be concise, precise, and unambiguous without embedding additional definitions of data elements that have already been defined once elsewhere (such as object classes). **[ISO 11179 Part 4]** says that definitions should not be nested inside other definitions. Furthermore, a data dictionary is not a language dictionary. It is acceptable to reuse terms (object class, property term, and qualifier terms) from a component name within its corresponding definition to enhance clarity, as long as the requirements and recommendations of **[ISO 11179 Part 4]** are not violated. This further enhances brevity and precision.

[Rule 7-13] (REF, EXT)

A data definition SHALL NOT contain explicit representational or data typing information such as number characters, type of characters, etc., unless the very nature of the component can be described only by such information.

Rationale

A component definition is intended to describe semantic meaning only, not representation or structure. How a component with simple content is represented is indicated through the representation term and further refined through constraints.

Figure 7-1: A definition that describes mathematical representation

```
<xsd:element name="AngularMinuteValue" type="nc:AngularMinuteType"
             nillable="true">
  <xsd:annotation>
    <xsd:documentation>
      A value that specifies a minute of a degree. The value comes
      from a restricted range of 0 (inclusive) to 60 (exclusive).
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

In Figure 7-1, above, the component definition contains representational information because the component is mathematical and therefore requires such. In Figure 7-2, below, the definition is incorrect and states unnecessary representational information about the data element. `nc:PersonSSNIdentification` is not a social security number (SSN); it is a complex element (type `nc:IdentificationType`) that contains a SSN identifier as well as other properties that describe a person's SSN identifier (such as issue date, issue authority, etc.). The phrase "9-digit" is incorrect and unnecessary because it applies only to the SSN identifier and should be applied as a length or pattern constraint on the identifier only.

Figure 7-2: A definition that describes syntactic representation

```
<xsd:element name="PersonSSNIdentification" type="nc:IdentificationType">
  <xsd:annotation>
    <xsd:documentation>
      A social security number that references a person; a 9-digit
      numeric identifier assigned to a living person by the United
      States Social Security Administration.
    </xsd:documentation>
  </xsd:annotation>
</xsd:element>
```

[Rule 7-14] (REF, EXT)

A component definition SHALL begin with a standard opening phrase that depends on the class of the component per Table 7-1: Standard Opening Phrases:

Table 7-1: Standard Opening Phrases

Component Class	Definition opening phrase
Abstract element	"A data concept for a…"
Association element	"A relationship…"

Component Class	Definition opening phrase
Association type	"A data type for a relationship..."
Augmentation element	"Supplements…"
Augmentation type	"A data type that supplements..."
Metadata element	Either "Metadata about..." or "Information that further qualifies..."
Metadata type	"A data type for metadata about..." or "A data type for information that further qualifies..."
Element with a date representation term	"A date..."
Element with a quantity representation term	"A (optional adjective) count/number of..."
Element with an image representation term	"A(n) (optional adjective) image/picture/photograph of..."
Element with an indicator representation term	"True if...; false otherwise/if..."
Element with an identification representation term	"A(n) (optional adjective) identification..."
Element with an ID representation term	"An identifier..."
Element with a status representation term	"A(n) (optional adjective) status/state of..."
Element with a name representation term	"A name of..."

Component Class	Definition opening phrase
Element with a category text representation term	"A kind of..."
Element with a description text representation term	"A description of..."
Other element	"A(n)..."
Other type	"A data type for a(n)..."

Rationale

A standard opening phrase based on component class helps to ensure consistent definitions that appropriate for the type of component item being defined. These opening phrases also provide a cue that facilitates recognition of the particular kind of component.

7.2.2 Machine-Readable Annotations

XML Schema provides *application information* schema components to provide for automatic processing and machine-readable content for schemas. NIEM utilizes application information to convey information that is outside schema definition and outside human-readable text definitions. NIEM uses application information to convey high-level data model concepts and additional syntax to support the NIEM conceptual model and validation of NIEM-conformant XML instances.

NIEM defines a single namespace that holds components for use in NIEM-conformant schema application information. This namespace is referred to as the `appinfo` namespace.

[Definition: appinfo namespace]

The **appinfo namespace** is the namespace represented by the URI "`http://niem.gov/niem/appinfo/2.0`".

The `appinfo` namespace defines elements which provide additional semantics and syntactic guidelines for components built by NIEM-conformant schemas.

[Rule 7-15] (REF, EXT)

The schema SHALL import the `appinfo` namespace.

Rationale

For uniformity, all NIEM-conformant schemas must import the `appinfo` namespace.

[Definition: application information]

A component is said to have **application information** of some element **E** when the root element that defines the component has an immediate child element

`xsd:annotation`, which has an immediate child element `xsd:appinfo`, which has as an immediate child the element **E**.

If a component is described as "having application information," this means that the application information elements under consideration are children of the element which defines the component.

The majority of uses of application information from the `appinfo` namespace are described in the modeling rules for the specific component.

7.2.2.1 Deprecation

The `appinfo` schema provides a construct for indicating that a construct is deprecated. A deprecated component is one whose use is not recommended. A deprecated component is kept in a schema for support of older versions but should not be used in new efforts. A deprecated component will be removed, replaced, or renamed in a later edition of a schema.

[Definition: deprecated component]

In a particular NIEM-conformant namespace, a **deprecated component** is one whose use is not recommended, yet which is maintained in the schema for compatibility with previous versions of the namespace.

[Rule 7-16] (REF, EXT)

A component that is deprecated SHALL be indicated as such by the component having application information `appinfo:Deprecated`, with an attribute `value` with a value of `true`.

Rationale

Deprecation can allow version management to be more consistent; versions of schema may be incrementally improved without introducing validation problems and incompatibility. As XML Schema lacks a deprecation mechanism, NIEM defines such a mechanism.

7.2.2.2 Indicating Conformance

The element `appinfo:ConformantIndicator` is used for two purposes:

1. To indicate that a schema is conformant or that it represents a conformant namespace.
2. To indicate that an imported schema is not conformant or represents a nonconformant namespace.

The specific rules concerning this element appear in Section 7.1, `xsd:schema` Document Element Restrictions, and Section 7.7, Using External Schemas.

7.2.2.3 Bases of Derived Components

The `appinfo` namespace provides an annotation for indicating the base of a derived component. This is expressed via the `appinfo:Base` application information.

[Rule 7-17] (REF, EXT)

Within the schema, the element `appinfo:Base` MAY be used in one of the following ways:

1. By a type definition, to indicate the base type, or `structures:Object` or `structures:Association`.
2. By an element declaration, to indicate the base element.

The element `appinfo:Base` SHALL NOT be used for any other purpose.

Rationale

The `appinfo:Base` element is required to clarify semantics of types as object or association types, when such derivation is not otherwise derivable from the component definitions.

[Rule 7-18] (REF, EXT)

Within the schema, the element `appinfo:Base` SHALL indicate, by namespace and name, one of the following:

1. A NIEM-conformant schema component.
2. `structures:Object`.
3. `structures:Association`.

[Rule 7-19] (REF, EXT)

Within the schema, an attribute `appinfo:namespace` owned by an element `appinfo:Base` SHALL have a value of either of the following:

1. A namespace which is the target namespace of a NIEM-conformant schema.
2. The `structures` namespace.

[Rule 7-20] (REF, EXT)

Within the schema, an element `appinfo:Base` that does not own an attribute `appinfo:namespace` SHALL refer to the target namespace of the schema in which it is used.

[Rule 7-21] (REF, EXT)

Within the schema, an element `appinfo:Base` SHALL own an attribute `appinfo:name`.

[Rule 7-22] (REF, EXT)

Within the schema, if an element `appinfo:Base` indicates a NIEM-conformant namespace, then the value of the attribute `appinfo:name` owned by the element `appinfo:Base` SHALL indicate a schema component in the indicated namespace.

[Rule 7-23] (REF, EXT)

Within the schema, if an element `appinfo:Base` indicates the `structures` namespace, then the value of the attribute `appinfo:name` owned by the element `appinfo:Base` SHALL have a value of one of the following:

1. `structures:Object`.
2. `structures:Association`.
3. A schema component defined by the `structures` schema.

Rationale

Together, this set of rules establishes the element `appinfo:Base` as a reference to either a NIEM-conformant schema component or to a special NIEM component, which acts as the base for the containing schema component.

7.2.2.4 Application of Constructs

NIEM-conformant schemas provide capability for modeling beyond that provided by basic XML Schema. Two methods made available by NIEM are augmentations and metadata. Both of these methods create schema components that may be applied to types in specific ways. The applicability of these components to types is expressed with the `appinfo:AppliesTo` element.

[Rule 7-24] (REF, EXT)

Within the schema, the element `appinfo:AppliesTo` MAY be used in any of the following ways:

1. To indicate a base type to which an augmentation may be applied.
2. To indicate a base type to which a metadata type may be applied.

The element `appinfo:AppliesTo` SHALL NOT be used for any other purpose.

Rationale

The `appinfo:AppliesTo` element is required to express constraints beyond those available within XML Schema. Use of this element allows advanced processing of instances and schemas for type safety.

[Rule 7-25] (REF, EXT)

Within the schema, the element `appinfo:AppliesTo` SHALL indicate a schema component by namespace and name.

[Rule 7-26] (REF, EXT)

Within the schema, an attribute `appinfo:namespace` owned by an element `appinfo:AppliesTo` SHALL indicate the namespace of the type to which `appinfo:AppliesTo` refers. The indicated namespace SHALL be defined by a NIEM-conformant schema.

[Rule 7-27] (REF, EXT)

Given that the element `appinfo:AppliesTo` refers to a type, the applicability described by the element SHALL be understood to be the indicated type or a type transitively derived from the indicated type.

[Rule 7-28] (REF, EXT)

Within the schema, an element `appinfo:AppliesTo` that does not carry an attribute `appinfo:namespace` SHALL refer to the target namespace of the schema in which it is used.

[Rule 7-29] (REF, EXT)

Within the schema, an element `appinfo:AppliesTo` SHALL carry an attribute `appinfo:name`. The value of this attribute SHALL indicate the local name of a schema component within the namespace specified by the element.

Rationale

Together, this set of rules establishes the element `appinfo:AppliesTo` as a reference to a NIEM-conformant schema component to which a NIEM construct may be applied.

7.2.2.5 Targets of References

NIEM provides references to avoid problems occurring when only XML element containment is available. The `appinfo:ReferenceTarget` element specifies the type to which a reference element may be applied.

[Rule 7-30] (REF, EXT)

Within the schema, the element `appinfo:ReferenceTarget` SHALL identify the XML Schema type definition of an element information item to which an instance of a reference element may validly refer. The element `appinfo:ReferenceTarget` SHALL NOT be used for any other purpose.

Rationale

This describes the meaning of a reference target. The term *type definition* is as used in **[XMLSchemaStructures]**, in the PSVI (post-schema-validation infoset) definition for an element information item. The element `appinfo:ReferenceTarget` is required to express the type of referenced content. XML Schema does not provide this level of type safety.

[Rule 7-31] (REF, EXT)

Within the schema, a reference element MUST have at most one instance of the element `appinfo:ReferenceTarget`.

Rationale

Content elements in XML Schema may have at most one type. This rule ensures that reference elements follow the same pattern.

[Rule 7-32] (REF, EXT)

Within the schema, the element `appinfo:ReferenceTarget` SHALL indicate a type definition schema component, by namespace and name.

[Rule 7-33] (REF, EXT)

Within the schema, an attribute `appinfo:namespace` carried by an element `appinfo:ReferenceTarget` SHALL indicate the namespace of the referenced schema component. The indicated namespace SHALL be defined by a reference or extension schema.

[Rule 7-34] (REF, EXT)

Within the schema, an element `appinfo:ReferenceTarget` that does not carry an attribute `appinfo:namespace` SHALL refer to the target namespace of the schema in which it is used.

[Rule 7-35] (REF, EXT)

Within the schema, an element `appinfo:ReferenceTarget` SHALL carry an attribute `appinfo:name`. The value of this attribute SHALL indicate the local name of a type definition schema component within the namespace specified by the element.

Rationale

Together, this set of rules establishes the element `appinfo:ReferenceTarget` as a reference to a NIEM-conformant type definition schema component that a reference element instance may reference.

7.3 Simple Type Definitions

NIEM places very few restrictions on the definition of simple types in conformant schemas. The use of lists should be reserved for cases where the data is fairly uniform.

[Rule 7-36] (REF, SUB, EXT)

Within the schema, a simple type definition that uses `xsd:list` SHOULD NOT be defined if any member of the list requires a property or metadata that is different than other members of the list. All members of the list SHOULD have the same metadata, and should be related via the same properties.

Rationale

The members of a list are not individually addressable by NIEM metadata techniques. The members are also not individually addressable by properties; a property has a value of all the members of the list. NIEM provides no method for individually addressing a member of a list. If an individual member of a list needs to be marked up in a manner

different than other members of the list, the use of individual elements may be preferred to the definition of a list simple type.

7.4 Complex Type Definitions

Under XML Schema rules, a CCC (complex type with complex content) may not be the base type of a CSC (complex type with simple content), and a CSC may not be a base for a CCC. Therefore, NIEM defines one pattern for defining a CCC and a different pattern for defining a CSC. These patterns supply common base definitions that will be provided for CSCs and CCCs. These patterns are established by the rules for use of `xsd:extension` in `xsd:complexContent` and `xsd:simpleContent` elements. The relevant rules may be found in Section 6.5.2, Simple Content (CSC) Restrictions, and Section 6.5.3, Complex Content (CCC) Restrictions.

[Rule 7-37] (REF, SUB, EXT)

Within the schema, a complex type definition SHALL be one of the following classes of types:

1. An object type.
2. A role type.
3. An association type.
4. A metadata type.
5. An augmentation type.
6. An adapter type.

Rationale

This rule establishes the classes of NIEM complex types. It is a limited set, each class with distinct semantics.

The first five types are described in subsections below. The adapter type is described in Section 7.7, Using External Schemas.

[Rule 7-38] (REF, SUB, EXT)

Within the schema, an element MUST NOT be introduced more than once into the direct content of a type definition. This applies to content acquired through extension of base types. This does not apply to a base element or derived element to one previously existing in the type definition.

Rationale

This rule ensures that sequences of elements are simple sequences. A type should not define, for example, a sequence of elements A, B, then A again. Definitions should define, instead, what elements may be included, and their cardinality. Specific orders should be expressed in instances, when necessary, by the use of the attribute `structures:sequenceID`.

7.4.1 Object Types

[Definition: object type]

In a NIEM-conformant schema, an **object type** is a complex type definition, an instance of which asserts the existence of an object. An object type represents some kind of object: a thing with its own lifespan that has some existence. The object may or may not be a physical object. It may be a conceptual object.

[Rule 7-39] (REF, EXT)

Within the schema, an object type SHALL be a complex type definition that either constitutes a NIEM-conformant component or for which there exists a NIEM-conformant component of one of the following forms:

1. Has simple content, is based on a simple type, and contains the attribute group `structures:SimpleObjectAttributeGroup`, and has application information `appinfo:Base` of `structures:Object`.
2. Has complex content, and is based on complex type `structures:ComplexObjectType`, and has application information `appinfo:Base` of `structures:Object`.
3. Is a complex type that is derived from an object type, which is defined according to this rule.

Rationale

Object types are at the core of NIEM. They are built in a uniform way, from a simple design pattern: they take one of the two "root" forms outlined above, or they are built from other object types, depending on whether they are of simple or complex content.

7.4.2 Role Types

NIEM differentiates between an object and a role of the object. The term "role" is used here to mean a function or part played by some object.

[Definition: role type]

A **role type** is a type that represents a particular function, purpose, usage, or role of an object.

The simplest way to represent a role of an object is to use an element. The following example represents the role of a person who performs an assessment:

Figure 7-3: An element definition that constitutes a role without the use of a role type

