NIEM Naming and Design Rules (NDR) Version 6.0

> NOTE: I have copied the CSS styling from *styles/ndr-styles.css* to help your markdown previewer do the right thing. This must be removed before publication. The following code block must also be removed. (TODO).

```
If you want to check the line length in code blocks,
set the width of your markdown preview window so that the following line does not wrap,
and the terminal "X" is visible without the horizontal scrollbar:
.........1.........2.........3.........4.........5.........6.........7.........8.........9.........0.........1.........2.......X
This is the width of a code block when the OASIS-formatted HTML is printed to PDF by Microsoft Edge.
```

NIEM Naming and Design Rules (NDR) Version 6.0

Project Specification Draft 01

11 November 2024 draft

This stage:

https://docs.oasis-open.org/niemopen/ndr/v6.0/psd01/ndr-v6.0-psd01.md (Authoritative)
https://docs.oasis-open.org/niemopen/ndr/v6.0/psd01/ndr-v6.0-psd01.html
https://docs.oasis-open.org/niemopen/ndr/v6.0/psd01/ndr-v6.0-psd01.pdf

Previous stage:

N/A

Latest stage:

https://docs.oasis-open.org/niemopen/ndr/v6.0/ndr-v6.0.md (Authoritative)
https://docs.oasis-open.org/niemopen/ndr/v6.0/ndr-v6.0.html
https://docs.oasis-open.org/niemopen/ndr/v6.0/ndr-v6.0.pdf

Open Project:

OASIS NIEMOpen OP

Project Chair:

Katherine Escobar (katherine.b.escobar.civ@mail.mil), Joint Staff J6

NTAC Technical Steering Committee Chairs:

Scott Renner (sar@mitre.org), MITRE
James Cabral (jim@cabral.org), Individual

Editors:

Scott Renner (sar@mitre.org), MITRE
James Cabral (jim@cabral.org), Individual
Tom Carlson (Thomas.Carlson@gtri.gatech.edu), Georgia Tech Research Institute

Additional artifacts:s

- Other parts (list titles and/or file names)
- `(Note: Any normative computer language definitions that are part of the Work Product, such as XML instances, schemas and Java(TM) code, including fragments of such, must be (a) well formed and valid, (b) provided in separate plain text files, (c) referenced from the Work Product; and (d) where any definition in these separate files disagrees with the definition found in the specification, the definition in the separate file prevails. Remove this note before submitting for publication.)`

Related work:

This specification replaces or supersedes:

- *National Information Exchange Model Naming and Design Rules*. Version 5.0 December 18, 2020. NIEM Technical Architecture Committee (NTAC). https://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/niem-ndr-5.0.html.

This specification is related to:

- *NIEM Model Version 6.0*. Edited by Christina Medlin. Latest stage: https://docs.oasis-open.org/niemopen/niem-model/v6.0/niem-model-v6.0.html.
- Related specifications (include hyperlink, preferably to HTML format)

Abstract:

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.

Status:

Key words:

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] and [RFC8174] when, and only when, they appear in all capitals, as shown here.

Citation format:

When referencing this specification the following citation format should be used:

[NIEM-NDR-v6.0]

NIEM Naming and Design Rules (NDR) Version 6.0. Edited by Tom Carlson. 22 February 2024. OASIS Project Specification Draft 01. https://docs.oasis-open.org/niemopen/ndr/v6.0/psd01/ndr-v6.0-psd01.html. Latest stage: https://docs.oasis-open.org/niemopen/ndr/v6.0/ndr-v6.0.html.

Notices

Table of Contents

[[TOC will be inserted here]]

1 Introduction

NIEM, formerly known as the "National Information Exchange Model," is a framework for exchanging information among public and private sector organizations. The framework includes a reference data model for objects, properties, and relationships; and a set of technical specifications for using and extending the data model in information exchanges. The NIEM framework supports developer-level specifications of data that form a contract between developers. The data being specified is called a *message* in NIEM. While a message is usually something passed between applications, NIEM works equally well to specify an information resource published on the web, an input or output for a web service or remote procedure, and so forth, basically, any package of data that crosses a system or organization boundary.

NIEM promotes scalability and reusability of messages between information systems, allowing organizations to share data and information more efficiently. It was launched in 2005 in response to the U.S. Homeland Security Presidential Directives to improve information sharing between agencies following 9/11. Until 2023, NIEM was updated and maintained in a collaboration between the U.S. federal government, state and local government agencies, private sector, and non-profit and international organizations, with new versions released around once per year. NIEM defines a set of common objects, the *NIEM Core*, and 17 sets of objects that are specific to certain government or industry verticals, the *NIEM Domains*.

In 2023, NIEM became the NIEMOpen OASIS Open Project. NIEMOpen welcomes participation by anyone irrespective of affiliation with OASIS. Substantive contributions to NIEMOpen and feedback are invited from all parties, following the OASIS rules and the usual conventions for participation in GitHub public repository projects.

NIEMOpen is the term generally used when referring to the organization such as Project Governing Board (PGB), NIEMOpen Technical Architecture Committee (NTAC), NIEMOpen Business Architecture Committee (NBAC), organization activities or processes. NIEM is the term used when directly referring to the model i.e. NIEM Domain, NIEM Model version.

This document specifies principles and enforceable rules for NIEM data components and schemas. Schemas and components that obey the rules set forth here are conformant to specific conformance targets. Conformance targets may include more than the level of conformance defined by this NDR, and may include specific patterns of use, additional quality criteria, and requirements to reuse NIEM release schemas.

1.1 Changes from earlier Versions

This optional section provides a description of significant differences from previously published, differently numbered Versions of this specification, if any. (Detailed revision history of this numbered Version should be tracked in an Appendix.)

1.2 Glossary

1.2.1 Definitions of terms

Term	Definition
abstract class	A class that is a base for extension, and must be specialized to be used directly
adapter class	A class that contains only properties from a single external namespace [*see §4.4*]
artifact	
association class	Represents a specific relationship between objects
atomic class	Contains no object properties, one or more attribute properties, and exactly one data property that is not an attribute property. Does not have a special name.
attribute	
attribute declaration	
attribute property	A data property in which `isAttribute` is true
augmentable class	A class that can be augmented with additional properties

Term	Definition
augmentation element	Element based on an augmentation type that is substitutable for the augmentation point element in the augmented type.
augmentation point element	An abstract element declaration that provides a place for any augmtation properties.
augmentation type	Contains the augmenting properties
cardinality	How many times a property may/must appear in an object
class	Defines the content of a corresponding object (or resource) in a message.
code datatype	A restriction in which each value that is valid for the datatype corresponds to a code value in a code list.
code list	A list of distinct conceptual entities, each represented by a code value that has a known meaning beyond its text representation. These codes may be meaningful text or may be a string of alphanumeric identifiers that represent abbreviations for literals.
common model format (cmf)	A NIEM representation of the matamodel.
complete model	A model including a definition for every referenced component.
complex content	A object or type with child elements.
complex type	
component	
conformance target	A class of artifact, such as an interface, protocol, document, platform, process or service, that is the subject of conformance clauses and normative statements.
conformance target identifier	
conforming namespace	
constraint	
constraint rule	A requirement on an artifact with respect to its conformance to a conformance target.
data definition	A text definition of each component, describing what the component means.
data property	Defines a relationship between an object and a literal value.
datatype	Defines the allowed values of a corresponding atomic literal value in a message
deprecated	A component that is provided, but the use of which is not recommended.
document element	A property of a CMF object or XSD schema - defined by ref Infoset
documentation	A human-readable text documentation of a component.
documented component	A CMF object or XSD schema component that has an associated data definition.
element	
element declaration	
enumeration	
extension	

Term	Definition
extension namespace	Expresses the additional vocabulary required for an information exchange, above and beyond the vocabulary available from the NIEM model
extension schema document	
external adapter type	A set of data that embodies a single concept from an external standard.
external components	A schema component defined by an external schema document.
external namespace	
external schema document	Non-NIEM-conformant schema documents.
facet	A data concept for a facet that restricts an aspect of a data type.
imported namespace	
informative	Material that appears as supporting text, description, and rationales for the normative material.
interpretation rule	Defines a methodology, pattern, or procedure for understanding some aspect of an instance of a conformance target.
json	
json document	
json schema	
json text	
json-ld	
list	An object that defines a datatype as a whitespace-separated list of atomic values.
literal	
local term	A word, phrase, acronym, or other string of characters that is used in the name of a namespace component, but that is not defined in OED, or that has a non-OED definition in this nameespace, or has a word sense that is in some way unclear.
local type	
message	A package of data shared at runtime; a sequence of bits that convey information to be exchanged or shared. (see section 3.1.1)
message designer	A person that creates a message type and format from an information requirement, so that a message at runtime will contain all the facts that need to be conveyed.
message developer	A person that writes software to implement a message specification, producing or processing messages that conform to the message format.
message format	A specification of the valid syntax of messages. (see section 3.1.2)
message schema document	A XSD representation of a message namespace
message specification	A collection of related message formats and types. (see section 3.1.4)
message type	A specification of the information content of messages. (see section 3.1.3)
metadata	
mixed content	The mixing of data tags with text

Term	Definition
model object	Represents a complete or partial NIEM model.
namespace	A collection of uniquely-named components.
namespace author	
namespace prefix	
namespace uri	A unique URI for a namespace.
niem	A framework for exchanging information among public and private sector organizations.
niem core	A set of common objects.
niem domain	A set of objects that are specific to certain government or industry vertical
niem metamodel	An abstract model for NIEM data models.
niemopen business architecture committee (nbac)	
niemopen project governing board (pgb)	
niemopen technical architecture committee (ntac)	
normative	Required for conformance (e.g. rules).
object class	Represents a class of objects defined by a NIEM model
object class term	
object property	Defines a subject-predicate-value relationship between an object and another object.
ordered/ordinality	
particle	
property term	Describes or represents a characteristic or subpart of an entity or concept.
proxy type	A complex type definition wrapper for a simple type in the XML Schema namespace.
qualifier term	Provide additional context to resolve subtleties between properties of objects.
range	The class or datatype of a property.
reference attribute	A pointer to an element in a message.
reference data model	A model of objects, properties, and relationships
reference namespace	Namespaces that include all components in the NIEM model.
reference property	An attribute property that contains a reference to an object in a message.
reference schema document	
representation term	Indicate the style of component, prevents name conflicts or indicates the nature of the value.
restriction object	Defines a datatype as a restriction of a base datatype plus zero or more restricting Facet objects.

Term	Definition
schema	
schema document	
serialization	*(Verb)* A process of converting a data structure into a sequence of bits that can be stored or transferred. *(Noun)* A standard for the output of serialization; for example, XML and JSON.
simple content	An object or type that has only attributes, not elements.
simple type	
subset	A collection of NIEM components required for an information exchange.
subset namespace	A subset of the components in a reference or extension namespace.
subset schema document	
target namespace	
type definition	
union	Defines a datatype as the union of one or more datatypes.
xml catalog	
xml schema document set	Defines an XML Schema that may be used to validate an XML document

1.2.2 Acronyms and abbreviations

Term	Literal
APPINFO	Application Information
CCC	Complex type with Complex Content
CMF	Common Model Format
CSC	Complex type with Simple Content
CSV	Comma Separated Values
CTAS	Conformance Targets Attribute Specification
EXT	Extension namespace conformance target
ID	Identifier
IEP	Information Exchange Package
IEPD	Information Exchange Package Documentation
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
JSON-LD	JavaScript Object Notation Linked Data
MSG	Message namespace conformance target
N5R	NIEM 5 Rule
NBAC	NIEMOpen Business Architecture Committee
NC	NIEM Core

Term	Literal
NIEM	Formerly National Information Exchange Model
NS	Namespace
NTAC	NIEMOpen Technical Architecture Committee
OED	Oxford English Dictionary
OP	Open Project
OWL	Web Ontology Language
PGB	Project Governing Board
Qname	Qualified Name
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
REF	Reference namespace conformance target
RFC	Request For Comments
SUB	Subset namespace conformance target
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
XML	Extensible Markup Language
XSD	XML Schema Definition

1.2.3 Document conventions

- Naming conventions
- Font colors and styles
- Typographic conventions

2. How To Read This Document

This document provides normative specifications for NIEM-conforming data models. It also describes the goals and principles behind those specifications. It includes examples and explanations to help users of NIEM understand the goals, principles, and specifications. The relevant sections of this document will depend on the role of the user. Figure 2-1 illustrates the relationships between these roles and NIEM activites.