```
<xsd:element name="AssessmentPerson" type="nc:PersonType"/>
```

In many cases, there is a further need to represent characteristics and additional information associated with a role of an object. In such cases, the above element is insufficient. For

example, when a person is a driver involved in an automotive crash, the person plays the role of a `j:CrashDriver`. In the case of a crash, there is more information associated with the role of the driver than just his identity for the role. One such example would be the traffic violation code; `j:CrashDriverViolationCode` is frequently a characteristic property of a `j:CrashDriver`. For this reason, a role type, `j:CrashDriverType` is created.

A role type provides the location for information associated with an object playing a role. A role type is used instead of the base type (in this case, `nc:PersonType`). The role type holds information specific to the role but not specific to the context or the base object (the object that plays the role). Developers of NIEM-conformant schemas should create and use role types whenever they have nonpersistent information specific to a base object. Such information generally expires when the base object is no longer playing the role. Information that is persistent to the base object probably does not belong in a role type.

[Definition: `RoleOf` element]

> In a NIEM-conformant schema, a **`RoleOf` element** is a reference element whose type is the base type of the role.

Here is an example of a role type from the NIEM justice domain that uses a `RoleOf` element:

Figure 7-4: A definition of a role type

```
<xsd:complexType name="CrashPersonType">
  ...
  <xsd:sequence>
    <xsd:element ref="nc:RoleOfPersonReference" minOccurs="0"
      maxOccurs="unbounded"/>
    ...
    <xsd:element ref="j:CrashPersonInjury" minOccurs="0"
      maxOccurs="unbounded"/>
    ...
    <xsd:element ref="j:AlcoholTestResultCode" minOccurs="0"
      maxOccurs="unbounded"/>
    ...
  </xsd:sequence>
  ...
</xsd:complexType>
```

`nc:RoleOfPersonReference` is defined as "An entity of whom the role object is a function." In this example, the role object is `j:CrashPersonType` and the base type of the role object is a `nc:PersonType`, the entity of whom `j:CrashPersonType` is a function (per the definition above).

This role object represents a particular role of a person: a person involved in a vehicular crash. It refers to the person of whom this object is a role through the `nc:RoleOfPersonReference` element. It also includes additional information particular to the person's role in the crash.

Here is an example of the `CrashPerson` role type used in an instance:

Figure 7-5: A role type used in an instance

```
<j:CrashPerson>
  <nc:RoleOfPersonReference s:ref="p1">
  <j:AlcoholTestResultCode>101</j:AlcoholTestResultCode>
  <j:AlcoholTestResultQuantity>07</j:AlcoholTestResultCodeQuantity>
</j:CrashPerson>
<nc:Person s:id="p1">
  <nc:PersonBirthDate>
    <nc:Date>1966-06-06</nc:Date>
  </nc:PersonBirthDate>
  <nc:PersonName>
    <nc:PersonFullName>John Doe</nc:PersonFullName>
  </nc:PersonName>
</nc:Person>
```

[Rule 7-40] (REF, SUB, EXT)

Within the schema, any element with a name beginning with the string `RoleOf` SHALL represent a base type, of which the containing type represents a role.

Rationale

A `RoleOf` element references its corresponding base element. The `RoleOf` label on the reference element ensures that a role object is distinguishable from other objects and its link to the associated base is also distinguishable from the additional properties that are characteristic of this role or that add information.

NIEM does not require that there be only one `RoleOf` element within a single type. However, the use of multiple `RoleOf` elements may not make sense; indeed, an example of a role that references two or more base types is very difficult (if not impossible) to conceive.

An object should be a role of only a single object. However, there may be varied assertions of what object that might be or time constraints on the role. Many exchanges may wish to restrict `RoleOf` elements to a single occurrence within a type.

`RoleOf` elements are generally reference elements, targeting the base type. That is, a `RoleOf` element is usually a reference element, not a content element.

7.4.3 Association Types

Within NIEM, an association is 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.

Here is an example of an association in an XML instance:

Figure 7-6: An association in an instance

```
<nc:GuardianAssociation>
  <nc:PersonGuardianReference s:ref="p1"/>
  <nc:PersonDependentReference s:ref="p2"/>
</nc:GuardianAssociation>
<nc:Person s:id="p1">
  <nc:PersonName>
    <nc:PersonFullName>John Doe</nc:PersonFullName>
  </nc:PersonName>
</nc:Person>
<nc:Person s:id="p2">
  <nc:PersonName>
    <nc:PersonFullName>Jane Doe</nc:PersonFullName>
  </nc:PersonName>
</nc:Person>
```

This example shows an association between a guardian and a dependent. This relationship is defined by the element `nc:GuardianAssociation`, whose structure is defined by the type `nc:GuardianAssociationType`. The type defines what an association relates, but the element defines the actual meaning of the association.

An example of an association type defined by an XML Schema document follows.

Note that the NIEM Core schema in NIEM 2.0 defines a type `nc:AssociationType`, which acts as the base type for all other association types defined within NIEM Core. This is a convention adopted by the NIEM Core namespace but is not a requirement of the NDR. Implementers of NIEM-conformant schemas are not required to base association types on `nc:AssociationType`.

Figure 7-7: A definition of an association type

```
<xsd:complexType name="AssociationType">
  ...
  <xsd:complexContent>
    <xsd:extension base="s:ComplexObjectType">
      <xsd:sequence>
        <xsd:element ref="nc:AssociationBeginDate" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element ref="nc:AssociationEndDate" minOccurs="0"
            maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="GuardianAssociationType">
  ...
  <xsd:complexContent>
    <xsd:extension base="nc:AssociationType">
      <xsd:sequence>
        <xsd:element ref="nc:PersonGuardianReference" minOccurs="0"
            maxOccurs="unbounded"/>
        <xsd:element ref="nc:PersonDependentReference" minOccurs="0"
            maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:element name="GuardianAssociation" type="nc:GuardianAssociationType"
    nillable="true">
  ...
</xsd:element>
```

This schema fragment shows the definition of a generic `AssociationType`, which contains a begin and end date. It then defines a specific association type, which contains the structure required to express guardianship. This is followed by the definition of an element that expresses the semantics of the guardian relationship.

[Definition: association type]

In a NIEM-conformant schema, an **association type** is a type that establishes a relationship between objects, along with the properties of that relationship. An association type provides a structure that does not establish existence of an object but instead specifies relationships between objects.

[Definition: association]

In a NIEM-conformant schema, an **association** is an element whose type is an association type.

[Rule 7-41] (REF, EXT)

Within the schema, an association type SHALL be a complex type definition that either constitutes a NIEM-conformant component or for which there exists a NIEM-conformant component definition. The NIEM-conformant component definition SHALL have one of the following forms:

1. Has complex content, is based on the complex type `structures:ComplexObjectType`, and has application information `appinfo:Base` of `structures:Association`.
2. Is a complex type that is derived from an association type, which is defined according to this rule.

Rationale

Associations within reference schemas, extensions schemas, and exchange schemas are easily identifiable as such and have a commonly defined base type. For subset schemas, the NIEM-conformant definition may be located in a primary schema and then identified.

[Rule 7-42] (REF, SUB, EXT)

Given that an association type defines a relationship between a set of participants, within an association type definition, any element that represents a participant SHALL be a reference element.

Rationale

Associations are intended to relate objects defined elsewhere. They are not intended to carry content of participant objects.

7.4.4 Metadata Types

Within NIEM, metadata is defined as "data about data." This may include information such as the security of a piece of data or the source of the data. These pieces of metadata may be composed into a metadata type. The types of data to which metadata may be applied may be constrained.

[Definition: metadata type]

A **metadata type** describes data about data, that is, information that is not descriptive of objects and their relationships, but is descriptive of the data itself. It is useful to provide a general mechanism for data about data. This provides required flexibility to precisely represent information.

[Definition: metadata element]

Within a NIEM-conformant schema, a **metadata element** is an element whose type is a metadata type. There are specific limitations on the meaning of a metadata element in an instance; it does not establish existence of an object, nor is it a property of its containing object.

[Rule 7-43] (REF, SUB, EXT)

Within the schema, a metadata type SHALL contain elements appropriate for a specific class of data about data.

[Rule 7-44] (REF, SUB, EXT)

Within the schema, a metadata type and only a metadata type SHALL be derived directly from `structures:MetadataType`.

Rationale

A metadata type establishes a specific, named aggregation of data about data. Any type derived from `structures:MetadataType` is a metadata type. Metadata types should not be derived from other metadata types. Such metadata types should be used as is and additional metadata types defined for additional content.

[Rule 7-45] (REF, EXT)

Within the schema, a metadata type MAY have application information `appinfo:AppliesTo`, indicating the NIEM-conformant object, association, or external adapter types to which the metadata applies.

[Rule 7-46] (REF, EXT)

Within the schema, a metadata type that does not have application information `appinfo:AppliesTo` MAY be applied to any object type, association type, or external adapter type.

Rationale

Metadata may be constrained to be applicable to only specific types, or it may be defined to be applicable to any type. The source of a piece of data and the security classification of a piece of data are examples of metadata that may be considered globally applicable.

7.4.5 Augmentation Types

Builders of domains and extensions to NIEM distribution schemas need to be able to define extensions to types. However, extension of types by multiple domain schemas and extension schemas proves problematic, as it results in multiple extensions of a single type. XML Schema does not provide for multiple types of an instance; consequently, such a method results in duplication of base type content and a need to resolve "same-as" relationships between the instances of the various derived types.

Instead, it is preferable for domains and extensions to provide augmentations. These are reusable types and elements of those types, which may be added to an object class, in a single extended type, by the author of a NIEM-conformant schema. This avoids the problem of multiple extended types but allows domains and extensions to define reusable extensions.

Augmentation types such as `dom:PersonAugmentationType` (where `dom:` is a NIEM domain namespace) exist to extend NIEM Core types such as `nc:PersonType` without creating a new specialized object within the model. Augmentation types are never applied within the model to the types they are designed to augment. Doing so would restrict reusing and combining these augmentations.

Instead, augmentation should be applied within IEPDs. So in an IEPD (NOT within NIEM), base `nc:PersonType` may be extended, for example, as `my-iepd:PersonType` by adding elements `a:PersonAugmentation` and `b:PersonAugmentation`. As a result, `my-iepd:PersonType` will contain all the properties in `nc:PersonType` plus the properties in both of the elements `a:PersonAugmentation` and `b:PersonAugmentation`, which, in turn, each contain their respective sets of subelements.

All NIEM augmentation types extend the abstract type `structures:AugmentationType`. Therefore, all augmentation types automatically contain the attributes `structures:id` and `structures:metadata` for referencing and metadata, respectively. NIEM also provides the abstract element `structures:Augmentation` (of type `structures:AugmentationType`) as the common substitution group head for all augmentation elements. An augmentation element placed into this substitution group can be used in an instance wherever `structures:Augmentation` occurs in the corresponding IEPD schema. The user must follow NIEM naming conventions for augmentation component names and must place new augmentation elements into the `structures:Augmentation` substitution group. Further, if an augmentation element cannot be applied to all types in the model, then the user must document those types that the new augmentation element can be applied to using the `appinfo:AppliesTo` element.

[Definition: augmentation type]

An **augmentation type** is a complex type that provides a reusable block of data that may be added to object types or association types.

[Definition: augmentation]

An **augmentation** of a NIEM-conformant object type is a block of additional data added to an object type to carry additional data beyond that of the original object definition.

[Rule 7-47] (REF, SUB, EXT)

An augmentation type:

1. SHALL be transitively derived from `structures:AugmentationType`.
2. SHALL contain elements that represent properties to be applied to a base type.

Rationale

A base type is the type to which an augmentation is to be applied. An augmentation may be applied to any number of types. Base types are assigned by augmentation elements.

[Rule 7-48] (REF, SUB, EXT)

Within the schema, an augmentation element definition:

1. SHALL have a type that is an augmentation type.
2. SHALL use the `substitutionGroup` attribute such that it is transitively substitutable for the element `structures:Augmentation`.

An element that is not an augmentation element SHALL NOT meet either of the above criteria.

Rationale

An augmentation is trivially identifiable as such. The use of the common `structures:Augmentation` element allows message builders to optionally delay specifying augmentations to be applied to a type until runtime.

[Rule 7-49] (REF, EXT)

Within the schema, an element definition for an augmentation element MAY contain one or more instances of the element `structures:AppliesTo` as application information to specify types to which the augmentation element applies.

[Rule 7-50] (REF, EXT)

Within the schema, an element definition for an augmentation element that does not contain any instances of the element `structures:AppliesTo` MAY be applied to any object or association type.

Rationale

These rules allow schema builders to establish applicability for augmentations. An augmentation may be applicable to specific types.

Users who wish to apply an augmentation type to a given object type may do so by creating a new augmentation element, applicable to the object type.

7.5 Component Usage

[Rule 7-51] (REF, SUB, EXT)

Any type definition referenced by a component within the schema MUST be from one of the following:

1. The schema being defined.
2. A namespace imported as NIEM-conformant.
3. The XML Schema namespace.
4. The `structures` namespace.

Rationale

NIEM-conformant schemas are based on other NIEM-conformant schemas and the supporting namespaces. This simplifies processing and understanding of data.

[Rule 7-52] (REF, SUB, EXT)

Any element declaration referenced by a component within the schema MUST be from one of the following:

1. The schema being defined.

2. A namespace imported as NIEM-conformant.
3. The `structures` namespace.
4. An external namespace, in accordance with the rules for external schemas as specified by this specification.

[Rule 7-53] (REF, SUB, EXT)

Any attribute declaration referenced by a component within the schema MUST be from one of the following:

1. The schema being defined.
2. A namespace imported as NIEM-conformant.
3. The `structures` namespace.
4. The XML namespace.
5. An external namespace, in accordance with the rules for external schemas as specified by this specification.

Rationale

NIEM-conformant schemas are based on other NIEM-conformant schemas. All attributes and elements must be from NIEM-conformant schemas, the `structures` namespace, the XML namespace, or an external namespace. This applies to elements referenced for substitution groups as well. It does not apply to content of the schema (e.g., within annotations) or to the XML Schema declarations themselves. It applies only to attributes and elements referenced by the XML Schema components.

7.6 NIEM Structural Facilities

NIEM provides the structures schema that contains base types for types defined in NIEM-conformant schemas. It provides base elements to act as heads for substitution groups. It also provides attributes that provide facilities not otherwise provided by XML Schema. These structures should be used to augment XML data. The structures provided are not meant to replace fundamental XML organization methods; they are intended to assist them.

[Definition: structures namespace]

The **structures namespace** is the namespace represented by the URI "`http://niem.gov/niem/structures/2.0`".

The structures namespace is a single namespace, separate from namespaces that define NIEM-conformant data. This document refers to this content via the prefix `structures`.

[Rule 7-54] (REF, EXT)

The schema MUST import the NIEM `structures` namespace.

Rationale

For uniformity, all NIEM-conformant schemas must import the `structures` namespace.

[Rule 7-55] (REF, SUB, EXT, INS)

The schema or instance MUST use content within the NIEM `structures` namespace as specified in this document and ONLY as specified by this document.

Rationale

This rule further enforces uniformity and consistency by mandating use of the NIEM `structures` namespace as is, without modification. Users are not allowed to insert types, attributes, etc. that are not specified by this document (the NDR).

7.6.1 Sequence ID

NIEM provides the attribute `structures:sequenceID` for specification of sequential order of instances, when a complex type's defined element sequence is insufficient. A limitation of XML Schema is that control of cardinality (the number of times an element may occur in an instance) requires the use of sequences of elements. This use of `xsd:sequence` defines the elements occurring within a type in a specific order. This order may not match the desired sequential order of the represented entities.

An example would be proper names, where the natural order of the names may not appear in the same order as the sequence defined by a complex type. In this case, the structure defined by nc:PersonNameType defines a sequence of name parts, including given name followed by surname. This works well enough for Western names:

Figure 7-8: An instance of a name type

```
<nc:Person>
  <nc:PersonName>
    <nc:PersonGivenName>John</nc:PersonGivenName>
    <nc:PersonSurName>Doe</nc:PersonSurName>
  </nc:PersonName>
</nc:Person>
```

However, it does not work well for Chinese names, where the surname precedes the given name. For example, the basketball player Yao Ming has a given name of Ming and a surname of Yao. This cannot be expressed by the simple sequence used above because it lists the given name before the surname. To express the proper sequence of the data, use the `structures:sequenceID` attribute.

Figure 7-9: An instance of a name type that uses `structures:sequenceID`