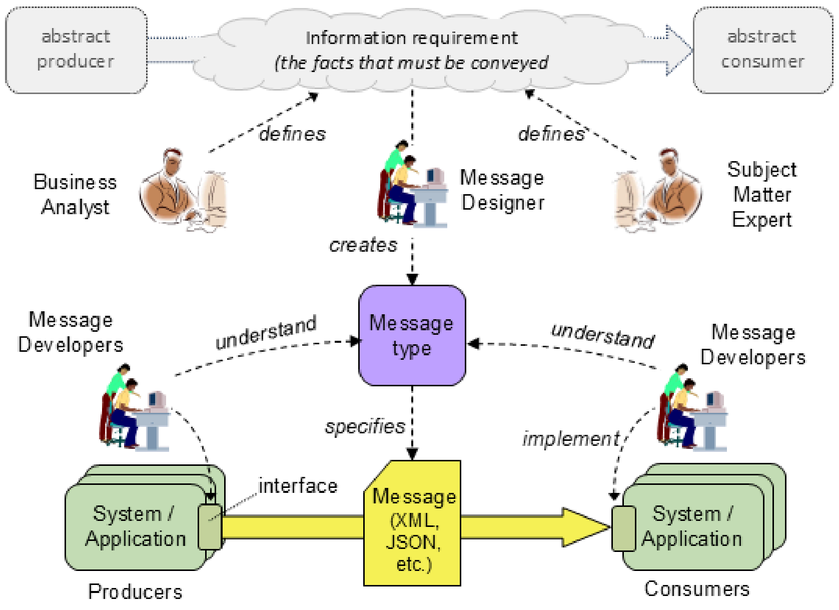


Figure 2-1: User roles and activities

The user roles in the above figure are:

- *Business analysts* and *subject matter experts*, who provide the requirements for information transfer. These requirements might describe an information resource available to all comers. They could describe an information exchange as part of a business process. They need not be tied to known producers and consumers.
- *Message designers*, who express those requirements as a ·message type·, which specifies the syntax and semantics of the data that will convey the required information at runtime.
- *Message developers*, who write software to construct messages that contain the required information and follows the defined syntax, and who write software to parse and process such messages.

The remaining sections of this document most relevant to each of these roles are shown in the following table:

Section	Manager	Business Analyst	Message Designer	Message Developer
3. Overview of NIEM technical architecture	x	x	x	x
4. Data models in NIEM			x	
5. Data modeling patterns			x	
6. Conformance		x	x	x
7. Rules for names of components		x	x	
8. Rules for documentation		x	x	
9. Rules for the NIEM profile of XSD			x	
10. Rules for models in XSD			x	
11. Rules for NIEM messages in XML			x	x

Section	Manager	Business Analyst	Message Designer	Message Developer
12. Rules for the NIEM profile of JSON-LD			x	
13. Rules for NIEM messages in JSON			x	x

2.1 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 3986 is shown as [RFC 3986]. All reference documents are recorded in Appendix B, References, below.

2.2 Clark notation and qualified names

This document uses both Clark notation and QName notation to represent qualified names.

QName notation is defined by **[XML Namespaces]** Section 4, Qualified Names. A QName for the XML Schema string datatype is xs:string. Namespace prefixes used within this specification are listed in Section 2.3, Use of namespaces and namespace prefixes, below.

This document sometimes uses Clark notation to represent qualified names in normative text. Clark notation is described by **[ClarkNS]**, and provides the information in a QName without the need to first define a namespace prefix, and then to reference that namespace prefix. A Clark notation representation for the qualified name for the XML Schema string datatype is `{http://www.w3.org/2001/XMLSchema}string`.

Each Clark notation value usually consists of a namespace URI surrounded by curly braces, concatenated with a local name. The exception to this is when Clark notation is used to represent the qualified name for an attribute with no namespace, which is ambiguous when represented using QName notation. For example, the element targetNamespace, which has no [namespace name] property, is represented in Clark notation as `{}targetNamespace`.

2.3 Use of namespaces and namespace prefixes

The following namespace prefixes are used consistently within this specification. These prefixes are not normative; this document issues no requirement that these prefixes be used in any conformant artifact. Although there is no requirement for a schema or XML document to use a particular namespace prefix, the meaning of the following namespace prefixes have fixed meaning in this document.

- `xs`: The namespace for the XML Schema definition language as defined by [XML Schema Structures] and [XML Schema Datatypes], http://www.w3.org/2001/XMLSchema.
- `xsi`: The XML Schema instance namespace, defined by [XML Schema Structures] Section 2.6, Schema-Related Markup in Documents Being Validated, for use in XML documents, http://www.w3.org/2001/XMLSchema-instance.
- `ct`: The namespace defined by [CTAS] for the conformanceTargets attribute, https://docs.oasis-open.org/niemopen/ns/specification/conformanceTargets/6.0/.
- `appinfo`: The namespace for the appinfo namespace, https://docs.oasis-open.org/niemopen/ns/model/appinfo/6.0/.
- `structures`: The namespace for the structures namespace, https://docs.oasis-open.org/niemopen/ns/model/structures/6.0/.
- `cmf`: The namespace for the CMF model representation, https://docs.oasis-open.org/niemopen/ns/specification/cmf/1.0/.

3. Overview of the NIEM Technical Architecture

This overview describes NIEM's design goals and principles, and introduces key features of the architecture. The major design goals are:

- *Shared understanding of data.* NIEM helps developers working on different systems to understand the data their systems share with each other.
- *Reuse of community-agreed data definitions.* NIEM reduces the cost of data interoperability by promoting shared data definitions - without requiring a single data model of everything for everyone.
- *Open standards with free-and-open-source developer tools.* NIEM does not depend on proprietary standards or the use of expensive developer tools.

The key architecture features mentioned in section 3 are:

- *The NIEM metamodel* - an abstract, technology-neutral data model for NIEM data models
- *Two equivalent model representations* - One is a profile of XML Schema (XSD) that has been used in every version of NIEM. The other is itself a NIEM-based data specification, suitable for XML and many other data technologies.
- *Model namespaces* - for model configuration management by multiple authors working independently.

3.1 Machine-to-machine data specifications

NIEM is a framework for developer-level specifications of data. A NIEM-based data specification – which is built *using* NIEM and in *conformance* to NIEM, but is not itself a *part* of NIEM – describes data to the developers of producing and consuming systems. This data may be shared via:

- a ·message· passed between applications
- an information resource published on the web
- an API for a system or service

NIEM is potentially useful for any data sharing mechanism that transfers data across a system or organization boundary. (Within a system, NIEM may be useful when data passes between system components belonging to different developer teams.)

The primary purpose of a NIEM-based data specification is to establish a common understanding among developers, so that they can write software that correctly handles the shared data, hence "machine-to-machine". (NIEM-conforming data may also be directly presented to human consumers, and NIEM can help these consumers understand what they see, but that is not the primary purpose of NIEM.)

Data sharing in NIEM is implemented in terms of messages, message formats, and message types. These are illustrated in figure 3-1.

- ·message· - a package of data shared at runtime; an instance of a ·message format· and type
- ·message format· - a definition of a syntax for the messages of a ·message type·
- ·message type· - a definition of the information content in equivalent ·message formats·

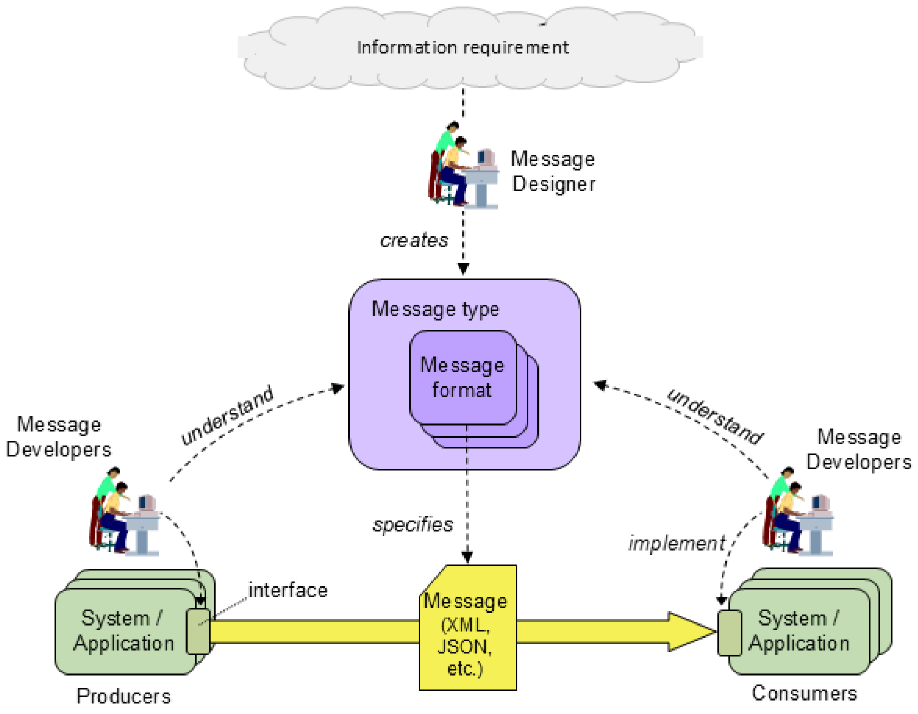


Figure 3-1: Message types, message formats, and messages

A message designer turns information requirements into a ·message type·, then turns a ·message type· into one or more ·message formats·. Message developers then use the ·message type· and ·message format· to understand how to implmement software that produces or consumes conforming messages.

3.1.1 Messages

In NIEM terms, the package of data shared at runtime is a ·message·. This data is arranged according to a supported ·serialization·. The result is a sequence of bits that represent the information content of the message. Figure 3-2 shows two messages representing the same information, one serialized in XML, the other in JSON. Each message in this example is a request for a quantity of some item. (In all examples, closing tags and brackets may be omitted, and long lines may be truncated.)

```
<msg:Request                                                 | {
 xmlns:nc="https://docs.oasis-open.org/niemopen/ns/model/    |   "@context": "http://example.com/ReqRes/Request/JSON/1.0",
 xmlns:msg="http://example.com/ReqRes/1.0/">                 |   "msg:Request": {
  <msg:RequestID>RQ001</msg:RequestID>                       |     "msg:RequestID" : "RQ001",
  <msg:RequestedItem>                                        |     "msg:RequestedItem": {
    <nc:ItemName>Wrench</nc:ItemName>                        |       "nc:ItemName": Wrench",
    <nc:ItemQuantity>10</nc:ItemQuantity>                    |       "nc:ItemQuantity": 10
  </msg:RequestedItem>                                       |     }
</msg:Request>                                               |   }
                                                             | }
```

Figure 3-2: Example of messages in XML and JSON syntax

The data structure of a NIEM message appears to be a tree with a root node. It is actually a directed graph with an ·initial node·. The ·initial node· in figure 3-2 is the `msg:Request` element in the XML message. In the JSON message it is the value for the `msg:Request` key.

Every NIEM serialization has a mechanism for references; that is, a way for one node in the serialized graph to point to a node elsewhere in the graph. This mechanism supports cycles and avoids duplication in the graph data structure. (See section TODO.)

Every ·message· is an instance of a ·message format·. A conforming message must satisfy the rules in section 11; in particular, it must be valid according to the ·schema· of its ·message format·.

> A NIEM message was originally known as an *information exchange package (IEP)*, a term that found its way into the U.S. Federal Enterprise Architecture (2005). A ·message specification· was originally known

> as an *information exchange package documentation (IEPD).* These terms are in widespread use within the NIEM community today, and will not go away soon (if ever).

3.1.2 Message format

A ·message format· specifies the syntax of valid messages. This provides message developers with an exact description of the messages to be generated or processed by their software.

A ·message format· includes a ·schema· that can be used to assess the validity of a ·message·. This ·schema· is expressed in XML Schema (XSD) for XML message formats, and JSON Schema for JSON message formats. Figure 3-3 shows a portion of the schemas for the two example messages in figure 3-2.

```
<xs:complexType name="RequestType">                                  | JSON Schema example TODO
  <xs:sequence>                                                      |
    <xs:element ref="msg:RequestID"/>                                |
    <xs:element ref="msg:RequestedItem"/>                            |
  </xs:sequence>                                                     |
</xs:complexType>                                                    |
<xs:element name="Request" type="msg:RequestType"/>                  |
```

Figure 3-3: Example of message format schemas

Producing and consuming systems may use the message format schema to validate the syntax of messages at runtime, but are not obligated to do so. Message developers may also use the schema during development for software testing. The schemas may also be used by developers for data binding; for example, JAXB with XSD schemas.

A ·message format· belongs to exactly one ·message type·. A conforming ·message format· must satisfy the rules in section 10; in particular, it must be constructed so that every ·message· that is valid according to the format also satisfies the information content constraints of its ·message type·.

3.1.3 Message type

One important feature of NIEM is that every ·message· has an equivalent ·message· in every other supported serialization. These equivalent messages have a different ·message format·, but have the same ·message type·. For example, the messages in figure 3-2 above are equivalent. They represent the same information content, and can be converted one to the other without loss of information.

A ·message type· specifies the information content of its messages without prescribing their syntax. A ·message type· includes a ·message model·, which precisely defines the mandatory and optional content of its messages and the meaning of that content. This model is expressed in either of NIEM's two model representations, which are described in section 3.5 and section 3.6, and fully defined in section 4. Figure 3-4 shows a portion of the message model for the two message formats in figure 3-3.

```
<xs:complexType name="ItemType" appinfo:referenceCode="NONE"> | <Class structures:id="nc.ItemType">
  <xs:annotation>                                             |   <Name>ItemType</Name>
    <xs:documentation>A data type for an article or thing.    |   <Namespace structures:ref="nc" xsi:nil="true"/>
  </xs:annotation>                                            |   <DocumentationText>A data type for an article or th
  <xs:sequence>                                               |   <ReferenceCode>NONE</ReferenceCode>
    <xs:element ref="nc:ItemName"/>                           |   <PropertyAssociation>
    <xs:element ref="nc:ItemQuantity"/>                       |     <DataProperty structures:ref="nc.ItemName" xsi:nil="true"/>
  </xs:sequence>                                              |     <MinOccursQuantity>1</MinOccursQuantity>
</xs:complexType>                                             |     <MaxOccursQuantity>1</MaxOccursQuantity>
<xs:element name="ItemName" type="nc:TextType">               |   </PropertyAssociation>
  <xs:annotation>                                             |   <PropertyAssociation>
    <xs:documentation>A name of an item.</xs:documentation>   |     <DataProperty structures:ref="nc.ItemQuantity"
  </xs:annotation>                                            |     <MinOccursQuantity>1</MinOccursQuantity>
</xs:element>                                                 |     <MaxOccursQuantity>1</MaxOccursQuantity>
<xs:element name="RequestedItem" type="nc:ItemType">          |   </PropertyAssociation>
  <xs:annotation>                                             | </Class>
    <xs:documentation>A specification of an item request.</xs | <<DataProperty structures:id="nc.ItemName">
  </xs:annotation>                                            |   <Name>ItemName</Name>
</xs:element>                                                 |   <Namespace structures:ref="nc" xsi:nil="true"/>
                                                              |   <DocumentationText>A name of an item.
                                                              |   <Datatype structures:ref="nc.TextType" xsi:nil="true"/>
                                                              | </DataProperty>
|   <ObjectProperty structures:id="msg.RequestedItem">
|   <Name>RequestedItem</Name>
|   <Namespace structures:ref="msg" xsi:nil="true"/>
|   <DocumentationText>A specification of an item
|   <Class structures:ref="nc.ItemType" xsi:nil="true"/>
|   <ReferenceCode>NONE</ReferenceCode>
| </ObjectProperty>
```

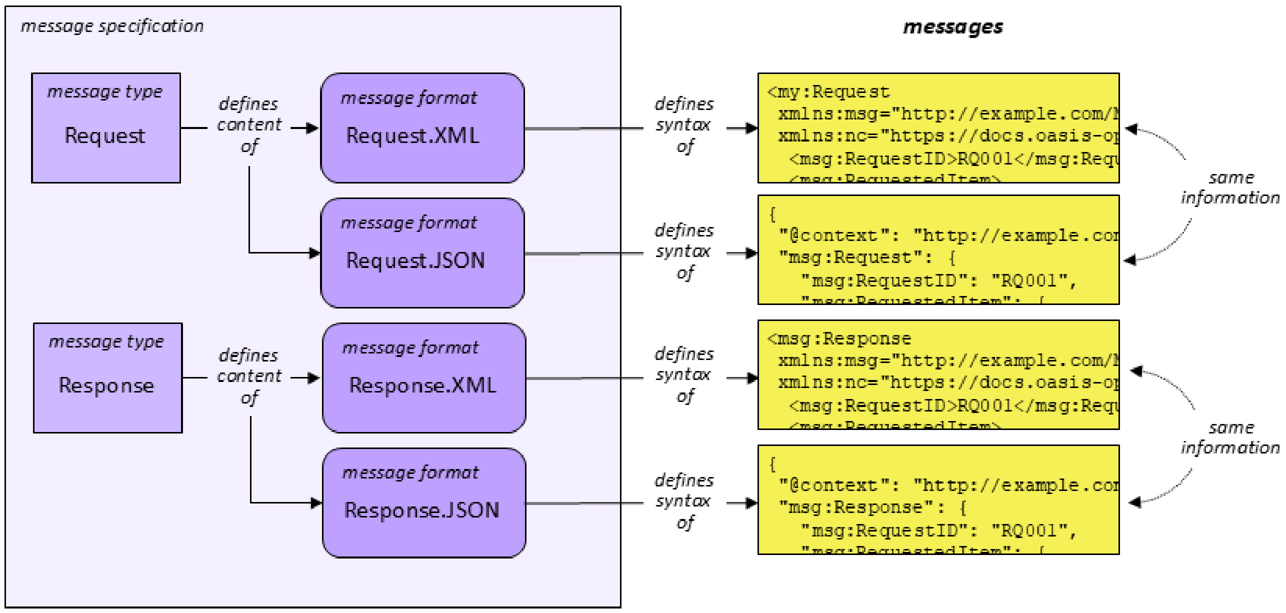
Figure 3-4: Example message model in XSD and CMF

A ·message type· provides all of the information needed to generate the schema for each ·message format· it specifies. NIEMOpen provides free and open-source software tools to generate these schemas from the message model. (Messsage designers are also free to compose these schemas by hand.)

A conforming ·message type· must satisfy all of the rules in section 10.

3.1.4 Message specification

A ·message specification· is a collection of related message types. For instance, the example Request message above might be paired with a Response message as part of a request/response protocol. Those two message types could be collected into a ·message specification· for the protocol, as illustrated below in figure 3-5.



```
<my:Request
 xmlns:msg="http://example.com/N
 xmlns:nc="https://docs.oasis-op
  <msg:RequestID>RQ001</msg:Requ
```

```
{
 "@context": "http://example.com
 "msg:Request": {
   "msg:RequestID": "RQ001",
```

```
<msg:Response
 xmlns:msg="http://example.com/N
 xmlns:nc="https://docs.oasis-op
  <msg:RequestID>RQ001</msg:Requ
```

```
{
 "@context": "http://example.com
 "msg:Response": {
   "msg:RequestID": "RQ001",
```

Figure 3-5: Message specifications, types, and formats

Summary:

- A ·message specification· defines one or more ·message types·; a ·message type· belongs to one ·message specification·
- A ·message type· defines one or more ·message formats·; a ·message format· belongs to one [message type[(#def)]
- A ·message format· defines the syntax of valid ·messages·
- A ·message type· defines the semantics of valid messages, plus their mandatory and optional content
- A ·message· is an instance of a ·message format· and of that format's ·message type·

3.2 Reuse of community-agreed data models

NIEM is also a framework for communities to create ·reuse models· for concepts that are useful in multiple data specifications. These community models are typically not *complete* for any particular specification. Instead, they reflect the community's judgement on which definitions are *worth the trouble of agreement*. The NIEM core model contains definitions found useful by the NIEM community as a whole. NIEM domain models reuse the core, extending it with definitions found useful by the domain community. The core model plus the domain models comprise the "NIEM model". Figure 3-6 below illustrates the relationships between domain communities and community models.

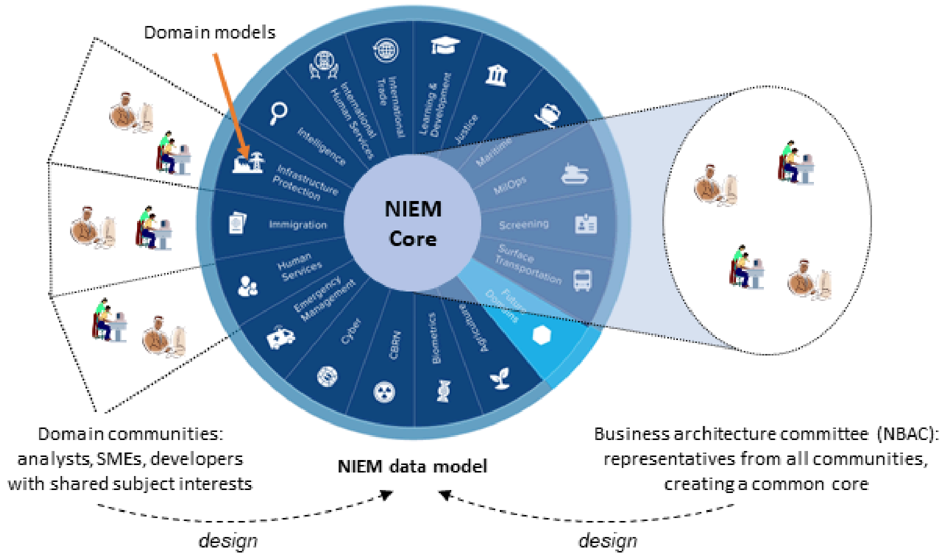


Figure 3-6: NIEM communities and data models

Message designers reuse definitions from the NIEM model, selecting a (usually small) subset of definitions that express part of their information requirement. Message designers then create model extensions, adding components that do not yet exist in NIEM. Local extensions that could be useful to others in the community beyond the scope of the original message can be submitted for potential adoption into the NIEM model (https://github.com/niemopen/niem-model/issues).

Data model reuse is especially useful in a large enterprise. Its value grows with the number of developer teams, and with the degree of commonality in the shared data. NIEM was originally designed for data sharing among federal, state, and local governments – where commonality and number of developer teams is large indeed.

3.3 Reuse of open standards

NIEM is built on a foundation of open standards, primarily:

- XML and XSD – message serialization and validation; also a modeling formalism
- JSON and JSON-LD – message serialization and linked data
- JSON Schema - message validation
- RDF, RDFS, and OWL – formal semantics
- ISO 11179 – data element conventions

One of NIEM's principles is to reuse well-known information technology standards when these are supported by free and open-source software. NIEM avoids reuse of standards that effectively depend on proprietary software. When the NIEMOpen project defines a standard of its own, it also provides free and open-source software to support it.

3.4 The NIEM metamodel

A data model in NIEM is either a ·message model·, defining the information content of a ·message type·, or a ·reuse model·, making the agreed definitions of a community available for reuse. The information required for those purposes can itself be modeled. The model of that information is the *NIEM metamodel* -- an abstract model for NIEM data models. The metamodel is expressed in UML, and is described in detail in section 4. At a high level, the major components of the metamodel are properties, classes, datatypes, namespaces, and models. Figure 3-7 provides an Illustration.