```
<nc:Person>
  <nc:PersonName>
    <nc:PersonGivenName s:sequenceID="2">Ming</nc:PersonGivenName>
    <nc:PersonSurName s:sequenceID="1">Yao</nc:PersonSurName>
  </nc:PersonName>
</nc:Person>
```

Without the `structures:sequenceID` attribute, this example would create a dilemma: which name to represent correctly, and which to represent incorrectly? The `structures:sequenceID` attribute allows the schema sequence to be separated from the implied meaning.

As another example, when using a derived type, within an instance, the base type's elements occur first, followed by any elements added by extension. If those elements need to be interleaved into the existing structure for the proper meaning to be conveyed, the `structures:sequenceID` attribute is called for.

The `structures:sequenceID` attribute allows instances to express the sequential order of data relative to a parent. The order of data is as yielded by the `xsl:sort` element, which is defined by XSLT, with data-type of `xsl:number`, and order of `ascending`. Content with identical `structures:sequenceID` values has undefined order.

[Rule 7-56] (REF, SUB, EXT)

> Within the schema, a complex type definition SHALL include the attribute `structures:sequenceID` if the order of an occurrence of the type, within its parent, relative to its siblings, is meaningful and pertinent and if the schema does not specify the desired sequential order.

Rationale

> This rule indicates that, if order is meaningful and the schema will not always represent the desired order, then data modelers need to include `sequenceID` to allow the proper order to be represented in instances.

Rules on the use of `sequenceID` may be found in the rules on conformant instances in Section 8.4, Component Ordering.

7.6.2 Reference Elements

In XML instances, relationships between data objects are expressed as XML elements:

1. Data objects are expressed as XML elements.
2. XML elements contain attributes and other elements.

In this way, there is generally some implicit relationship between the outer element (the "containing" element, also known as the parent element) and the inner elements (the *contained*

elements, also known as the *child* elements). Such expression of relationships is said to be by containment.

Expression of all relationships via element containment is not always possible. Situations that cause problems include:

- Circular relationships. For example, suppose that object 1 has a relationship to object 2 and object 2 has a relationship to object 1. Expressed via containment, this relationship would result in infinite recursive descent.
- Repeated relationships. For example, suppose object 1 has a relationship to object 2 and object 3 has a relationship to object 2. Expressed via containment, this would result in a duplicate of object 2.

A method that solves this problem is the use of references. In a C or assembler, a pointer would be used. In C++, a reference might be used. In Java, a reference value might be used. The method defined by the XML standard is the use of ID and IDREF. An ID refers to an IDREF. NIEM uses this method and assigns to it specific semantics.

[Definition: reference element]

A **reference element** is an element that refers to its value by a reference attribute instead of carrying it as content.

[Rule 7-57] (REF, SUB, EXT)

Within the schema, a reference element and only a reference element SHALL be defined to be of type `structures:ReferenceType`.

Rationale

Reference elements must be of the reference type, and elements of the reference type must be reference elements. This rule ensures that users always create reference elements using `structures:ReferenceType` and cannot use `structures:ReferenceType` for any other purpose.

[Rule 7-58] (REF, SUB, EXT)

Within the schema, a complex type SHALL NOT be defined such that an instance of that type owns the attribute `structures:ref`.

Rationale

The use of references is limited to reference elements. This constrains the semantics and syntax of references within NIEM instances. Only `structures:ReferenceType` may use `structures:ref`, which is the only means for referencing within NIEM-conformant instances.

[Rule 7-59] (REF, SUB, EXT)

Within the schema, any two elements of the form

NCName

and

NCName`Reference`

where the string value of *NCName* is the same in both forms, SHALL be defined to have identical semantics. NIEM recognizes no difference in meaning between a reference element and an element that is not a reference element.

Rationale

NIEM-conformant data instances may use concrete data elements and reference elements as needed, to represent the meaning of the fundamental data. There is no difference in meaning between reference and concrete data representations. The two different methods are available for ease of representation. No difference in **meaning** should be implied by the use of one method or the other.

Assertions that indicate "included" data is intrinsic, while referenced data is extrinsic, are not valid and are not applicable to NIEM-conformant data instances and data definitions.

[Rule 7-60] (REF, EXT)

Within the schema, if both elements *NCName* and *NCName*`Reference` exist, then the `appinfo:ReferenceTarget` of any *NCName*`Reference` element MUST be the type of the element *NCName*.

Rationale

By [Rule 7-59], any such pair of elements, *NCName* and *NCName*`Reference`, will have identical semantics. This rule ensures that an *NCName*`Reference` element is documented to refer to the appropriate type (the type of the corresponding *NCName* element) and no other.

The NIEM structures schema defines `structures:ReferenceType` to require the use of an attribute `structures:ref`, which is of type `IDREF` as specified by **[XMLSchemaStructures]**. According to the rules of XML, such an attribute must contain a value that is represented by an attribute of type `ID`. In NIEM-conformant instance, the targets of IDREFs are expected to be values of the attribute `structures:id`.

The NIEM structures schema defines `structures:ReferenceType` such that it is unavailable as a base for extension or restriction.

The NIEM structures schema defines `structures:ReferenceType` such that it has an optional attribute `structures:id`. This may be used to describe additional metadata or information about the relationship described by an element of type `structures:ReferenceType`.

Within a NIEM-conformant instance, the element referenced by an attribute `structures:ref` must be of a type valid for the object of the fundamental element of the reference element. The attribute `structures:ref` is discussed in more detail in Section 8.3.

7.7 Using External Schemas

There are a variety of commonly used standards that are represented in XML Schema. Such schemas are generally not NIEM-conformant. NIEM-conformant schemas may reference components defined by these external schemas. NIEM-conformant components may be constructed from schema components that are not NIEM-conformant.

[Definition: external schema]

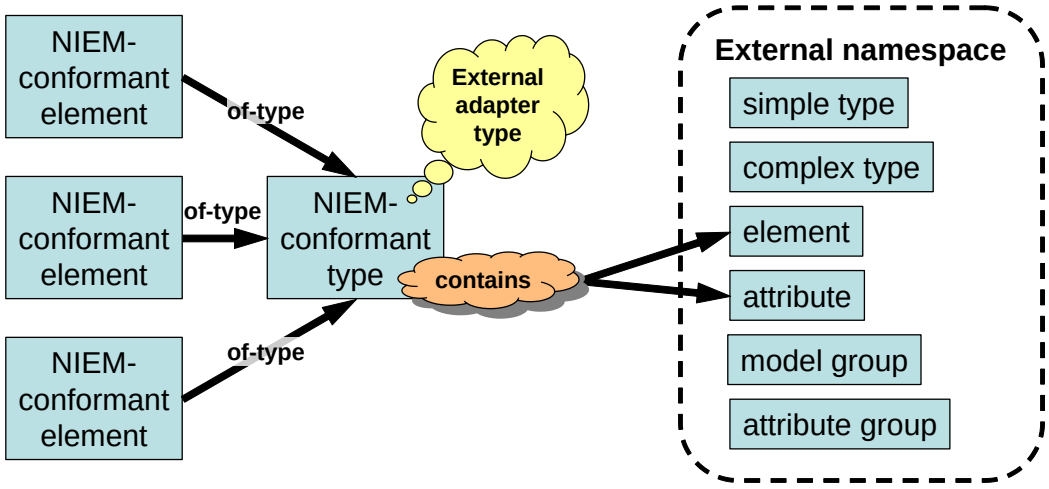
> An **external schema** is any schema that is not a supporting schema and that is not NIEM-conformant.

Note that the supporting schemas `structures` and `appinfo` are nonconformant because they define the fundamental framework on which NIEM is built. However, they are not considered external schemas because of their supporting nature and are thus excluded from this definition.

NIEM-conformant schemas may work with external schemas by creating external adapter types.

A single method is used to integrate external components into NIEM-conformant schemas: NIEM-conformant types are constructed from the external components.

Figure 7-10: Use of external components to create a NIEM-conformant type



Components defined by external schemas are called *external components*. A NIEM-conformant type may use external components in a specific way: to construct a NIEM-conformant type from external components. The goal in this method is to preserve as a single unit a set of data that embodies a single *concept* from an external standard.

For example, a NIEM-conformant type may be created to represent a bibliographic reference from an external standard. Such an object may be composed of multiple elements and types from the external standard. These pieces are put together to form a single NIEM-conformant type. For example, an element representing an author, a book, and a publisher may be included in a single bibliographic entry.

A NIEM-conformant type built from these components may be used as any other NIEM-conformant type. That is, elements may be constructed from such a type, and those elements are fully NIEM-conformant.

To construct such a component, a NIEM-conformant schema must first import an external schema.

[Rule 7-61] (REF, EXT)

Within the schema, an element `xsd:import` that imports a namespace defined by an external schema MUST have the application information `appinfo:ConformantIndicator`, with a value of `false`.

Rationale

Knowledge of the conformance of an imported schema allows processors to understand the semantics of referenced components, without additional processing. Namespaces imported into NIEM-conformant schemas are assumed to be conformant unless otherwise indicated.

[Rule 7-62] (REF, EXT)

Within the schema, an element `xsd:import` that imports a namespace defined by an external schema MUST be a documented component.

Rationale

A NIEM-conformant schema has well-known documentation points. Therefore, a schema that imports a NIEM-conformant namespace need not provide additional documentation. However, when an external schema is imported, appropriate documentation must be provided at the point of import because documentation associated with external schemas is undefined and variable. In this particular case, documentation of external schemas is required at their point of use in NIEM.

[Definition: adapter type]

An **adapter type** is a NIEM-conformant type that adapts external components for use within NIEM. An adapter type creates a new class of object that embodies a single concept composed of external components. A NIEM-conformant schema defines an adapter type.

[Rule 7-63] (REF, EXT)

Within the schema, an adapter type MUST have application information `appinfo:ExternalAdapterTypeIndicator` with a value of `true`. A type that is not an adapter type SHALL NOT contain that indicator.

Rationale

This rule flags as external adapters those types that may contain external content. This allows for easier processing.

[Rule 7-64] (REF, SUB, EXT)

Within the schema, an adapter type MUST be an immediate extension of type `structures:ComplexObjectType`.

Rationale

The adapter type must contain the content defined for any NIEM component. The type `structures:ComplexObjectType` provides such content

[Rule 7-65] (REF, SUB, EXT)

Within the schema, an adapter type MUST be composed of only elements and attributes from an external standard.

Rationale

An adapter type should contain the information from an external standard to express a complete concept. This expression should be composed of content entirely from an external schema. Most likely, the external schema will be based on an external standard with its own legacy support.

In the case of an external expression that is in the form of model groups, attribute groups, or types, additional elements and type components may be created in an external schema, and the adapter type may use those components.

[Rule 7-66] (REF, EXT)

Within the schema, an element reference used in an adapter type definition MUST be a documented component.

[Rule 7-67] (REF, EXT)

Within the schema, an attribute reference used in an adapter type definition MUST be a documented component.

Rationale

In normal (conformant) type definition, a reference to an attribute or element is a reference to a documented component. Within an adapter type, the references to the attributes and elements being adapted are references to undocumented components. These components must be documented to provide comprehensibility and interoperability. Since documentation made available by nonconformant schemas is undefined and variable, documentation of these components is required at their point of use, within the conformant schema.

[Rule 7-68] (REF, SUB, EXT)

Within the schema, an adapter type MUST NOT be extended or restricted.

Rationale

Adapter types are meant to stand alone; each type expresses a single concept from an external schema, and adapter types are maintained in separate schemas that only

contain adapter types. In this way, processors may easily switch modes, processing NIEM-conformant content in one way, and external content in another.

7.8 NIEM Subset Schemas

Subset schemas are schemas that are based on other NIEM-conformant schemas but have been modified for any of several reasons. A subset schema may be created that limits what is considered valid data to a subset of what is valid against the base schema. The subset schema may also remove constructs from the schema that do not affect XML Schema validation of instances against the schema, which may include removing documentation, `appinfo` annotations, and comments.

[Rule 7-69] (SUB)

The value of the `targetNamespace` attribute owned by the `xsd:schema` document element of the subset schema must be the same as the value of the `targetNamespace` attribute owned by the `xsd:schema` document element of the reference schema.

[Rule 7-70] (SUB)

The schema must be constructed such that any instance that is XML Schema valid against the schema must also be XML Schema valid against the base schema.

Rationale

A subset schema is a briefer, abridged form of its base schema. The subset schema is intended to stand in the place of the base schema for the purpose of XML Schema validation in many situations. As such, it is imperative that the subset schema sustain the constraints expressed by the base schema. The NDR does not specify what mechanisms a subset schema must use to support the constraints of the base schema.

7.9 Container Elements

All NIEM properties establish a relationship between the object holding the property and the value of the property. For example, an activity object of type `nc:ActivityType` may have an element `nc:ActivityDescriptionText`. This element will be of type `nc:TextType` and represents a NIEM property owned by that activity object. An occurrence of this element within an activity object establishes a relationship between the activity object and the text: the text is the description of the activity.

In a NIEM-conformant instance, an element establishes a relationship between the object that contains it and the element's value. This relationship between the object and the element may be semantically strong, such as the text description of an activity in the previous example, or it may be semantically weak, with its exact meaning left unstated. In NIEM, the contained element involved in a weakly defined semantic relationship is commonly referred to as a **container element**.

A container element establishes a weakly defined relationship with its containing element. For example, an object of type nc:ItemDispositionType may have a container element nc:Item of type nc:ItemType. The container element nc:Item does not establish what relationship exists between the object of nc:ItemDispositionType and itself. There could be any of a number of possible semantics between an object and the value of a container element. It could be a contained object, a subpart, a characteristic, or some other relationship. The appearance of this container element inside the nc:ItemDispositionType merely establishes that the disposition has an item.

The name of the container element is usually based on the NIEM type that defines it: nc:PersonType uses a container element nc:Person, while nc:ActivityType uses a container element nc:Activity. The concept of an element as a container element is a notional one.

There are no formalized rules addressing what makes up a container element. A container element is vaguely defined and carries very little semantics about its context and its contents. Accordingly, there is no formal definition of container elements in NIEM: There are no specific artifacts that define a container element; there are no appinfo or other labels for container elements.

The appearance of a container element within a NIEM type carries no additional semantics about the relationship between the property and the containing type. The use of container elements indicates only that there is a relationship; it 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 with the semantic-free implications associated with the use of the nc:Person container element.

When a NIEM-conformant schema requires a new container element, it may define a new element with a concrete type and a general name, with general semantics. Any schema may define a container element when it requires one. NIEM-conformant schemas may also create reference elements with general semantics. For example, an element nc:PersonReference will carry the same general, container-like meaning as an element nc:Person.

8 XML Instance Rules

This specification attempts to restrict XML instance data as little as possible while still maintaining interoperability. Section 2.6, NIEM-Conformant XML Documents and Elements, defines terminology for NIEM-conformance and XML documents.

The NIEM does not require a specific encoding or specific requirements for the XML prologue, except as specified by **[XML]**.

8.1 Instance Validation

[Rule 8-1] (INS)

> The XML document MUST be schema-valid, assessed with reference to the schema composed of the reference schemas, extension schemas, exchange schemas, utility schemas, and external schemas for the relevant namespaces.

Rationale

> The schemas that define the exchange must be authoritative. Each is the reference schema, extension schema, or exchange schema for the namespace it defines. Application developers may use other schemas for various purposes, but for the purposes of determining conformance, the authoritative schemas are relevant.
>
> This rule should not be construed to mean that XML validation must be performed on all XML instances as they are served or consumed; only that the XML instances validate if XML validation is performed. The XML Schema component definitions specify XML documents and element information items, and the instances should follow the rules given by the schemas, even when validation is not performed.

NIEM embraces the use of XML Schema instance attributes, including `xsi:type`, `xsi:nil`, and `xsi:schemaLocation`, as specified by **[XMLSchemaStructures]**.

8.2 Instance Meaning

[Rule 8-2] (INS)

> Within the instance, the meaning of an element with no content is that additional properties are not asserted. There SHALL NOT be additional meaning interpreted for an element with no content.

Rationale

> Elements without content only show a lack of asserted information. That is, all that is asserted is what is explicitly stated, through a combination of XML instance data and its schema. Data that is not present makes no claims. It may be absent due to lack of availability, lack of knowledge, or deliberate withholding of information. These cases should be modeled explicitly, if they are required.

8.3 Component Representation

NIEM uses element containment for the majority of its data representation needs; that is, an element containing another element. In general, one object (the content of the outer element) has a relationship (defined by the name of the inner element) to another object (the content of the inner element).

Figure 8-1: Example of element containment

```
<OuterElement>
  <!-- object1: the content of outer element -->
  <InnerElement>
     <!-- object2: the content of inner element -->
  </InnerElement>
  <!-- object1, continued -->
</OuterElement>
```

This use of the element containment method has limitations. Specifically, recursive and symmetric relationships (direct or transitive) create difficulties, such as repetition of data and resolution of duplicates.

To avoid these problems, NIEM allows references between elements. In this way, one object (the content of one element) has a relationship (defined by the name of the inner element) to another object (the content of an element referenced by an attribute of the inner element).

Figure 8-2: Example of element reference

```
<OuterElement>
  <!-- object1: the content of outer element -->
  <InnerElementReference structures:ref="object2"/>
  <!-- object1, continued -->
</OuterElement>

<OtherElement structures:id="object2">
   <!-- object2: the content of other element -->
</OtherElement>
```

[Rule 8-3] (INS)

Within an element instance, there SHALL NOT be any difference in meaning between a property asserted via element containment and a property asserted by element reference, except as explicitly described by the semantics of the elements involved.

Rationale

There is no difference in meaning between relationships established by containment and those established by reference. They are simply two mechanisms for expressing connections between objects. Neither mechanism implies that properties are intrinsic or extrinsic. Such characteristics must be explicitly stated in property definitions.

Being of type `xsd:ID` and `xsd:IDREF`, validating schema parsers will perform certain checks on the values of `structures:id` and `structures:ref`. Specifically, no two IDs may

have the same value. This includes `structures:id` and other IDs that may be used in an instance. Also, any value of `structures:ref` must also appear as the value of an ID.

[Rule 8-4] (INS)

Given that the IDREF that is the value of an attribute `structures:ref` matches the value of an ID attribute on some element in the XML document, that ID attribute must be an occurrence of the attribute `structures:id`.

Rationale

This states that in NIEM-conformant content, `structures:ref` attributes must refer to `structures:id` attributes. By **[XML]**, an IDREF is required to reference an ID. This rule ensures that the target of a reference is a NIEM ID for easier processing of XML documents.

Reference element definitions may include constraints on the type of object that may be referenced by that element.

[Rule 8-5] (INS)

Within an element instance, given that a reference element is restricted to a target type T, any attribute `structures:ref` MUST reference an element that has a type definition of type T or that is derived from type T.

Rationale

This rule says that the type of the object pointed to by a `structures:ref` attribute must be of a type specified by the reference element definition. The restriction of types is defined in the application information of the reference element definition by the use of the `appinfo:ReferenceTarget` attribute. The definition of *reference* is as given in **[XMLInfoSet]**, in the description of attribute information items.

8.4 Component Ordering

An instance may express the natural order of components by using the order of content within an XML file. It may also use the `structures:sequenceID` to indicate the order of components.

[Rule 8-6] (INS)

The order of elements that are children of an element SHALL be presented as if their sequential order is as follows:

1. First, elements owning an attribute `structures:sequenceID`, in the order that would be yielded with their sequence IDs sorted via `sort` element as defined by **[XSLT]**, with a data type of `number` and an order of `ascending`.
2. Following those elements, the remaining elements, in the order in which they occur within the XML instance.

Rationale

Because of NIEM's use of structured, defined types and its use of `xsd:sequence`, as well as various representation mechanisms, the order of data within an XML instance may require more precise definition and may vary from instance to instance. The true order of objects (such as parts of a name, lines in an address, or parts of a phone number) may need an explicit method to define their order.

In this definition, the term "presented" may mean presentation to the user, reports, or transfer to other data systems. It is meaningful only when the order of appearance of items within a sequence is expressed. Such an order is only the default for the content within an instance. Any meaningful sorting or other processing may overrule it.

[Rule 8-7] (REF, EXT, INS)

Within a schema or instance, the attribute `structures:sequenceID` SHALL NOT be interpreted as meaningful beyond an indicator of sequential order of an object relative to its siblings.

Rationale

Siblings of a data item are items that have the same parent. Note that, using the reference and relationships mechanisms, data objects may have multiple parents. The sequenceID is truly metadata, helping to express the structure of the data rather than its content.

Note that reference elements have the same semantics as concrete data elements; thus they follow the same rules for sequential order. By using reference elements, an entity may have one order within one structure and another order within another structure.

Within NIEM-conformant instances, the order of objects is found to be given by sorting the objects by numerical value of their respective attribute `structures:sequenceID`, from smallest to highest. The relative order of objects with equal values for `structures:sequenceID` is their order within the XML instance. Objects with no value for `structures:sequenceID` occur after all objects that have values for `structures:sequenceID`, in their relative order within the XML instance.

The use of instance-based sequencing, including the use of `structures:sequenceID`, is preferred over efforts to sequence data definitions. For example, the use of "address line 1," "address line 2," "address line 3," etc., is not recommended. Instead, a single "address line" would be preferred, with order expressed in the XML instance.

8.5 Instance Metadata

NIEM provides the metadata mechanism for giving information about object assertions. An object may have an attribute that refers to one or more metadata objects. A `structures:metadata` attribute indicates that a data item has the given metadata. A `structures:linkMetadata` attribute asserts that the link (or relationship) established by an element has the given metadata.

Figure 8-3: Example of metadata used in an instance