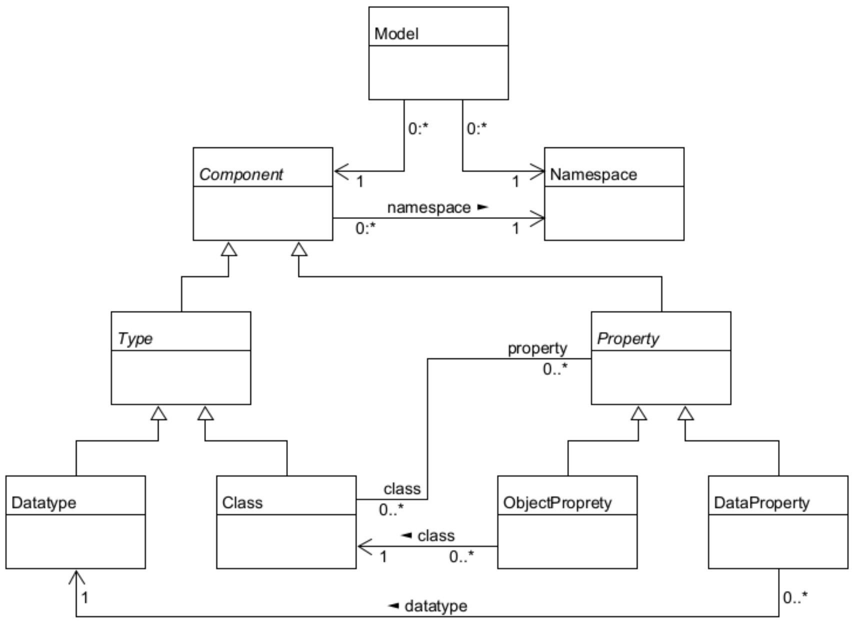


Figure 3-7: High-level view of the NIEM metamodel

Examples TODO

- A *property* is a concept, idea, or thing. It defines a field that may appear in a ·message· and can contain subfields (for objects / object properties) or a value (for literals / data properties). For example, in figure 3-4, `req:RequestedItem` and `nc:ItemName` are names of properties. `req:RequestedItem` is an object property for the requested item; `nc:ItemName` is a data property for the name of the item. The meaning of these properties is captured in the documentation text.

- A *class* defines the properties that may appear in the content of a corresponding *object* in a ·message·. A class has one or more *properties*. An *object property* in a class defines a subject-property-value relationship between two objectst. A *data property* defines a relationship between an object and a literal value. In figure 3-4, `nc:ItemType` is the name of a class.

- A *datatype* defines the allowed values of a corresponding atomic *literal value* in a ·message·. In figure 3-4, `nc:TextType` is the name of a datatype.

- Classes and datatypes are the two kinds of *type* in the metamodel. For historical reasons, the name of every class and datatype in the NIEM model ends in "Type". This is why the high-level view of the metamodel includes the abstract Type UML class.

- Classes, datatypes, and properties are the three kinds of metamodel *component*. (All of the common properties of classes and datatypes are defined in the Component class, which is why the abstract Type class is not needed in the detailed metamodel diagram in section 4.)

- A *namespace* is a collection of uniquely-named components defined by an authority. (See section 3.6)

- A *model* is a collection of components (organized into namespaces) and their relationships.

Figure 3-8 below illustrates the relationships among metamodel components, NIEM model components, and the corresponding ·message· objects and values.

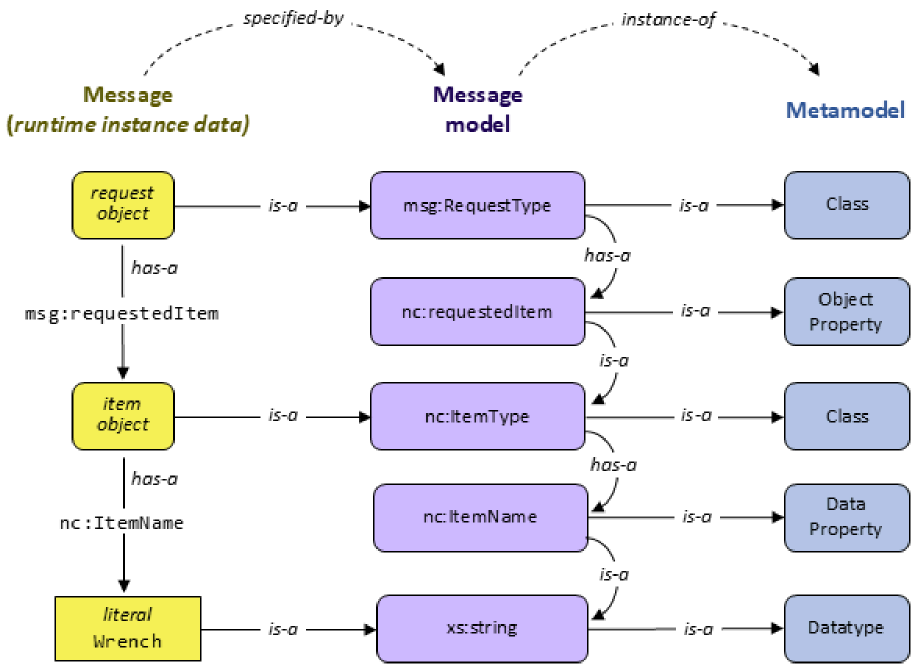


Figure 3-8: Message, message model, and metamodel relationships

A NIEM ·message· contains properties which are based on objects or literal values. These are specified by the Class, Property, and Datatype objects in a NIEM ·message model·, which defines the content of a conforming ·message· and also defines the meaning of that content. In figure 3-8, the *item object* is defined by the `nc:ItemType` Class object; the *literal value* (`Wrench`) is defined by the `xs:string` Datatype object, and the property relationship between the two is defined by the `nc:ItemName` DataProperty object.

3.5 NIEM model representations: XSD and CMF

The abstract metamodel has two concrete representations: NIEM XSD and NIEM CMF. These are equivalent representations and may be converted from one to the other without loss. (NIEMOpen provides free and open-source software tools that perform the conversion; see **[Tools]**.)

Every version of NIEM has used a profile of XML Schema (XSD) as a NIEM model representation. In XSD, a NIEM model is represented as a schema assembled from a collection of schema documents. Every aspect of the metamodel is represented in some way by a schema component.

XSD as a model representation directly supports conformance testing of NIEM XML messages through schema validation. However, JSON developers (and developers working with other formats) cannot use XSD to validate their messages. Nor do they want to read XSD specifications of message content.

NIEM 6 introduces the Common Model Format (CMF), a NIEM model representation intended to support all developers. CMF is the result of applying the NIEM framework to the information requirements in the metamodel. That result is a NIEM-based ·message type·, which is defined in **[CMF]**. In CMF, a model is represented as an instance of that ·message type·; that is, a CMF ·message·, also known as a ·model file·.

CMF is a technology-neutral model representation, because:

- A CMF model can be transformed into XSD for validation of XML messages, and into JSON Schema for validation of JSON messages.
- A CMF model can itself be represented in XML or JSON, according to developer preference. That is, like any other NIEM message, the CMF representation of a model can be serialized in either XML or JSON. For example, figure 3-9 shows a portion of the message model from figure 3-4 in both XML and JSON syntax.

```
<Class structures:id="nc.ItemType">                                   | {
  <Name>ItemType</Name>                                               |   "cmf:Class": {
  <Namespace structures:ref="nc" xsi:nil="true"/>                     |     "cmf:Name": "ItemType",
  <DocumentationText>A data type for an article or thing.</Docum |     "cmf:Namespace": { "@id": "#nc" },
  <ReferenceCode>NONE</ReferenceCode>                                 |     "cmf:DocumentationText": "A data type for an article
  <PropertyAssociation>                                               |     "cmf:ReferenceCode": "NONE",
    <DataProperty structures:ref="nc.ItemName" xsi:nil="true"/>       |     "cmf:PropertyAssociation": {
```

```
    <MinOccursQuantity>1</MinOccursQuantity>                  |       "cmf:DataProperty": { "@id": "#nc.ItemName" },
    <MaxOccursQuantity>1</MaxOccursQuantity>                  |       "cmf:MinOccursQuantity": 1,
  </PropertyAssociation>                                      |       "cmf:MaxOccursQuantity": 1
  <PropertyAssociation>                                       |     },
    <DataProperty structures:ref="nc.ItemQuantity" xsi:nil="true |     "cmf:PropertyAssociation": {
    <MinOccursQuantity>1</MinOccursQuantity>                  |       "cmf:DataProperty": { "@id": "#nc.ItemQuantity" },
    <MaxOccursQuantity>1</MaxOccursQuantity>                  |       "cmf:MinOccursQuantity": 1,
  </PropertyAssociation>                                      |       "cmf:MaxOccursQuantity": 1
</Class>                                                      |     }
                                                              |   }
                                                              | }
```

Figure 3-9: CMF model in XML and JSON syntax

Section 4 defines the mappings between the metamodel, NIEM XSD, and CMF.

> While NIEM uses JSON Schema to validate JSON messages, there is no JSON Schema representation of the metamodel, because JSON Schema does not have all of the necessary features for NIEM models.

3.6 Namespaces

The components of a NIEM model are partitioned into *namespaces.* This prevents name clashes among communities or domains that have different business perspectives, even when they choose identical data names to represent different data concepts.

Each namespace has an author, a person or organization that is the authoritative source for the namespace definitions. A namespace is the collection of model components for concepts of interest to the namespace author. Namespace cohesion is important: a namespace should be designed so that its components are consistent, may be used together, and may be updated at the same time.

Each namespace must be uniquely identified by a URI. The namespace author should also be the URI's owner, as defined by **[webarch]**. Both URNs and URLs are allowed. It is helpful, but not required, for the namespace URI to be accessible, returning the definition of the namespace content in a supported model format. (See **[repositoriesTODO]** for an alternative way to obtain namespace definitions.)

NIEM defines two categories of authoritative namespace: ·reference namespace· and ·extension namespace·.

- *Reference namespace:* The NIEM model is a ·reuse model· comprised entirely of ·reference namespaces·. The components in these namespaces are intended for the widest possible reuse. They provide names and definitions for concepts, and relations among them. These namespaces are characterized by "optionality and over-inclusiveness". That is, they define more concepts than needed for any particular data exchange specification, without cardinality constraints, so it is easy to select the concepts that are needed and omit the rest. They also omit unnecessary range or length constraints on property datatypes.

  A ·reference namespace· is intended to capture the meaning of its components. It is not intended for a complete definition of any particular ·message type·. Message designers are expected to subset, profile, and extend the components in ·reference namespaces· as needed to match their information exchange requirements.

- *Extension namespace:* The components in an ·extension namespace· are intended for reuse within a more narrow scope than those defined in a ·reference namespace·. These components express the additional vocabulary required for an information exchange, above and beyond the vocabulary available from the NIEM model. The intended scope is often a particular ·message specification·. Sometimes a community or organization will define an ·extension namespace· for components to be reused in several related message specifications. In this case, the namespace components may also omit cardinality and datatype constraints, and may be incomplete for any particular ·message type·.

  Message designers are encouraged to subset, profile, and extend the components in ·extension namespaces· created by another author when these satisfy their modeling needs, rather than create new components.

Namespaces are the units of model configuration management. Once published, the components in a ·reference namespace· or ·extension namespace· may not be removed or changed in meaning. A change of that nature may only be made in a new namespace with a different URI. As a result of this rule, a change by the author of namespace X does not force a change by the author of any other namespace Y. Instead, the author of namespace Y may continue to use components from namespace X as long as desired; the revised namespace X' may be adopted, if desired, whenever convenient.

Message designers almost never require *all* the components in the NIEM model, and so NIEM defines a third namespace category:

- *Subset namespace:* Technically, this is a "namespace subset", which contains only some of the components of a ·reference namespace· or ·extension namespace·. It provides components for reuse, while enabling message designers and developers to:
    - Omit optional components in a ·reference namespace· or ·extension namespace· that they do not need.
    - Provide cardinality and datatype constraints that precisely define the content of one or more message types.
    - Augment a ·reference namespace· or ·extension namespace· with an ·attribute property·.

  All message content that is valid for a subset namespace must also be valid for the ·reference namespace· or ·extension namespace· with the same URI. Widening the value space of a component is not allowed. With the exception of attribute augmentations, adding components is not allowed. Changing the documentation of a component is not allowed.

NIEM has a fourth namespace category, for namespaces containing components from standards or specifications that are based on XML but not based on NIEM.

- *External namespace:* Any namespace defined by a ·schema document· that is not:
    - a ·reference namespace·
    - an ·extension namespace·
    - a ·subset namespace·
    - the ·structures namespace·, `https://docs.oasis-open.org/niemopen/ns/model/structures/6.0/`
    - the XML namespace, `http://www.w3.org/XML/1998/namespace`.

  XML attributes defined in an external namespace may be part of a NIEM model. XML elements defined in an external namespace are not part of a NIEM model, but may be used as properties of an ·adapter type·; see section TODO.

Three special namespaces do not fit into any of the four categories:

- The ·structures namespace· is not part of any NIEM model. It provides base types and attributes that are used in the XSD representation of NIEM models.
- The XML namespace is not considered to be an external namespace. It defines the `xml:lang` attribute, which may be a component in a NIEM model.
- The XSD namespace (`http://www.w3.org/2001/XMLSchema`) defines the primitive datatypes (`xs:string`, etc.) This namespace appears explicitly in CMF model representations, and is implicitly part of every XSD representation.

4. Data models in NIEM

The NIEM metamodel is an abstract model that specifies the content of a NIEM data model. It is described by the UML diagram in figure 4-1 below.

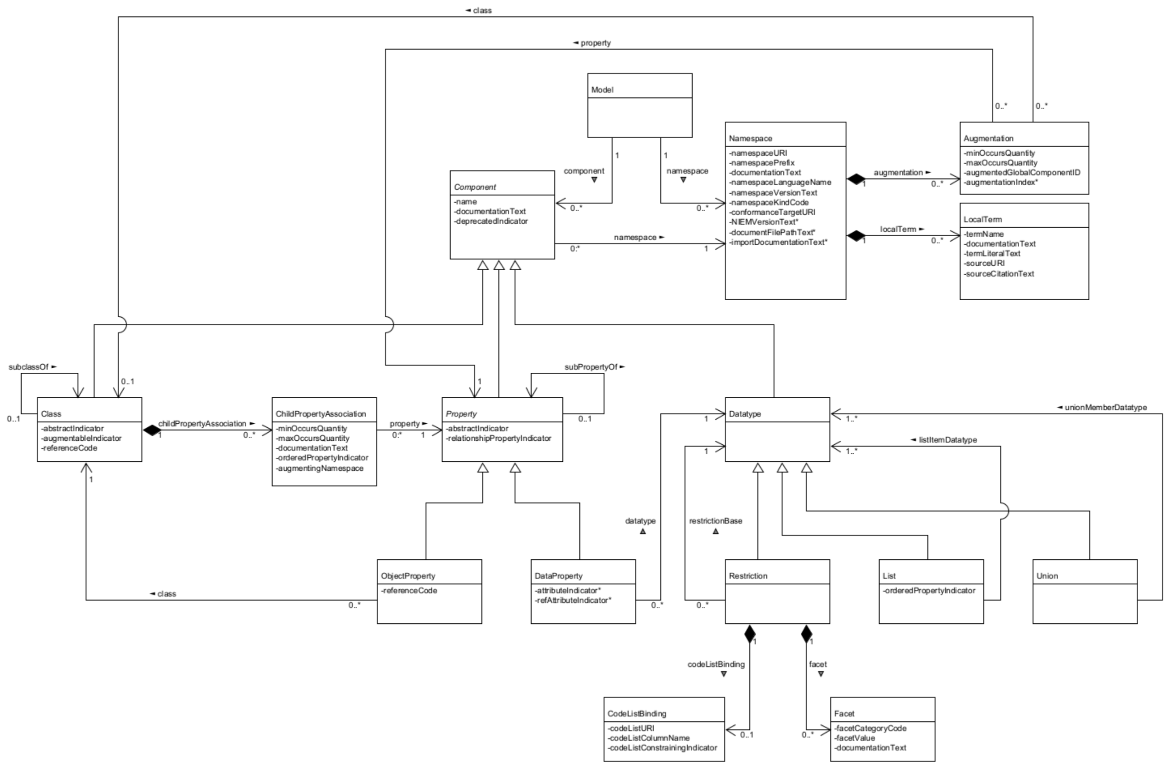


Figure 4-1:The NIEM metamodel

This section specifies:

- the meaning of the classes, attributes, and relationships in the metamodel
- the meaning of the classes, datatypes, and properties in CMF, which implements the metamodel
- the XSD constructs that correspond to CMF classes, datatypes, and properties, and which also implement the metamodel

In addition to the UML diagram, this section contains several tables that document the classes, attributes, and relationships in the metamodel. These tables have the following columns:

Column	Definition
Name	the name of the class, attribute, or relationship
Definition	the definition of the object or property
Card	the number of times this property may/must appear in an object
Ord	true when the order of the instances of a repeatable property in an object is significant
Range	the class or datatype of a property

Classes, attributes, and relationships have the same names in the metamodel and in CMF. (Attributes and relationship names have lower camel case in the diagram and tables, following the UML convention. The tables and the CMF specification use the same names in upper camel case, following the NIEM convention.)

The definitions in these tables follow NIEM rules for documentation (which are described in section 8). As a result, the definition of each metamodel class begins with "A data type for..." instead of "A class for...". (For historical reasons, the name of every class and datatype in the NIEM model ends in "Type", and this is reflected in the conventions for documentation. See section 3.4.)

Names from CMF and the metamodel do not appear in the XSD representation of a model. Instead, NIEM defines special interpretations of XML Schema components, making the elements and attributes in an XSD ·schema document· equivalent to CMF model components. The mapping between CMF components and XSD schema components is provided by a table in each section below, with these columns:

Column	Definition
CMF	CMF component name
XSD	XSD equivalent

4.1 Model

A Model object represents a complete or partial NIEM model. (A complete model has a definition for every referenced component. In an incomplete model, some components have URI references to components defined in another model.)

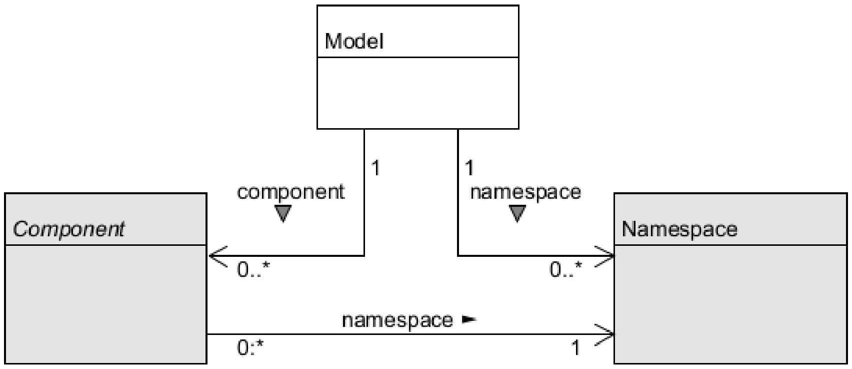


Figure 4-2: Model class diagram

Name	Definition	Card	Ord	Range
Model	A data type for a NIEM data model.			
component	A data concept for a component of a NIEM data model.	0..*	-	ComponentType
namespace	A namespace of a data model component	0..*	-	NamespaceType

In XSD, an instance of the Model class is represented by a ·schema document set·.

4.2 Namespace

A Namespace object represents a namespace in a model. For example, the namespace with the URI `https://docs.oasis-open.org/niemopen/ns/model/niem-core/6.0/` is a namespace in the NIEM 6.0 model.

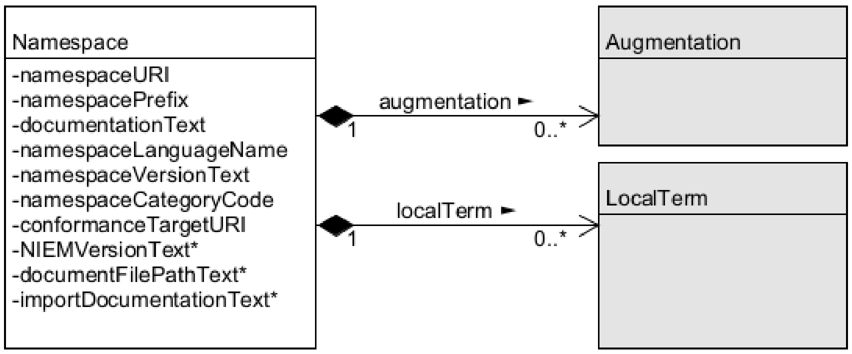


Figure 4-3: Namespace class diagram

Name	Definition	Card	Ord	Range
Namespace	A data type for a namespace.			
NamespaceURI	A URI for a namespace.	1	-	xs:anyURI
NamespacePrefixText	A namespace prefix name for a namespace.	1	-	xs:NCName
DocumentationText	A human-readable text documentation of a namespace.	1..*	Y	TextType
NamespaceLanguageName	A name of a default language of the terms and documentation text in a namespace.	1	-	xs:language
NamespaceVersionText	A version of a namespace; for example, used to distinguish a namespace subset, bug fix, documentation change, etc.	1	-	xs:token
NamespaceCategoryCode	A kind of namespace in a NIEM model (external, core, domain, etc.).	1	-	NamespaceCategoryCodeType
ConformanceTargetURI	A ·conformance target identifier·.	0..*	-	xs:anyURI
NIEMVersionText	A NIEM version number of the builtin schema components used in a namespace; e.g. "5" or "6".	0..1	-	xs:token
DocumentFilePathText	A relative file path from the top schema directory to a schema document for this namespace.	0..1	-	xs:string
ImportDocumentationText	Human-readable documentation from the first `xs:import` element importing this namespace.	0..1	-	xs:string
AugmentationRecord	An augmentation of a class with a property by a namespace.	0..*	-	AugmentationType
LocalTerm	A data type for the meaning of a term that may appear within the name of a model component.	0..*	-	LocalTermType

In XSD, an instance of the Namespace class is represented by the `<xs:schema>` element in a schema document. Figure 4-4 shows the representation of a Namespace object in CMF and in the corresponding XSD.

```
<Namespace>
  <NamespaceURI>https://docs.oasis-open.org/niemopen/ns/model/niem-core/6.0/</NamespaceURI>
  <NamespacePrefixText>nc</NamespacePrefixText>
  <DocumentationText>NIEM Core.</DocumentationText>
  <ConformanceTargetURI>
    https://docs.oasis-open.org/niemopen/ns/specification/XNDR/6.0/#ReferenceSchemaDocument
  </ConformanceTargetURI>
  <NamespaceVersionText>ps02</NamespaceVersionText>
  <NamespaceLanguageName>en-US</NamespaceLanguageName>
</Namespace>
---------------
<xs:schema
  targetNamespace="https://docs.oasis-open.org/niemopen/ns/model/niem-core/6.0/"
  xmlns:ct="https://docs.oasis-open.org/niemopen/ns/specification/conformanceTargets/6.0/"
  xmlns:nc="https://docs.oasis-open.org/niemopen/ns/model/niem-core/6.0/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  ct:conformanceTargets="https://docs.oasis-open.org/niemopen/ns/specification/XNDR/6.0/#ReferenceSchemaDocument"
```

```
   version="ps02"
   xml:lang="en-US">
   <xs:annotation>
     <xs:documentation>NIEM Core.</xs:documentation>
   </xs:annotation>
 </xs:schema>
```

Figure 4-4: Namespace object in CMF and XSD

The following table shows the mapping between Namespace object representations in CMF and XSD.

CMF	XSD
NamespaceURI	`xs:schema/@targetNamespace`
NamespacePrefixText	The prefix in the first namespace declaration of the target namespace
DocumentationText	`xs:schema/xs:annotation/xs:documentation`
ConformanceTargetURI	Each of the URIs in the list attribute `xs:schema/@ct:conformanceTargets`
NamespaceVersionText	`xs:schema/@version`
NamespaceLanguageName	`xs:schema/@xml:lang`

4.3 Component

A Component is either a Class object, a Property object, or a Datatype object in a NIEM model. This abstract class defines the common properties of those three concrete subclasses.