```
<nc:Person>
  <nc:PersonName s:metadata="m1 m2" s:linkMetadata="m3">
    <nc:PersonFullName>John Doe</nc:PersonFullName>
  </nc:PersonName>
  <nc:PersonBirthDate s:metadata="m2">
    <nc:Date>1945-12-01</nc:Date>
  </nc:PersonBirthDate>
</nc:Person>
<nc:Metadata structures:id="m1">
  <nc:SourceText>Adam Barber</nc:SourceText>
</nc:Metadata>
<nc:Metadata structures:id="m2">
  <nc:ReportedDate>
    <nc:Date>2005-04-26</nc:Date>
  </nc:ReportedDate>
</nc:Metadata>
<nc:Metadata structures:id="m3">
  <nc:ProbabilityNumeric>0.25</nc:ProbabilityNumeric>
</nc:Metadata>
```

This example shows a person named John Doe, born 12/1/1945. This data has several pieces of metadata on it:

- Metadata `m1` asserts Adam Barber gave the name.
- Metadata `m2` asserts the name and the birth date were reported on 4/26/2005.
- Link metadata `m3` asserts a 25% probability that the name goes with the person.

This shows several characteristics of metadata:

- Metadata objects may appear outside the data they describe.
- Metadata objects may be reused.
- Data may refer to more than one metadata object.
- Metadata pertains to an object or simple content, while link metadata pertains to the relationship between objects.

An instance would not be valid XML if the `structures:metadata` or `structures:linkMetadata` attributes contained references for which there were no defined IDs. The instance would not be NIEM-conformant if the references were not to IDs defined with the `structures:id` attribute.

The definition of a metadata type may contain an `appinfo:AppliesTo` element, which indicates the type to which the metadata applies. For example:

Figure 8-4: A metadata type that describes applicability using `structures:AppliesTo`