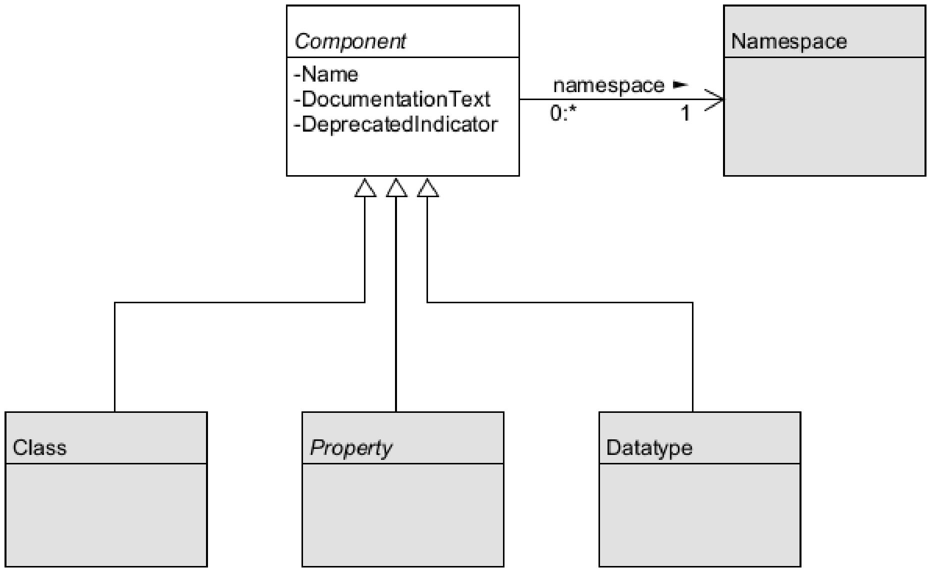


Figure 4-5: Component class diagram

Name	Definition	Card	Ord	Range
Component	A data type for common properties of a data model component in NIEM.			
Name	The name of a data model component.	1	-	xs:NCName
DocumentationText	A human-readable text definition of a data model component.	0..*	Y	TextType
DeprecatedIndicator	True for a deprecated schema component; that is, a component that is provided, but the use of which is not recommended.	0..1	-	xs:boolean

Name	Definition	Card	Ord	Range
Namespace	The namespace of a data model component.	1	-	NamespaceType

In XSD, the common properties of a Component object are represented by a complex type definition or an element or attribute declaration. Figure 4-6 shows the representation of those common properties in CMF and XSD.

```
<DataProperty>
  <Name>ActivityCompletedIndicator</Name>
  <Namespace structures:ref="nc"/>
  <DocumentationText>True if an activity has ended; false otherwise.</DocumentationText>
  <DeprecatedIndicator>false</DeprecatedIndicator>
---------------
<xs:element name="ActivityCompletedIndicator" type="niem-xs:boolean" appinfo:deprecated="false">
  <xs:annotation>
    <xs:documentation>True if an activity has ended; false otherwise.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Figure 4-6: Component object (abstract) in CMF and XSD

The following table shows the mapping between Component properties in CMF and XSD.

CMF	XSD
Name	`@name` of element or attribute declaration
NamespaceURI	`@targetNamespace` of schema document
DocumentationText	`xs:annotation/xs:documentation` of element or attribute declaration
DeprecatedIndicator	'@appinfo:deprecated` of element or attribute declaration

4.4 Class

A Class object represents a class of message objects defined by a NIEM model; that is, an ·object class·. For example, `nc:ItemType` is a Class object in the NIEM Core model.

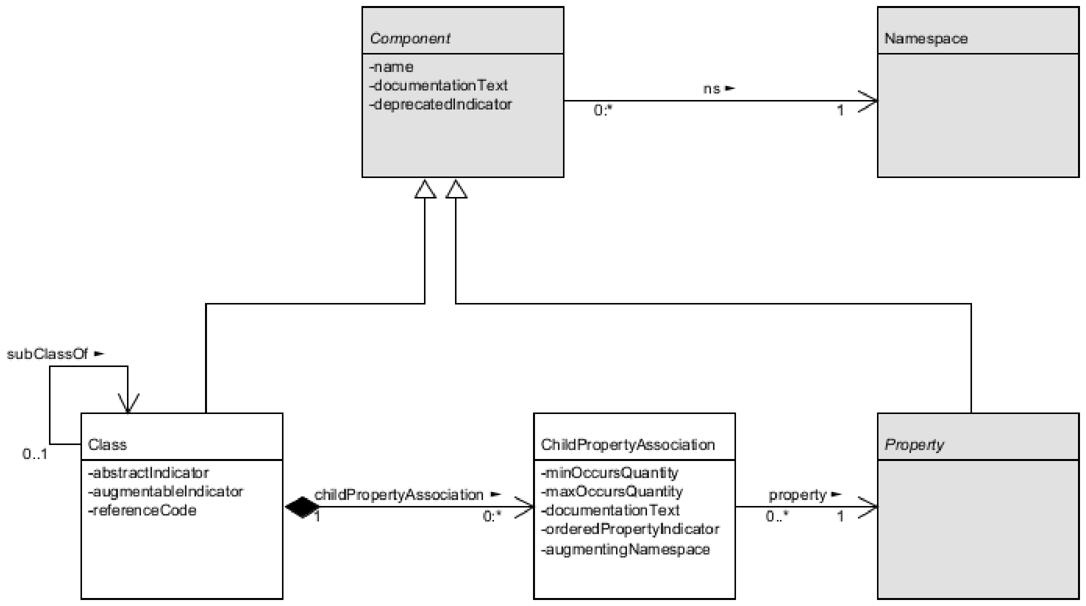


Figure 4-7: Class and PropertyAssociation class diagram

Name	Definition	Card	Ord	Range
Class	A data type for a class.			
AbstractIndicator	True if a class is a base for extension, and must be specialized to be used directly; false if a class may be used directly.	0..1	-	xs:boolean
AugmentableIndicator	True for an ·augmentable class·; that is, if a class can be augmented with additional properties.	0..1	-	xs:boolean
ReferenceCode	A code describing how a property may be referenced (or must appear inline).	0..1	-	ReferenceCodeType
subClassOf	A base class of a subclass.	0..1	-	ClassType
ChildPropertyAssociation	An association between a class and a child property of that class.	0..*	Y	ChildPropertyAssociationType

The range of the `ReferenceCode` property is a code list with the following codes and meanings:

Code	Definition
REF	A code for a property that may be referenced by an IDREF (in XML) or NCName (in JSON).
URI	A code for a property that may be referenced by a URI.
ANY	A code for a property that may be reference by IDREF/NCName or URI.
NONE	A code for a property that my not be referenced and must appear inline.

There are three special categories of ·object class·:

- An ·adapter class· contains only properties from a single ·external namespace·. It acts as a conformance wrapper around data components defined in standards that are not NIEM conforming. An ·adapter class· has a name ending in "AdapterType". (See section 11.4.)
- An ·association class· represents a specific relationship between objects. Associations are used when a simple NIEM property is insufficient to model the relationship clearly, or to model properties of the relationship itself. An ·association class· has a name ending in "AssociationType".
- An ·atomic class· contains no object properties, at least one ·attribute property·, and exactly one data property that is not an attribute property. An ·atomic class· does not have a special name.

4.4.1 Ordinary class

The instances of most classes (including adapter and association classes) are represented in XML as an element with complex content; that is, with child elements, and sometimes with attributes. For example, figure 4-8 shows an XML element with complex content, and also the equivalent JSON message.

```
<ex:ItemWeightMeasure>                                      | {
  <ex:MassUnitCode>KGM</unece:MassUnitCode>                 |   "ex:ItemWeightMeasure": {
  <ex:MeasureDecimalValue>22.5</ex:MeasureDecimalValue>     |     "ex:MassUnitCode": "KGM",
</ex:ItemWeightMeasure>                                     |     "ex:MeasureDecimalValue": 22.5
                                                            |   }
                                                            | }
```

Figure 4-8: Instance of an ordinary class in XML and JSON

These ordinary classes are represented in XSD as a complex type with complex content ("CCC type"); that is, a type with child elements. Figure 4-9 below shows a ordinary Class object defining the class of the `ItemWeightMeasure` property in the example above, represented first in CMF, and then in XSD as a complex type with child elements.

```
<Class structures:id="ex.WeightMeasureType">
  <Name>WeightMeasureType</Name>
  <Namespace structures:ref="ex" xsi:nil="true"/>
  <PropertyAssociation>
    <DataProperty structures:ref="ex.MassUnitCode" xsi:nil="true"/>
    <MinOccursQuantity>1</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
  </PropertyAssociation}>
  <PropertyAssociation>
    <DataProperty structures:ref="ex.MeasureDecimalValue" xsi:nil="true"/>
    <MinOccursQuantity>1</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
  </PropertyAssociation>
</Class>
---------------
<xs:complexType name="WeightMeasureType">
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      <xs:sequence>
        <xs:element ref="ex:MassUnitCode"/>
        <xs:element ref="ex:MeasureDecimalValue"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 4-9: An ordinary Class object in CMF and XSD (CCC type)

The following table shows the mapping between Class object representations in CMF and XSD.

CMF	XSD
AbstractIndicator	`xs:complexType/@abstract`
SubClassOf	`xs:complexType/xs:complexContent/xs:extension/@base`
AugmentableIndicator	True if the last element in the sequence is an *augmentation point*.
ReferenceCode	`xs:complexType/@appinfo:referenceCode`
PropertyAssociation	`xs:complexType/xs:complexContent/xs:extension/xs:sequence/xs:element` or `xs:complexType/xs:complexContent/xs:extension/xs:attribute`

4.4.2 Atomic class

Instances of an ·atomic class· are represented as an element with simple content and attributes in XML. Figure 4-10 below shows an XML and JSON instance of an atomic class.

```
<ex:ItemWeightMeasure ex:massUnitCode="KGM">     | {
  22.5                                           |   "ex:ItemWeightMeasure": {
</ex:ItemWeightMeasure>                          |     "ex:massUnitCode": "KGM",
                                                 |     "ex:WeightMeasureLiteral": 22.5
                                                 |   }
                                                 | }
```

Figure 4-10: Instance of an atomic class in XML and JSON

An atomic class is represented in XSD as a complex type with simple content ("CSC type") and attributes. This is illustrated in figure 4-11 below, which shows an ·atomic class· defining the class of the `ItemWeightMeasure` property in figure 4-10 above.

```
  <Class structures:id="ex.WeightMeasureType">
    <Name>WeightMeasureType</Name>
    <Namespace structures:ref="ex" xsi:nil="true"/>
      <PropertyAssociation>
        <DataProperty structures:ref="ex.massUnitCode" xsi:nil="true"/>
        <MinOccursQuantity>1</MinOccursQuantity>
        <MaxOccursQuantity>1</MaxOccursQuantity>
      </PropertyAssociation>
      <PropertyAssociation>
        <DataProperty structures:ref="ex.WeightMeasureLiteral" xsi:nil="true"/>
        <MinOccursQuantity>1</MinOccursQuantity>
        <MaxOccursQuantity>1</MaxOccursQuantity>
      </PropertyAssociation>
    </Class>
  ---------------
  <xs:complexType name="WeightMeasureType">
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute ref="ex:massUnitCode" use="required"/>
        <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
```

Figure 4-11: An atomic class object in CMF and XSD (CSC type)

An ·atomic class· always has one DataProperty that is not an ·attribute property·. This property is named after the class, with "Type" replaced by "Literal" It does not appear in the XSD representation of the atomic class, or as a separate element in the XML message.

An ·atomic class· always has at least one [attribute property(#def). In XSD, a complex type with simple content and no attributes represents a Datatype, not a Class.

4.5 ChildPropertyAssociation

An instance of the ChildPropertyAssociation class represents an association between a class and a child property of that class. For example, `nc:PersonMiddleName` property and `nc:personNameCommentText` are two child properties of the 'nc:PersonType` class.