```
<xsd:complexType name="MeasureMetadataType">
  <xsd:annotation>
    ...
    <xsd:appinfo>
      ...
      <i:AppliesTo i:name="MeasureType"/>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexContent>
    ...
  </xsd:complexContent>
</xsd:complexType>
```

Application of metadata to a type to which it is not applicable is not NIEM-conformant. A metadata type may contain multiple `structures:AppliesTo` elements, in which case it may apply to an instance of any of the listed types. If a metadata type contains no `structures:AppliesTo` elements, then it may apply to any type. This is the case for `nc:MetadataType` in NIEM 2.0.

[Rule 8-8] (INS)

Within an element instance, when an object O links to a metadata object via an attribute `structures:metadata`, the information in the metadata object SHALL be applied to the object O.

[Rule 8-9] (INS)

Within an element instance, when an object O1 contains an element E, with content object O2 or with a reference to object O2, and O2 links to a metadata object via an attribute `structures:linkMetadata`, the information in the metadata object SHALL be applied to the relationship E between O1 and O2.

Rationale

These two rules define the meaning of metadata:

- `structures:metadata` applies metadata to an object.
- `structures:linkMetadata` applies metadata to a relationship between two objects.

[Rule 8-10] (INS)

Given that each IDREF in the value of an attribute `structures:metadata` must match the value of an ID attribute on some element in the XML document, that ID attribute MUST be an occurrence of the attribute `structures:id`.

[Rule 8-11] (INS)

Each element that an attribute `structures:metadata` references MUST have a type definition that is derived from `structures:MetadataType`.

[Rule 8-12] (INS)

Given that each IDREF in the value of an attribute `structures:linkMetadata` must match the value of an ID attribute on some element in the XML document, that ID attribute MUST be an occurrence of the attribute `structures:id`.

[Rule 8-13] (INS)

Each element that an attribute `structures:linkMetadata` references MUST have a type definition that is derived from structures:MetadataType.

Rationale

All `structures:metadata` and `structures:linkMetadata` attributes must refer to metadata objects, and the reference to that object must be established using the `structures:id` attribute, to facilitate processing of XML documents.

[Rule 8-14] (INS)

Given that an element information item E has a type definition of some type T, each metadata type that is the type definition of an element information item referenced by an attribute `structures:metadata` or `structures:linkMetadata` on element E MUST be applicable to T.

Rationale

The applicability is determined by `structures:AppliesTo` application information of the metadata type definition. The instances must correspond to the types specified by the metadata type definition.

9 Naming Rules

This section outlines the rules used to create names for NIEM data components previously discussed in this document. Data component names must be understood easily both by humans and by machine processes. These rules improve name consistency by restricting characters, terms, and syntax that could otherwise allow too much variety and potential ambiguity. These rules also improve readability of names for humans, facilitate parsing of individual terms that compose names, and support various automated tasks associated with dictionary and controlled vocabulary maintenance.

9.1 Extension of XSD Namespace Simple Types

[Rule 9-1] (REF, SUB, EXT)

Within the schema, a complex type that is a direct extension of a simple type from the XML Schema namespace simple type MAY use the same local name as the simple type if and only if the extension adds no content other than the attribute group `structures:SimpleObjectAttributeGroup`.

Rationale

It is useful to build complex type bases for further extension. The NIEM distribution proxy schema `xsd.xsd` provides complex type bases for some of the simple types in the XML Schema namespace. However, the complex types in this proxy schema reuse the local names of the simple types they extend, even though the simple type names may not be NIEM-conformant. Requiring name changes for those NIEM-provided complex type bases would work against user understanding, for those already familiar with the names of the XML Schema namespace simple types being extended.

9.2 Usage of English

[Rule 9-2] (REF, SUB, EXT)

The name of any XML Schema component defined by the schema SHALL be composed of words from the English language, using the prevalent U.S. spelling, as provided by **[OED]**.

Rationale

The English language has many spelling variations for the same word. For example, American English “program” has a corresponding British spelling “programme.” This variation has the potential to cause interoperability problems when XML components are exchanged because of the different names used by the same elements. Providing users with a dictionary standard for spelling will mitigate this potential interoperability issue.

9.3 Characters in Names

[Rule 9-3] (REF, SUB, EXT)

The name of any XML Schema component defined by the schema SHALL contain only the following characters:

- Upper-case letters ('A'-'Z').
- Lower-case letters ('a'-'z').
- Digits ('0'-'9').
- Hyphen ('–').

Other characters, such as the underscore ('_') character and the period ('.') character SHALL NOT appear in component names in NIEM-conformant schemas.

[Rule 9-4] (REF, SUB, EXT)

The hyphen character ('–') MAY appear in component names only when used as a separator between parts of a single word, phrase, or value, which would otherwise be incomprehensible without the use of a separator.

Rationale

Names of standards and specifications, in particular, tend to consist of series of discrete numbers. Such names require some explicit separator to keep the values from running together. The separator used within NIEM is the hyphen.

Names of NIEM components follow the rules of XML Schema, by [Rule 5-3]. NIEM components also must follow the rules specified for each type of XML Schema component.

9.4 Character Case

[Rule 9-5] (REF, SUB, EXT)

Within the schema, any attribute declaration SHALL have a name that begins with a lower-case letter ('a'-'z').

[Rule 9-6] (REF, SUB, EXT)

Within the schema, any XML Schema component other than an attribute declaration SHALL have a name that begins with an upper-case letter ('A'-'Z').

Camel case is the practice of writing compound words or phrases in which the words are joined without spaces and are capitalized within the compound words. **[Wikipedia]**

[Rule 9-7] (REF, SUB, EXT)

The name of any XML Schema component defined by the schema SHALL use the camel case formatting convention.

Rationale

The foregoing rules establish *lowerCamelCase* for all NIEM components that are XML attributes and *UpperCamelCase* for all NIEM components that are types, elements, or groups.

9.5 Use of Acronyms and Abbreviations

Acronyms and abbreviations have the ability to improve readability and comprehensibility of large, complex, or frequently used terms. They also obscure meaning and impair understanding when their definitions are not clear or when they are used injudiciously. They should be used with great care. Acronyms and abbreviations that are used must be documented and used consistently.

[Rule 9-8] (REF, SUB, EXT)

The schema MUST consistently use approved acronyms, abbreviations, and word truncations within defined names. The approved shortened forms are defined in Table 9-1: Abbreviations Used in NIEM Core Names .

Table 9-1: Abbreviations Used in NIEM Core Names

Abbreviation	Full Meaning
ANSI	American National Standards Institute
CMV	Commercial Motor Vehicle
DEA	Drug Enforcement Agency
DNA	Deoxyribonucleic Acid
FGI	Foreign Government Information
FIPS	Federal Information Processing Standard
IC	Intelligence Community
ID	Identifier
IP	Internet Protocol
ISO	International Standards Organization
LIS	NCIC code list for license state
LSTA	NCIC code list for state/country index
MCO	Manufacturer's Certificate of Origin
MGRS	Military Grid Reference System
MSRP	Manufacturer's Suggested Retail Price
NANP	North American Numbering Plan
NCIC	National Crime Information Center
NCTC	National Counter Terrorist Center
NIBRS	National Incident Based Reporting System

NLETS	The International Justice & Public Safety Information Sharing Network (formerly known as the National Law Enforcement Teletype System)
ORI	Organization Identifier (Orion)
RES	NCIC code list for registration state for boat registrations
RF	Radio Frequency
SIM	Subscriber Identity Module
SSN	Social security number
TYP	NCIC code list for gun type
TYPO	NCIC code list for ORI type
URI	Uniform Resource Identifier
US	United States
UTM	Universal Transverse Mercator
VIN	Vehicle Identification Number
VINA	Vehicle Identification Number Analysis

Rationale

Consistent, controlled, and documented abridged terms that are used frequently and/or tend to be lengthy can support readability, clarity, and reduction of name length.

9.6 Word Forms

[Rule 9-9] (REF, SUB, EXT)

A noun used as a term in the name of an XML Schema component MUST be in singular form unless the concept itself is plural.

[Rule 9-10] (REF, SUB, EXT)

A verb used as a term in the name of an XML Schema component MUST be used in the present tense unless the concept itself is past tense.

[Rule 9-11] (REF, SUB, EXT)

Articles, conjunctions, and prepositions SHALL NOT be used in NIEM component names except where they are required for clarity or by standard convention.

Rationale

Articles (e.g., a, an, the), conjunctions (e.g., and, or, but), and prepositions (e.g., at, by, for, from, in, of, to) are all disallowed in NIEM component names, unless they are required. For example, `PowerOfAttorneyCode` requires the preposition. These rules constrain slight variations in word forms and types to improve consistency and reduce potentially ambiguous or confusing component names.

9.7 Name Generation

Elements in NIEM-conformant schemas are given names that follow a specific pattern. This pattern comes from **[ISO 11179 Part 5]**.

[Rule 9-12] (REF, SUB, EXT)

Except as specified elsewhere in this document, any element or attribute defined within the schema SHALL have a name that takes the form:

- Object-class qualifier terms (0 or more).
- An object class term (1).
- Property qualifier terms (0 or more).
- A property term (1).
- Representation qualifier terms (0 or more).
- A representation term (1).

Rationale

Consistent naming rules are helpful for users who wish to understand components with which they are unfamiliar, as well as for users to find components with known semantics. This rule establishes the basic structure for an element or attribute name, in line with the rules for names under **[ISO 11179 Part 5]**. Note that many elements with complex type should not have a representation term.

9.8 Object-Class Term

The NIEM adopts an object-oriented approach to representation of data. Object classes represent what **[ISO 11179 Part 5]** refers to as “things of interest in a universe of discourse that may be found in a model of that universe.” An object class or object term is a word that

represents a class of real-world entities or concepts. An object-class term describes the applicable context for a NIEM component.

[Rule 9-13] (REF, SUB, EXT)

The object-class term of a NIEM component SHALL consist of a term identifying a category of concrete concepts or entities.

Rationale

The object-class term indicates the object category that this data component describes or represents. This term provides valuable context and narrows the scope of the component to an actual class of things or concepts.

Example

Concept term: Activity

Entity term: Vehicle

9.9 Property Term

Objects or concepts are usually described in terms of their characteristic properties, data attributes, or constituent subparts. Most objects can be described by several characteristics. Therefore, a property term in the name of a data component represents a characteristic or subpart of an object class and generally describes the essence of that data component.

[Rule 9-14] (REF, SUB, EXT)

A property term SHALL describe or represent a characteristic or subpart of an entity or concept.

Rationale

The property term describes the central meaning of the data component.

9.10 Qualifier Terms

Qualifier terms modify object, property, representation, or other qualifier terms to increase semantic precision and reduce ambiguity. Qualifier terms may precede or succeed the terms they modify. The goal for the placement of qualifier terms is to generally follow the rules of ordinary English while maintaining clarity.

[Rule 9-15] (REF, SUB, EXT)

Multiple qualifier terms MAY be used within a component name as necessary to ensure clarity and uniqueness within its namespace and usage context.

[Rule 9-16] (REF, SUB, EXT)

The number of qualifier terms SHOULD be limited to the absolute minimum required to make the component name unique and understandable.

[Rule 9-17] (REF, SUB, EXT)

The order of qualifiers SHALL NOT be used to differentiate names.

Rationale

Very large vocabularies may have many similar and closely related properties and concepts. The use of object, property, and representation terms alone is often not sufficient to construct meaningful names that can uniquely distinguish such components. Qualifier terms provide additional context to resolve these subtleties. However, swapping the order of qualifiers rarely (if ever) changes meaning; qualifier ordering is no substitute for meaningful terms.

9.11 Representation Term

The representation term for a component name serves several purposes in NIEM:

1. It can indicate the style of component. For example, types are clearly labeled with the representation term `Type`.
2. It helps prevent name conflicts and confusion. For example, elements and types may not be given the same name.
3. It indicates the nature of the value carried by element. Labeling elements and attributes with a notional indicator of the content eases discovery and comprehension.

[Rule 9-18] (REF, EXT)

If any word in the representation term is redundant with any word in the property term, one occurrence SHOULD be deleted.

Rationale

This rule, carried over from 11179, is designed to prevent repeating terms unnecessarily within component names. For example, this rule allows designers to avoid naming an element "PersonFirstNameName."

The valid value set of a data element or value domain is described by the representation term. NIEM uses a standard set of representation terms in the representation portion of a NIEM-conformant component name. Table 9-2: Representation Terms lists the primary representation terms and a definition for the concept associated with the use of that term. The table also lists secondary representation terms that may represent more specific uses of the concept associated with the primary representation term.

Table 9-2: Representation Terms

Primary Representation Term	Secondary Representation Term	Definition
Amount	-	A number of monetary units specified in a currency where the unit of currency is explicit or implied.
BinaryObject	-	A set of finite-length sequences of binary octets.
	Graphic	A diagram, graph, mathematical curves, or similar representation
	Picture	A visual representation of a person, object, or scene
	Sound	A representation for audio
	Video	A motion picture representation; may include audio encoded within
Code		A character string (i.e., letters, figures, and symbols) that for brevity, language independence, or precision represents a definitive value of an attribute.
DateTime		A particular point in the progression of time together with relevant supplementary information.
	Date	A particular day, month, and year in the Gregorian calendar.

	Time	A particular point in the progression of time within an unspecified 24-hour day.
ID		A character string to identify and distinguish uniquely one instance of an object in an identification scheme from all other objects in the same scheme together with relevant supplementary information.
	URI	A string of characters used to identify (or name) a resource. The main purpose of this identifier is to enable interaction with representations of the resource over a network, typically the World Wide Web, using specific protocols. A URI is either a Uniform Resource Locator (URL) or a Uniform Resource Name (URN). The specific syntax for each is defined by **[RFC3986]**.
Indicator		A list of two mutually exclusive Boolean values that express the only possible states of a property.
Measure		A numeric value determined by measuring an object along with the specified unit of measure.

Numeric		Numeric information that is assigned or is determined by calculation, counting, or sequencing. It does not require a unit of quantity or unit of measure.
	Value	A result of a calculation.
	Rate	A representation of a ratio where the two units are not included.
	Percent	A representation of a ratio in which the two units are the same.
Quantity		A counted number of nonmonetary units possibly including fractions.
Text	-	A character string (i.e., a finite sequence of characters) generally in the form of words of a language.
	Name	A word or phrase that constitutes the distinctive designation of a person, place, thing, or concept.

[Rule 9-19] (REF, SUB, EXT)

Within the schema, the name of an element declaration that is of simple content MUST use a representation term found in Table 9-2: Representation Terms.

[Rule 9-20] (REF, SUB, EXT)

Within the schema, the name of an element declaration that is of complex content, and that corresponds to a concept listed in Table 9-2: Representation Terms, MUST use a representation term from that table.

[Rule 9-21] (REF, SUB, EXT)

Within the schema, the name of an element declaration that is of complex content and that does not correspond to a concept listed in Table 9-2: Representation Terms MUST NOT use a representation term.

[Rule 9-22] (REF, SUB, EXT)

Within the schema, the name of an attribute declaration MUST use a representation term from Table 9-2: Representation Terms.

Rationale

An element that represents a value listed in the table should have a representation term. It should do so even if its type is complex with multiple parts. For example, a type with multiple fields may represent a sound binary, a date, or a name.

9.12 NIEM Type Names

This section contains naming rules specific to various kinds of NIEM types.

9.12.1 All Type Components

[Rule 9-23] (REF, SUB, EXT)

Within the schema, the name of any type definition MUST use the representation term `Type`.

Rationale

Using the representation term `Type` immediately identifies XML types in a NIEM-conformant schema and prevents naming collisions with corresponding XML elements and attributes.

9.12.2 Simple Type Components

[Rule 9-24] (REF, SUB, EXT)

Within the schema, the name of any simple type definition SHALL use the representation term qualifier `Simple`. This qualifier SHALL appear after any other representation term qualifiers.

Rationale

Specific uses of type definitions have similar syntax but very different effects on data definitions. Schemas that clearly identify complex and simple type definitions are easier to understand without tool support. This rule ensures that names of simple types end in `SimpleType`.

9.12.3 Code Type Components

[Definition: code type]

A **code type** is a simple type schema component definition that contains multiple `xsd:enumeration` facets.

These types represent lists of values, each of which has a known meaning beyond the text representation. These values may be meaningful text or may be a string of alphanumeric identifiers that represent abbreviations for literals.

[Rule 9-25] (REF, SUB, EXT)

Within the schema, the name of any code type SHALL use the representation term qualifier `Code`.

Rationale

Using the qualifier `Code` (e.g. `CodeType, CodeSimpleType`) immediately identifies a type as representing a fixed list of codes. These types may be handled in specific ways, as lists of codes are expected to have their own lifecycles, including versions and periodic updates. Codes may also have responsible authorities behind them who provide concrete semantic bindings for the code values.

[Rule 9-26] (REF, SUB, EXT)

Within the schema, any type definition which has a base type definition of a code type or which is transitively based on a code type SHALL have a name that uses the representation term qualifier `Code`.

Rationale

This expands the use of the representation term qualifier `Code` to any type based on a code list.

9.12.4 Association Type Components

[Rule 9-27] (REF, SUB, EXT)

Within the schema, any association type SHALL have a name that uses the representation term qualifier `Association`. Types other than association types SHALL NOT use the representation term qualifier `Association`.

Rationale

Using the qualifier `Association` immediately identifies a type as representing an association.

9.12.5 Augmentation Type Components

[Rule 9-28] (REF, SUB, EXT)

Within the schema, any augmentation type SHALL have a name that uses the representation term qualifier `Augmentation`. Types other than augmentation types SHALL NOT use the representation term qualifier `Augmentation`.

Rationale

Using the qualifier `Augmentation` immediately identifies a type as representing an augmentation.

9.12.6 Metadata Type Components

[Rule 9-29] (REF, SUB, EXT)

Within the schema, any metadata type SHALL have a name that uses the representation term qualifier `Metadata`. Types other than metadata types SHALL NOT use the representation term qualifier `Metadata`.

Rationale

Using the qualifier `Metadata` immediately identifies a type as representing metadata.

9.13 NIEM Property Names

This section contains naming rules specific to different kinds of NIEM properties.

9.13.1 Attribute Group Names

[Rule 9-30] (REF, SUB, EXT)

Within the schema, the name of any attribute group definition schema component SHALL use the representation term `AttributeGroup`.

Rationale

This clearly identifies attribute groups and partitions their names from the names of other types of schema components.

9.13.2 Reference Names

[Rule 9-31] (REF, SUB, EXT)

Within the schema, the name of any reference element SHALL use the representation term suffix `Reference`.

Rationale

Reference elements are identical in semantics to elements that are not by reference. However, they refer to their values by a reference attribute instead of carrying it as content of the XML element. The use of a suffix helps indicate that the elements refer

to, instead of contain, their values, yet allows the basic semantics (e.g., property, representation term) to persist.

Note that the use of the representation term suffix is one of the situations in which there is a slight divergence from the general rule for name generation as discussed in [Rule 9-12].

9.13.3 Association Names

[Rule 9-32] (REF, SUB, EXT)

Within the schema, the name of an association element SHALL use the representation term qualifier `Association`.

Rationale

Using the qualifier `Association` immediately identifies an element as representing an association.

9.13.4 Augmentation Names

[Rule 9-33] (REF, SUB, EXT)

Within the schema, the name of an augmentation element SHALL use the representation term `Augmentation`.

Rationale

Using the qualifier `Augmentation` immediately identifies an element as representing an augmentation.

9.13.5 Metadata Names

[Rule 9-34] (REF, SUB, EXT)

Within the schema, the name of a metadata element SHALL use the representation term `Metadata`.

Rationale

Using the qualifier `Metadata` immediately identifies an element as representing metadata.

9.13.6 Role Names

[Rule 9-35] (REF, SUB, EXT)

Within the schema, the name of a role SHALL use the property term `RoleOf`.

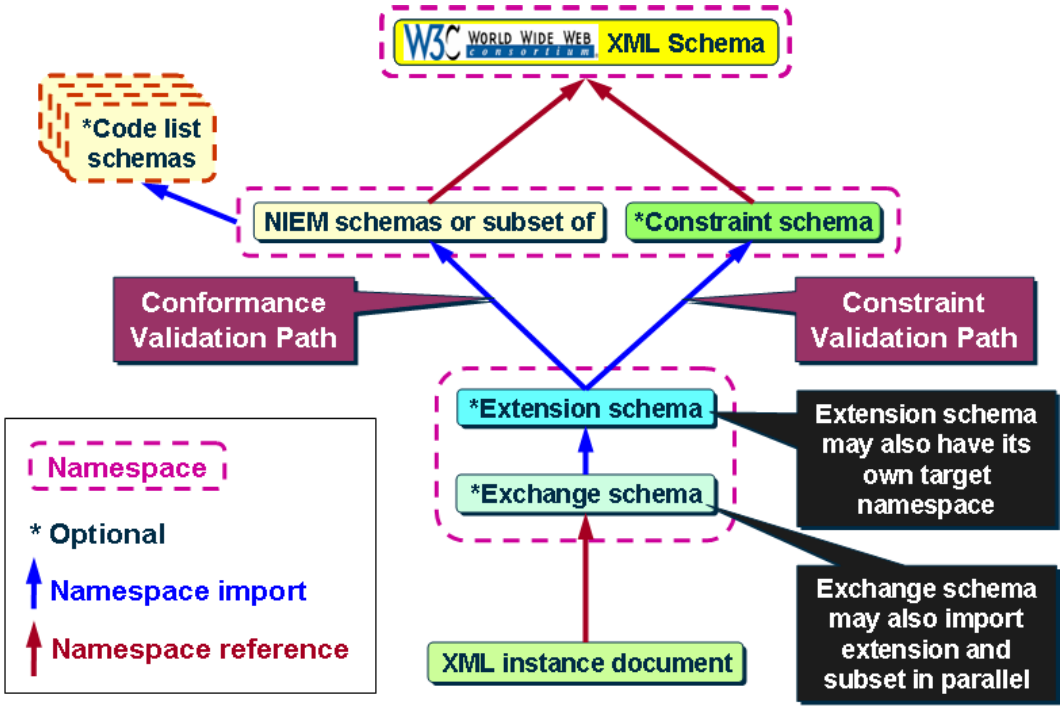
Rationale

Using the property term `RoleOf` immediately identifies an element as representing a role.

Appendix A: NIEM Overview

NIEM is a reference model of unconstrained components rendered in XML Schema. Associated with the NIEM-conformant 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 Schema documents for NIEM Information Exchange Package Documentation (IEPD).

Figure A-1: The NIEM XML Reference Architecture



An Exchange Package is defined by the Federal Enterprise Architecture (FEA) Data Reference Model **[DRM]** as a description of a specific recurring data exchange between a supplier and a consumer. A NIEM IEPD is a set of artifacts that implements an FEA DRM Exchange Package. The NIEM IEPD Specification **[IEPD]** contains a more detailed explanation of IEPDs and their contents.

The following kinds of 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. The structure and content of such distributions are not specified in this document. This document specifies rules that apply to the NIEM-conformant schemas that are released as part of such 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 that adds domain- or application-specific content to the base NIEM model.
- Exchange Schema: a NIEM-conformant schema that 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 require only 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. NIEM schemas may import additional schemas, such as code table schemas, as needed. The optional exchange schema imports, reuses, 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 NIEM but which are needed for the exchange.

Note that 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 overinclusive and underconstrained. The reason for this approach is that predetermining 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-conformant 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 reference schema or subset *conformance validation* path. This is done through multipass validation. A correctly constructed XML instance will validate through both the conformance and the constraint path.

Appendix B: Name Syntax for Special Components

The following table summarizes NIEM general naming syntax for special components and their associated types. Refer to Sections 9.12 and 9.13 for the specific rules associated with this table.

Note that this table does not mention the general syntax for standard types and properties introduced in Sections 9.12 and 9.13.

Table B-1: Name Syntax for Special Components

Name Syntax *	Notes
Association	
[Property]Association	Preferred: [Property] describes relationship
[Object1][Object2]Association	Alternate 1: related objects
[Object]Association	Alternate 2: related objects are same class
Role Reference	
RoleOf[Object]Reference	Element in the role that references base type
Type Augmentation	
[Object][Property]Augmentation	[Object][Property] is from type augmented
Metadata	
[Property]Metadata	
Adapter	
[Object][Property]Adapter	
Abstract	
[Object][Property]	Preferred

Name Syntax *	Notes
[Object][Property]Abstract	Alternate: when required to prevent name clash

***** Object and Property refer to **[ISO 11179 Part 5]** terms in a component name.

Appendix C: Supporting Schemas

NIEM provides a set of schemas that underlie the data model schemas. These schemas do not define data model content; they do not define people, vehicles, or relationships between them. Instead, these schemas define the fundamental framework on which the data model is built.

There are two supporting schemas. The first, called `appinfo`, is the namespace for application information that supports data model definitions. The second is called `structures` and is the namespace for basic types that augment the mechanisms of XML Schema for more sophisticated data modeling and information exchanges.

This appendix defines and discusses each of the framework components in the two supporting schemas. At the conclusion of the discussion of each schema, the full schema is provided as a reference.

This appendix also includes a directory listing of all the reference schemas that are part of NIEM 2.0.

The `appinfo` namespace

The `appinfo` schema provides support for high-level data model concepts and additional syntax to support the NIEM conceptual model and validation of NIEM-conformant instances. This schema is available at **[NIEMAppinfoXSD]**.

Figure C-1: Schema document element

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:i="http://niem.gov/niem/appinfo/2.0"
  xmlns:s="http://niem.gov/niem/structures/2.0"
  targetNamespace="http://niem.gov/niem/appinfo/2.0"
  attributeFormDefault="qualified" version="1">
```