Name	Definition	Card	Ord	Range
PropertyAssociation	A data type for an occurrence of a property as content of a class.			
MinOccursQuantity	The minimum number of times a property may occur within an object of a class.	1	-	xs:integer
MaxOccursQuantity	The maximum number of times a property may occur within an object of a class.	1	-	MaxOccursType
DocumentationText	A human-readable documentation of the association between a class and a child property content of that class.	0..*	Y	TextType
OrderedPropertyIndicator	True if the order of a repeated property within an object is significant.	0..1	-	xs:boolean
AugmentingNamespace	A namespace responsible for augmenting a class with a property.	0..*	-	NamespaceType
Property	The property that occurs in the class.	1	-	PropertyType

A PropertyAssociation object is represented in XSD as an element or attribute reference within a complex type definition. Figure 4-12 shows the representation of two PropertyAssociation objects, first in CMF, and then in XSD.

```
<PropertyAssociation>
  <ObjectProperty structures:ref="nc.PersonMiddleName" xsi:nil="true"/>
  <MinOccursQuantity>0</MinOccursQuantity>
  <MaxOccursQuantity>unbounded</MaxOccursQuantity>
  <DocumentationText>
    Documentation here is unusual; it refers to the association between the object and this property.
  </DocumentationText>
  <OrderedPropertyIndicator>true</OrderedPropertyIndicator>
</PropertyAssociation>
<PropertyAssociation>
  <DataProperty structures:ref="nc:personNameCommentText" xsi:nil="true"/>
  <MinOccursQuantity>0</MinOccursQuantity>
  <MaxOccursQuantity>1</MaxOccursQuantity>
</PropertyAssociation>
---------------
<xs:sequence>
  <xs:element ref="nc:PersonMiddleName"
    minOccurs="0" maxOccurs="unbounded" appinfo:orderedPropertyIndicator="true">
    <xs:annotation>
      <xs:documentation>
        Documentation here is unusual; it refers to the relationship between the object and this property.
      </xs:documentation>
    </xs:annotation>
  </xs:element>
</xs:sequence>
<xs:attribute ref="nc:personNameCommentText" use="optional"/>
```

Figure 4-12: PropertyAssociation object in CMF and XSD

The following table shows the mapping between PropertyAssociation representations in CMF and XSD.

CMF	XSD
Property	The property object for `xs:element/@ref` or `xs:attribute/@ref`.
MinOccursQuantity	`xs:element/@minOccurs` or `xs:attribute/@use`
MaxOccursQuantity	`xs:element/@maxOccurs`
DocumentationText	`xs:element/xs:annotation/xs:documentation` or `xs:attribute/xs:annotation/xs:documentation`
OrderedPropertyIndicator	`xs:element/@appinfo:orderedPropertyIndicator`
AugmentingNamespace	`xs:element/@appinfo:augmentingNamespace` or `xs:attribute/@appinfo:augmentingNamespace`

4.6 Property

A Property object is either an ObjectProperty or a DataProperty in a NIEM model. This abstract class defines the common properties of those two concrete subclasses.

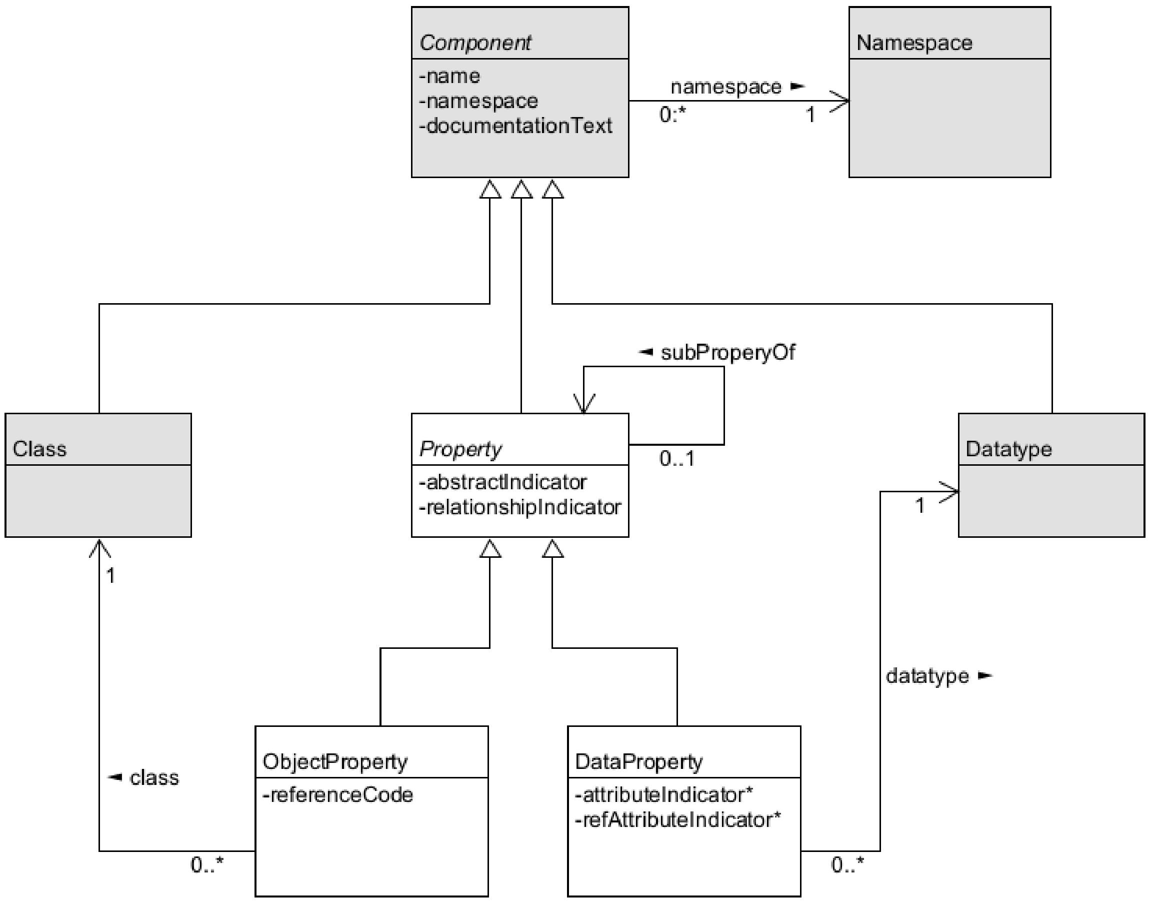


Figure 4-13: Property class diagram

Name	Definition	Card	Ord	Range
Property	A data type for a property.			
AbstractIndicator	True if a property must be specialized; false if a property may be used directly.	0..1	-	xs:boolean
RelationshipIndicator	True for a property that applies to the relationship between two objects (instead of to a single object).	0..1	-	xs:boolean
SubPropertyOf	A property of which a property is a subproperty.	0..1	-	PropertyType

The meaning of a relationship property is explained in Section 5: Data modeling patterns. TODO

The examples of a Property object in CMF and XSD, and the table showing the mapping between the CMF and XSD representations, are shown below in the definitions of the concrete subclasses, ObjectProperty and DataProperty.

4.7 ObjectProperty

An instance of the ObjectProperty class represents a property in a NIEM model with a range that is a class. For example, the `nc:PersonMiddleName` object in the NIEM core model is an object property with a range of the `nc:PersonNameTextType` class.

Name	Definition	Card	Ord	Range
ObjectProperty	A data type for an object property.			
ReferenceCode	A code describing how a property may be referenced (or must appear inline).	0..1	-	ReferenceCodeType

Name	Definition	Card	Ord	Range
Class	The class of this object property.	1	-	ClassType

An ObjectProperty object is represented in XSD as an element declaration with a type that is a Class object. Figure 4-14 shows an ObjectProperty object, represented first in CMF, and then in XSD.

```
<ObjectProperty structures:id="ex.ExampleObjectProperty">
  <Name>ExampleObjectProperty</Name>
  <Namespace structures:ref="ex" xsi:nil="true"/>
  <DocumentationText>Documentation text for ExampleObjectProperty.</DocumentationText>
  <DeprecatedIndicator>false</DeprecatedIndicator>
  <AbstractIndicator>true</AbstractIndicator>
  <ReferenceCode>URI</ReferenceCode>
  <Class structures:ref="ex.ExType" xsi:nil="true"/>
</ObjectProperty>
---------------
<xs:element name="ExampleObjectProperty" type="ex:ExType" abstract="true" appinfo:referenceCode="URI">
  <xs:annotation>
    <xs:documentation>Documentation text for ExampleObjectProperty.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Figure 4-14: ObjectProperty object in CMF and XSD

The following table shows the mapping between ObjectProperty object representations in CMF and XSD.

CMF	XSD
Namespace	The namespace object for the containing schema document.
Name	`xs:complexType/@name`
DocumentationText	`xs:complexType/xs:annotation/xs:documentation`
DeprecatedIndicator	`xs:complexType/@appinfo:deprecated`
AbstractIndicator	`xs:complexType/@abstract`
SubPropertyOf	The property object for `xs:element/@substitutionGroup`
RelationshipPropertyIndicator	`xs:element/@appinfo:relationshipPropertyIndicator`
Class	The class object for `xs:element/@type`
ReferenceCode	`xs:complexType/@appinfo:referenceCode`

4.8 DataProperty

An instance of the DataProperty class represents a property in a NIEM model with a range that is a datatype. For example, the `nc:personNameCommentText` property in the NIEM core model is a data property with a range of the `xs:string` datatype.

Name	Definition	Card	Ord	Range
DataProperty	A data type for a data property.			
AttributeIndicator	True for a property that is represented as attributes in XML.	0..1	-	xs:boolean
RefAttributeIndicator	True for a property that is an object reference attribute.	0..1	-	xs:boolean
Datatype	The datatype of this data property.	1	-	DatatypeType

An ·attribute property· is a data property in which `AttributeIndicator` is true. These are represented in XSD as an attribute declaration.

A ·reference property· is an attribute property that contains a reference to an object in a ·message·. (Object references are described in section ref message rules.)TODO

A DataProperty object is represented in XSD as an attribute declaration, or as an element declaration with a type that is a Datatype object. Figure 4-15 shows the representations of two DataProperty objects, first in CMF, and then in the corresponding XSD.