Discussion

The namespace for the `appinfo` namespace is `http://niem.gov/niem/appinfo/2.0`.

Figure C-2: Element `appinfo:Resource`

```
<xsd:element name="Resource">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:NCName" use="required"/>
  </xsd:complexType>
</xsd:element>
```

Discussion

The `Resource` element provides a method for application information to define a name within a schema, without the name being bound to a schema component. This is

used by the `structures` schema to define names for `structures:Object` and `structures:Association`.

Figure C-3: Element `appinfo:Deprecated`

```
<xsd:element name="Deprecated">
  <xsd:complexType>
    <xsd:attribute name="value" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:boolean">
          <xsd:pattern value="true"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    </xsd:complexType>
</xsd:element>
```

Discussion

The `Deprecated` element provides a method for identifying components as being deprecated. A deprecated component is one which is provided but whose use is not recommended.

Figure C-4: Element `appinfo:Base`

```
<xsd:element name="Base">
  <xsd:complexType>
      <xsd:attribute name="name" type="xsd:NCName" use="required"/>
      <xsd:attribute name="namespace" type="xsd:anyURI" use="optional"/>
  </xsd:complexType>
</xsd:element>
```

Discussion

The `Base` element provides a mechanism for indicating base types and base elements in schema for the cases in which XML Schema mechanisms are insufficient. For example, it is used to indicate `Object` or `Association` bases.

Figure C-5: Element `appinfo:ReferenceTarget`

```
<xsd:element name="ReferenceTarget">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:NCName" use="required"/>
    <xsd:attribute name="namespace" type="xsd:anyURI" use="optional"/>
  </xsd:complexType>
</xsd:element>
```

Discussion

The `ReferenceTarget` element indicates a NIEM type which may be a target (that is, a destination) of a NIEM reference element. It may be used in combinations to indicate a set of valid types.

Figure C-6: Element `appinfo:AppliesTo`

```
<xsd:element name="AppliesTo">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:NCName" use="required"/>
    <xsd:attribute name="namespace" type="xsd:anyURI" use="optional"/>
  </xsd:complexType>
</xsd:element>
```

Discussion

The `AppliesTo` element is used in two ways. First, it indicates the set of types to which a metadata type may be applied. Second, it indicates the set of types to which an augmentation element may be applied.

Figure C-7: Element `appinfo:ConformantIndicator`

```
<xsd:element name="ConformantIndicator" type="boolean"/>
```

Discussion

The `ConformantIndicator` element may be used in two ways. First, it is included as application information for a schema document element to indicate that the schema is NIEM-conformant. Second, it is used as application information of a namespace import to indicate that the schema is not NIEM-conformant.

Figure C-8: Element `appinfo:ExternalAdapterTypeIndicator`

```
<xsd:element name="ExternalAdapterTypeIndicator" type="boolean"/>
```

Discussion

The `ExternalAdapterTypeIndicator` element indicates that a complex type is an external adapter type. Such a type is one composed of elements and attributes from non-NIEM-conformant schemas. The indicator allows schema processors to switch to alternative processing modes when processing NIEM-conformant versus non-NIEM-conformant content.

Figure C-9: Full XML Schema for Appinfo Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:i="http://niem.gov/niem/appinfo/2.0" xmlns:s="http://niem.gov/niem/structures/2.0"
targetNamespace="http://niem.gov/niem/appinfo/2.0" attributeFormDefault="qualified"
version="1">

  <xsd:element name="Resource">
    <xsd:complexType>
      <xsd:attribute name="name" type="xsd:NCName" use="required"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="Deprecated">
    <xsd:complexType>
      <xsd:attribute name="value" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:boolean">
            <xsd:pattern value="true"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      </xsd:complexType>
  </xsd:element>

  <xsd:element name="Base">
    <xsd:complexType>
        <xsd:attribute name="name" type="xsd:NCName" use="required"/>
        <xsd:attribute name="namespace" type="xsd:anyURI" use="optional"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="ReferenceTarget">
    <xsd:complexType>
      <xsd:attribute name="name" type="xsd:NCName" use="required"/>
      <xsd:attribute name="namespace" type="xsd:anyURI" use="optional"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="AppliesTo">
    <xsd:complexType>
      <xsd:attribute name="name" type="xsd:NCName" use="required"/>
      <xsd:attribute name="namespace" type="xsd:anyURI" use="optional"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="ConformantIndicator" type="xsd:boolean"/>
  <xsd:element name="ExternalAdapterTypeIndicator" type="xsd:boolean"/>

</xsd:schema>
```

The `structures` schema

The `structures` schema provides support for fundamental NIEM linking mechanisms, as well as providing base types for definition of NIEM-conformant types. This schema is available at **[NIEMStructuresXSD]**.

Figure C-10: Schema document element

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    targetNamespace="http://niem.gov/niem/structures/2.0"
    version="1"
    xmlns:appinfo="http://niem.gov/niem/appinfo/2.0"
    xmlns:s="http://niem.gov/niem/structures/2.0"
    xmlns="http://www.w3.org/2001/XMLSchema">
```

Discussion

The target namespace for the `structures` schema is `http://niem.gov/niem/structures/2.0`.

Figure C-11: Import of `appinfo`

```
<xsd:import
  schemaLocation="../../appinfo/2.0/appinfo.xsd"
  namespace="http://niem.gov/niem/appinfo/2.0"/>
```

Discussion

The `structures` schema uses components from the `appinfo` namespace.

Figure C-12: Resource `structures:Object`

```
<xsd:annotation>
  <xsd:appinfo>
    <i:Resource i:name="Object"/>
  </xsd:appinfo>
</xsd:annotation>
```

Discussion

The `Object` resource defines an identifier that acts as a conceptual base for objects in NIEM-conformant schemas.

Figure C-13: Resource `structures:Association`

```
<xsd:annotation>
  <xsd:appinfo>
    <i:Resource i:name="Association"/>
  </xsd:appinfo>
</xsd:annotation>
```

Discussion

The `Association` resource defines an identifier that acts as a conceptual base for association in NIEM-conformant schemas.

Figure C-14: Attribute `structures:id`

```
<xsd:attribute name="id" type="ID"/>
```

Discussion

The `id` attribute is used to define XML IDs for NIEM objects. These IDs may be targets of reference elements, metadata attributes, and link metadata attributes.

Figure C-15: Attribute `structures:linkMetadata`

```
<xsd:attribute name="linkMetadata" type="IDREFS"/>
```

Discussion

The `linkMetadata` attribute allows an element to point to metadata that affects the relationship between the context and the value of the object.

Figure C-16: Attribute `structures:metadata`

```
<xsd:attribute name="metadata" type="IDREFS"/>
```

Discussion

The attribute `metadata` allows an object to point to metadata that affects itself.

Figure C-17: Attribute `structures:ref`

```
<xsd:attribute name="ref" type="IDREF"/>
```

Discussion

The `ref` attribute is used by reference elements in NIEM to refer to an object via an ID reference, rather than including the object itself as element content.

Figure C-18: Attribute `structures:sequenceID`

```
<xsd:attribute name="sequenceID" type="integer"/>
```

Discussion

The `sequenceID` attribute allows a series of elements to define a sequence for content that does not correspond to the order of element declarations within a type. This attribute may override the sequence of elements appearing within an instance.

Figure C-19: Attribute group `structures:SimpleObjectAttributeGroup`

```
<xsd:attributeGroup name="SimpleObjectAttributeGroup">
    <xsd:attribute ref="s:id"/>
    <xsd:attribute ref="s:metadata"/>
    <xsd:attribute ref="s:linkMetadata"/>
</xsd:attributeGroup>
```

Discussion

The `SimpleObjectAttributeGroup` attribute group provides a collection of attributes that are appropriate for definition of object types.

Figure C-20: Element `structures:Augmentation`

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

Discussion

The `Augmentation` element provides a substitution group head for augmentations. The designer of a message or object may use this element within an object definition. This will allow the selection of augmentations dynamically, at run time (or at least schema selection time) rather than at schema authoring time.

Figure C-21: Element `structures:Metadata`

```
<xsd:element name="Metadata" type="s:MetadataType" abstract="true"/>
```

Discussion

The `Metadata` element provides a substitution group head for metadata. Like the substitution group head for augmentations, this allows selection of metadata to be decided late in message creation, rather than at schema authoring time. This element may also be used to provide a single point in a container where all metadata for a message may be deposited.

Figure C-22: Complex type `structures:AugmentationType`

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

Discussion

The `AugmentationType` type is a base type for all augmentations. An augmentation may have metadata and an ID but may not have link metadata, as it does not establish a relationship between its value and its context. The individual element contents of an augmentation, however, do establish a relationship between the context of the augmentation and the values of the individual elements.

Figure C-23: Type `structures:ComplexObjectType`

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

Discussion

The `ComplexObjectType` type provides a base class for object definition, association definitions, and external adapter type definitions. An instance of one of these types may have an ID. It may have metadata as it establishes the existence of an object (maybe a conceptual object). It may also have link metadata, as an element of one of these types establishes a relationship between its value and its context.

Figure C-24: Type `structures:MetadataType`

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

Discussion

The `MetadataType` type is a base class for metadata type definition. This type provides only an ID, as the metadata may be referenced. It does not itself have metadata and does not have link metadata.

Figure C-25: Type `structures:ReferenceType`

```
<xsd:complexType name="ReferenceType" final="#all">
  <xsd:attribute ref="s:id"/>
  <xsd:attribute ref="s:ref"/>
  <xsd:attribute ref="s:linkMetadata"/>
</xsd:complexType>
```

Discussion

The `ReferenceType` type is the type of all reference elements within NIEM-conformant schemas. This type provides a reference attribute to reference an object defined elsewhere. It includes an ID, as the link established by a reference element may need to be identified, and link metadata, as an element of this type establishes a relationship between its context and the referenced object. It does not contain metadata, as it does not itself establish the existence of an object; it relies on a definition located elsewhere.

Figure C-26: Full XML Schema for Structures Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:i="http://niem.gov/niem/appinfo/2.0"
  xmlns:s="http://niem.gov/niem/structures/2.0"
  targetNamespace="http://niem.gov/niem/structures/2.0"
  version="1">

  <xsd:import
    schemaLocation="../../appinfo/2.0/appinfo.xsd"
    namespace="http://niem.gov/niem/appinfo/2.0"/>

  <xsd:annotation>
    <xsd:appinfo>
      <i:Resource i:name="Object"/>
    </xsd:appinfo>
  </xsd:annotation>

  <xsd:annotation>
    <xsd:appinfo>
      <i:Resource i:name="Association"/>
    </xsd:appinfo>
  </xsd:annotation>

  <xsd:attribute name="id" type="xsd:ID"/>
  <xsd:attribute name="linkMetadata" type="xsd:IDREFS"/>
  <xsd:attribute name="metadata" type="xsd:IDREFS"/>
  <xsd:attribute name="ref" type="xsd:IDREF"/>
  <xsd:attribute name="sequenceID" type="xsd:integer"/>

  <xsd:attributeGroup name="SimpleObjectAttributeGroup">
    <xsd:attribute ref="s:id"/>
    <xsd:attribute ref="s:metadata"/>
    <xsd:attribute ref="s:linkMetadata"/>
  </xsd:attributeGroup>

  <xsd:element name="Augmentation" type="s:AugmentationType"
    abstract="true"/>
  <xsd:element name="Metadata" type="s:MetadataType" abstract="true"/>

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

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

  <xsd:complexType name="MetadataType" abstract="true">
    <xsd:attribute ref="s:id"/>
  </xsd:complexType>

  <xsd:complexType name="ReferenceType" final="#all">
    <xsd:attribute ref="s:id"/>
    <xsd:attribute ref="s:ref"/>
    <xsd:attribute ref="s:linkMetadata"/>
  </xsd:complexType>

</xsd:schema>
```

NIEM 2.0 Reference Schemas – Directory Listing

```
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
        dhs-gmo/AS/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
        iso-10303-step/2/dhs-gmo/1.0.0/ex.xsd
        iso-19139-gmd/
            draft-0.1/gco/dhs-gmo/1.0.0/basicTypes.xsd
            draft-0.1/gco/dhs-gmo/1.0.0/gco.xsd
            draft-0.1/gco/dhs-gmo/1.0.0/gcoBase.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/applicationSchema.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/citation.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/constraints.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/content.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/dataQuality.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/distribution.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/extent.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/freeText.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/gmd.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/identification.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/maintenance.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/metadataApplication.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/metadataEntity.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/metadataExtension.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/portrayalCatalogue.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/referenceSystem.xsd
            draft-0.1/gmd/dhs-gmo/1.0.0/spatialRepresentation.xsd
            draft-0.1/gmx/dhs-gmo/1.0.0/catalogues.xsd
            draft-0.1/gmx/dhs-gmo/1.0.0/codelistItem.xsd
            draft-0.1/gmx/dhs-gmo/1.0.0/crsItem.xsd
            draft-0.1/gmx/dhs-gmo/1.0.0/extendedTypes.xsd
            draft-0.1/gmx/dhs-gmo/1.0.0/gmx.xsd
```

```
            draft-0.1/gmx/dhs-gmo/1.0.0/gmxUsage.xsd
            draft-0.1/gmx/dhs-gmo/1.0.0/uomItem.xsd
            draft-0.1/gsr/dhs-gmo/1.0.0/gsr.xsd
            draft-0.1/gsr/dhs-gmo/1.0.0/spatialReferencing.xsd
            draft-0.1/gss/dhs-gmo/1.0.0/geometry.xsd
            draft-0.1/gss/dhs-gmo/1.0.0/gss.xsd
            draft-0.1/gts/dhs-gmo/1.0.0/gts.xsd
            draft-0.1/gts/dhs-gmo/1.0.0/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
                om/dhs-gmo/1.0.0/event.xsd
                om/dhs-gmo/1.0.0/observation.xsd
                om/dhs-gmo/1.0.0/observationSpecializations.xsd
                om/dhs-gmo/1.0.0/om.xsd
                om/dhs-gmo/1.0.0/procedure.xsd
                om/dhs-gmo/1.0.0/procedureSpecializations.xsd
                st/dhs-gmo/1.0.0/simpleTypeDerivation.xsd
                swe/dhs-gmo/1.0.0/discreteCoverage.xsd
                swe/dhs-gmo/1.0.0/phenomenon.xsd
                swe/dhs-gmo/1.0.0/record.xsd
                swe/dhs-gmo/1.0.0/recordType.xsd
                swe/dhs-gmo/1.0.0/swe.xsd
                swe/dhs-gmo/1.0.0/SWE_basicTypes.xsd
                swe/dhs-gmo/1.0.0/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
        ogc-swe-common/1.0.0/dhs-gmo/1.0.0/parameters.xsd
        ogc-swe-common/1.0.0/dhs-gmo/1.0.0/positionData.xsd
        ogc-swe-common/1.0.0/dhs-gmo/1.0.0/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 D: References

[ARCH]: The NIEM Reference Architecture. Not yet available.

[DRM]: The Federal Enterprise Architecture Data Reference Model, Version 2.0, November 17 2005. Available from http://www.whitehouse.gov/omb/egov/documents/DRM_2_0_Final.pdf.

[IEPD]: Requirements for a National Information Exchange Model (NIEM) Information Exchange Package Documentation (IEPD) Specification, Version 2.1, June 2006. Available from http://www.niem.gov/files/NIEM_IEPD_Requirements_v2_1.txt.

[ISO 11179 Part 4]: ISO/IEC 11179-4:2004, Information technology — Metadata registries (MDR) — Part 4: Formulation of data definitions. Available from http://standards.iso.org/ittf/PubliclyAvailableStandards/c035346_ISO_IEC_11179-4_2004(E).zip.

[ISO 11179 Part 5]: ISO/IEC 11179-5:2005, Information technology — Metadata registries (MDR) — Part 5: Naming and identification principles. Available from http://standards.iso.org/ittf/PubliclyAvailableStandards/c035347_ISO_IEC_11179-5_2005(E).zip.

[N-ary]: Defining N-ary Relations on the Semantic Web, W3C Working Group Note 12 April 2006. Available from http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/.

Use case 3 is described at #useCase3.

[NIEMAppinfoXSD]: The appinfo schema from the NIEM 2.0 distribution. Available from http://niem.gov/niem/structures/2.0/structures.xsd.

[NIEMStructuresXSD]: The structures schema from the NIEM 2.0 distribution. Available from http://niem.gov/niem/structures/2.0/structures.xsd.

[OED]: Oxford English Dictionary, Second Edition, 1989. Available from http://dictionary.oed.com/.

[OJP]: OJP Information Technology Website.Available from http://www.it.ojp.gov/jxdm.

[RDFPrimer]: RDF Primer, W3C Recommendation 10 February 2004. Available from http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

Basic concepts are covered at #basicconcepts.

[RDFConcepts]: Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation 10 February 2004. Available from http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

RDF data model is described at #section-data-model.

[RDFSemantics]: RDF Semantics, W3C Recommendation 10 February 2004. Available from http://www.w3.org/TR/rdf-mt/.

The meaning of RDF assertions is described at #interp.

[RFC2119]: Bradner, S. Key words for use in RFCs to Indicate Requirement Levels, IETF RFC 2119, March 1997. Available from http://www.ietf.org/rfc/rfc2119.txt.

[RFC3986]: Berners-Lee, T., et al.: Uniform Resource Identifier (URI): Generic Syntax, Request for Comments 3986, January 2005. Available from http://www.ietf.org/rfc/rfc3986.txt.

[SchemaForXMLSchema]: XML Schema schema for XML Schemas: Part 1: Structures. Available from http://www.w3.org/2001/XMLSchema.xsd.

[SchemaforXMLSchemaInstance]: XML Schema instance namespace. Available from http://www.w3.org/2001/XMLSchema-instance.xsd

[Wikipedia]: Wikipedia, the free encyclopedia that anyone can edit. Available from http://en.wikipedia.org/.

Camel Case is described at http://en.wikipedia.org/w/index.php?title=CamelCase&oldid=230035120.

[XML]: Extensible Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation 16 August 2006. Available from http://www.w3.org/TR/2006/REC-xml-20060816/.

EBNF notation is described at #sec-notation.

IDREF constraint is described at #idref.

[XML-ID]: xml:id Version 1.0, W3C Proposed Recommendation 12 July 2005. Available from http://www.w3.org/TR/2005/PR-xml-id-20050712/.

[XMLInfoSet]: XML Information Set (Second Edition), W3C Recommendation 4 February 2004. Available from http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.

[XMLNamespaces]: Namespaces in XML, World Wide Web Consortium 16 August 2006. Available from http://www.w3.org/TR/2006/REC-xml-names-20060816.

NCName is described at #NT-NCName.

[XMLNamespacesErrata]: Namespaces in XML Errata, 6 December 2002. Available from http://www.w3.org/XML/xml-names-19990114-errata.

[XMLSchemaDatatypes]: XML Schema Part 2: Datatypes Second Edition, W3C Recommendation 28 October 2004. Available at http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[XMLSchemaStructures]: XML Schema Part 1: Structures Second Edition, W3C Recommendation 28 October 2004. Available from `http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/`.

Annotations are described at `#Annotation_details`.

[XSLT]: XSL Transformations (XSLT), Version 1.0, W3C Recommendation 16 November 1999. Available from `http://www.w3.org/TR/1999/REC-xslt-19991116`.

The element `xsl:sort` is described at `#element-sort`.

Appendix E: List of Principles

Appendix F: List of Definitions

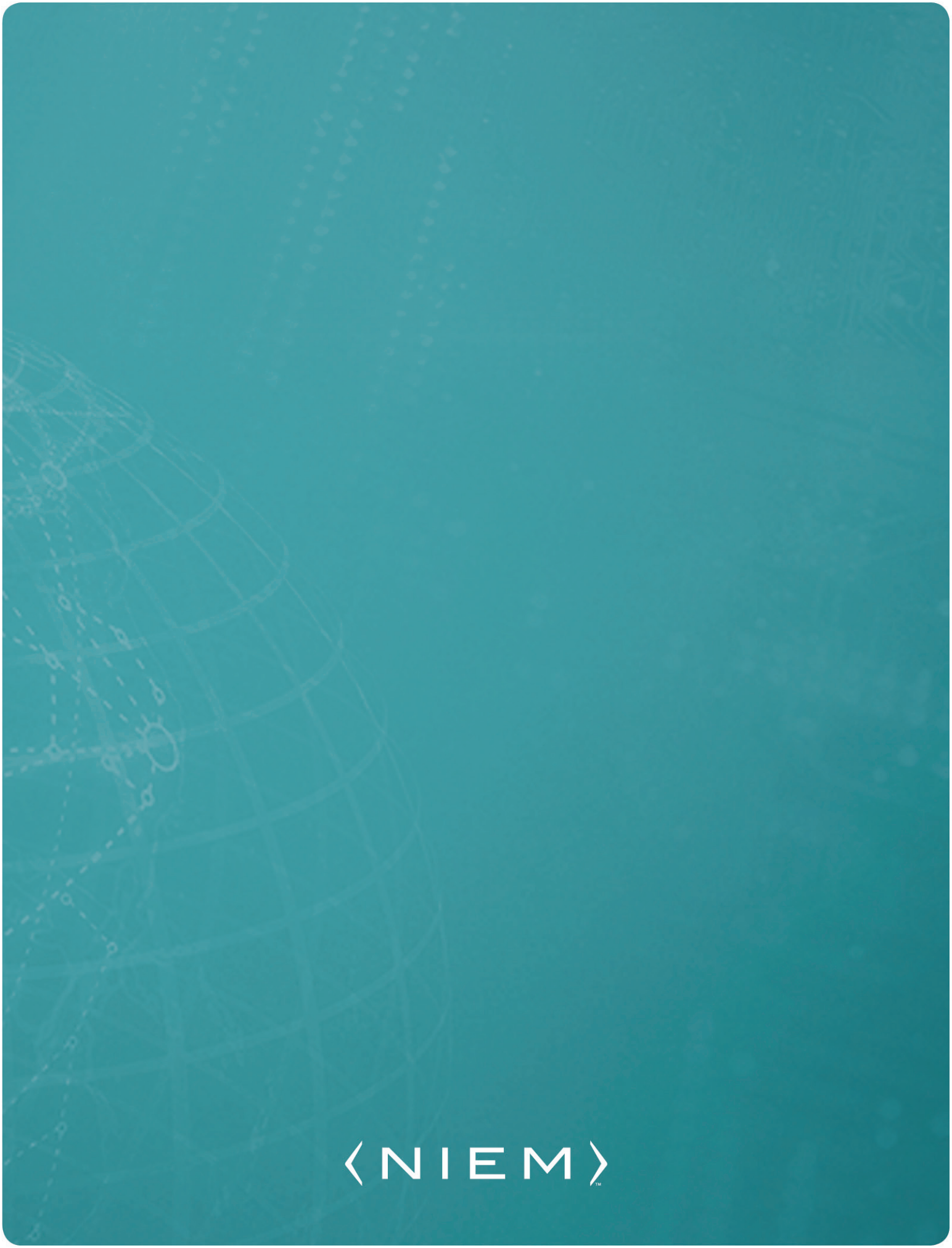
Appendix G: List of Rules

Appendix H: Index

This index points to important uses and definitions of key terms. It is not intended as a complete index of all uses of these terms.

Appendix I: Notices

This document and the information contained herein is provided on an “AS IS” basis and the authors DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.


NIEM