```
<DataProperty structures:id="ex.ExampleDataProperty">
  <Name>ExampleDataProperty</Name>
  <Namespace structures:ref="ex" xsi:nil="true"/>
  <DocumentationText>Documentation text for ExampleDataProperty.</DocumentationText>
  <DeprecatedIndicator>true</DeprecatedIndicator>
  <AbstractIndicator>true</AbstractIndicator>
  <SubPropertyOf structures:ref="ex.PropertyAbstract" xsi:nil="true"/>
  <Datatype structures:ref="ex.ExType" xsi:nil="true"/>
</DataProperty>
<DataProperty structures:id="ex.exampleAttributeProperty">
  <Name>exampleAttributeProperty</Name>
  <Namespace structures:ref="ex" xsi:nil="true"/>
  <DocumentationText>Documentation text for AttributeProperty.</DocumentationText>
  <DeprecatedIndicator>true</DeprecatedIndicator>
  <Datatype structures:ref="xs.string" xsi:nil="true"/>
  <AttributeIndicator>true</AttributeIndicator>
  <RefAttributeIndicator>true</RefAttributeIndicator>
</DataProperty>
---------------
<xs:element name="ExampleDataProperty" type="ex:ExType" substitutionGroup="ex:PropertyAbstract"   appinfo:deprecated="true">
  <xs:annotation>
    <xs:documentation>Documentation text for ExampleDataProperty.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:attribute name="exampleAttributeProperty" type="xs:string" appinfo:referenceAttributeIndicator="true">
  <xs:annotation>
    <xs:documentation>Documentation text for ExampleDataProperty.</xs:documentation>
  </xs:annotation>
</xs:attribute>
```

Figure 4-15: DataProperty object in CMF and XSD

The following table shows the mapping between DataProperty representations in CMF and XSD.

CMF	XSD
Namespace	The namespace object for the containing schema document.
Name	`xs:complexType/@name`
DocumentationText	`xs:complexType/xs:annotation/xs:documentation`
DeprecatedIndicator	`xs:complexType/@appinfo:deprecated`
AbstractIndicator	`xs:complexType/@abstract`
SubPropertyOf	The property object for `xs:element/@substitutionGroup`
RelationshipPropertyIndicator	`xs:element/@appinfo:relationshipPropertyIndicator`
Datatype	The datatype object for `xs:element/@type`
AttributeIndicator	True for an attribute declaration.
RefAttributeIndicator	`xs:attribute/@appinfo:referenceAttributeIndicator`

4.9 Datatype

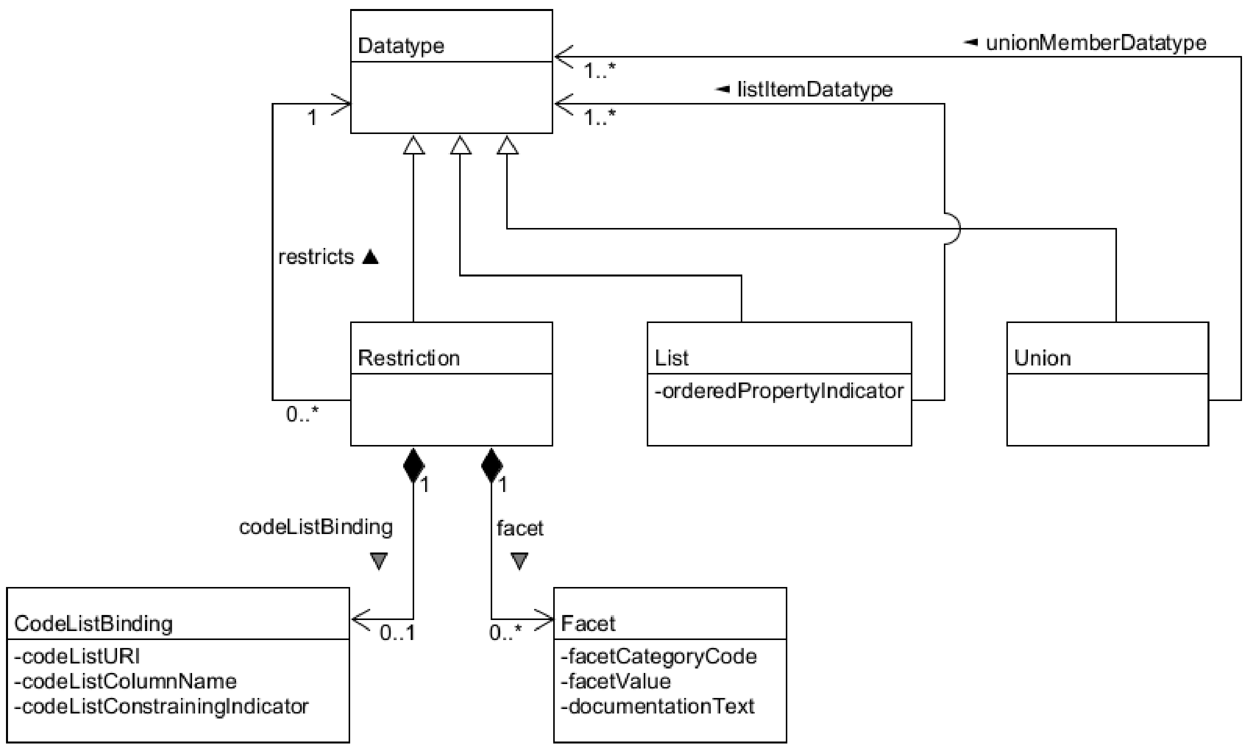


Figure 4-16: Datatype classes

An instance of the Datatype class defines the allowed values of a data property in a ·message·. Objects for primitive data types, corresponding to the XSD data types, have only the *name*, *namespace*, and *documentation* properties inherited from the Component class. For example, figure 4-17 shows the CMF representation of the `xs:string` primitive data type. All other datatypes are represented by either a Restriction, List, or Union object.

```
<Datatype>
  <Name>string</Name>
  <Namespace structures:ref="xs" xsi:nil="true"/>
</Datatype>
```

Figure 4-17: Plain CMF datatype object for `xs:string`

4.10 List

An instance of the List class represents a NIEM model datatype with values that are a whitespace-separated list of atomic values.

Name	Definition	Card	Ord	Range
List	A data type for a NIEM model datatype that is a whitespace-separated list of atomic values.			
OrderedPropertyIndicator	True if the order of a repeated property within an object is significant.	0..1	-	xs:boolean
ListItemDatatype	The datatype of the atomic values in a list.	1	-	DatatypeType

A List object is represented in XSD as a complex type definition that extends a simple type definition that has an `xs:list` element. Figure 4-18 shows the CMF and XSD representation of a List object.

```
<List structures:id="ex.ExListType">
  <Name>ExListType</Name>
  <Namespace structures:ref="ex" xsi:nil="true"/>
  <DocumentationText>A data type for a list ofintegers.</DocumentationText>
  <ListItemDatatype structures:ref="xs.integer" xsi:nil="true"/>
  <OrderedPropertyIndicator>true</OrderedPropertyIndicator>
</List>
---------------
<xs:simpleType name="ExListSimpleType">
  <xs:list itemType="xs:integer"/>
</xs:simpleType>
<xs:complexType name="ExListType" appinfo:orderedPropertyIndicator="true">
  <xs:annotation>
    <xs:documentation>A data type for a list of integers.</xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="ex:ExListSimpleType">
      <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Figure 4-18: List object in CMF and XSD

The following table shows the mapping between List object representations in CMF and XSD.

CMF	XSD
Namespace	The namespace object for the containing schema document.
Name	`xs:complexType/@name`
DocumentationText	`xs:complexType/xs:annotation/xs:documentation`
DeprecatedIndicator	`xs:complexType/@appinfo:deprecated`
ListItemDatatype	`xs:simpleType/xs:list/@itemType`
OrderedPropertyIndicator	`xs:complexType/@appinfo:orderedPropertyIndicator`

4.11 Union

An instance of the Union class represents a NIEM model datatype that is the union of one or more datatypes.

Name	Definition	Card	Ord	Range
Union	A data type for a NIEM model datatype that is a union of datatypes.			
UnionMemberDatatype	A NIEM model datatype that is a member of a union datatype.	1..*	-	DatatypeType

A Union object is represented in XSD as a complex type definition that extends a simple type definition that has an `xs:union` element. Figure 4-19 shows the XSD and CMF representations of a Union object.

```
<Union structures:id="ex.UnionType">
  <Name>UnionType</Name>
  <Namespace structures:ref="test" xsi:nil="true"/>
  <DocumentationText>A data type for a union of integer and float datatypes.</DocumentationText>
  <UnionMemberDatatype structures:ref="xs.integer" xsi:nil="true"/>
  <UnionMemberDatatype structures:ref="xs.float" xsi:nil="true"/>
</Union>
---------------
<xs:simpleType name="UnionSimpleType">
```

```
  <xs:union memberTypes="xs:integer xs:float"/>
</xs:simpleType>
<xs:complexType name="UnionType">
  <xs:annotation>
    <xs:documentation>A data type for a union of integer and float datatypes.</xs:documentation>
  </xs:annotation>  <xs:simpleContent>
    <xs:extension base="ex:UnionSimpleType">
      <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Figure 4-19: Union object in CMF and XSD

The following table shows the mapping between UnionDatatype object representations in CMF and XSD.

CMF	XSD
Namespace	The namespace object for the containing schema document.
Name	`xs:complexType/@name`
DocumentationText	`xs:complexType/xs:annotation/xs:documentation`
DeprecatedIndicator	`xs:complexType/@appinfo:deprecated`
UnionMemberDatatype	`xs:simpleType/xs:union/@memberTypes`

4.12 Restriction

An instance of the Restriction class represents a NIEM model datatype as a base datatype plus zero or more constraining facets.

Name	Definition	Card	Ord	Range
Restriction	A data type for a restriction of a data type.			
RestrictionBase	The NIEM model datatype that is restricted by this datatype.	1	-	DatatypeType
Facet	A constraint on an aspect of a data type.	0..*	-	FacetType
CodeListBinding	A property for connecting atomic values defined by a data type to a a column of a code list.	0..1	-	CodeListBindingType

A Restriction object is represented in XSD as a complex type with simple content that contains an `xs:restriction` element. Figure 4-20 shows the CMF and XSD representations of a Restriction object.

```
<Restriction structures:id="test.RestrictionType">
  <Name>RestrictionType</Name>
  <Namespace structures:ref="test" xsi:nil="true"/>
  <DocumentationText>Exercise code list binding</DocumentationText>
  <RestrictionBase structures:ref="xs.token" xsi:nil="true"/>
  <Facet>
    <FacetCategoryCode>enumeration</FacetCategoryCode>
    <FacetValue>GB</StringValue>
  </Facet>
  <Facet>
    <FacetCategoryCode>enumeration</FacetCategoryCode>
    <FacetValue>US</StringValue>
  </Facet>
  <CodeListBinding>
    <CodeListURI>http://api.nsgreg.nga.mil/geo-political/GENC/2/3-11</CodeListURI>
    <CodeListColumnName>foo</CodeListColumnName>
    <CodeListConstrainingIndicator>true</CodeListConstrainingIndicator>
  </CodeListBinding>
</Restriction>
---------------
<xs:complexType name="RestrictionType">
  <xs:annotation>
    <xs:appinfo>
      <clsa:SimpleCodeListBinding codeListURI="http://api.nsgreg.nga.mil/geo-political/GENC/2/3-11"
        columnName="foo" constrainingIndicator="true"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:simpleContent>
    <xs:restriction base="niem-xs:token">
      <xs:enumeration value="GB"/>
      <xs:enumeration value="US"/>
    </xs:restriction>
  </xs:simpleContent>
</xs:complexType>
```

Figure 4-20: Restriction object in CMF and XSD

The following table shows the mapping between Restriction object representations in CMF and XSD.

CMF	XSD
Namespace	The namespace object for the containing schema document.
Name	`xs:complexType/@name`
DocumentationText	`xs:complexType/xs:annotation/xs:documentation`
DeprecatedIndicator	`xs:complexType/@appinfo:deprecated`
RestrictionBase	The datatype object for `xs:complexType/xs:simpleContent/xs:restriction/@base`
Facet	`xs:complexType/xs:simpleContent/xs:restriction/`*XSD-Facet-element*
CodeListBinding	`xs:complexType/xs:annotation/xs:appinfo/clsa:SimpleCodeListBinding`

A ·code list· is a list of distinct conceptual entities, each represented by a code value that has a known meaning beyond its text representation. These codes may be meaningful text or may be a string of alphanumeric identifiers that represent abbreviations for literals.

A ·code list datatype· is a Restriction in which each value that is valid for the datatype corresponds to a code value in a code list.

Many code datatypes have simple content composed of xs:enumeration values. Code types may also be constructed using the NIEM Code Lists Specification **[Code Lists]**, which supports code lists defined using a variety of methods, including CSV spreadsheets; these are represented by a CodeListBinding object, described below.

4.13 Facet

An instance of the Facet class specifies a constraint on the base datatype of a Restriction object.

Name	Definition	Card	Ord	Range
Facet	A data type for a constraint on an aspect of a data type.			
FacetCategoryCode	A kind of constraint on a restriction datatype.	1	-	FacetCategoryCodeType
FacetValue	A value of a constraint on a restriction datatype.	1	-	xs:string
DocumentationText	A human-readable documentation of a constraint on a restriction datatype.	0..*	Y	TextType

The range of the `FacetCategoryCode` property is a code list. The twelve codes correspond to the twelve constraining facets in XML Schema; that is, the code `length` corresponds to the `xs:length` constraining facet in XSD, and constrains the valid values of the base datatype in the same way as the XSD facet.

A Facet object is represented in XSD as a constraining facet on a simple type. Figure 4-21 shows the representation of two Facet objects, first in CMF, then in XSD:

```
<Facet>
  <FacetCategoryCode>minInclusive</FacetCategoryCode>
  <FacetValue>0</FacetValue>
</Facet>
<Facet>
  <FacetCategoryCode>maxExclusive</FacetCategoryCode>
  <FacetValue>360</FacetValue>
</Facet>
---------------
<xs:restriction base="niem-xs:decimal">
  <xs:minInclusive value="0"/>
  <xs:maxExclusive value="360"/>
</xs:restriction>
```

Figure 4-21: Facet object in CMF and XSD

The following table shows the mapping between Facet representations in CMF and XSD:

CMF	XSD
FacetCategoryCode	*the local name of the facet element; e.g.*`minInclusive`
FacetValue	`@value`
DocumentationText	`xs:annotation/xs:documentation`

4.14 CodeListBinding

An instance of the CodeListBinding class establishes a relationship between a Restriction object and a code list specification. The detailed meaning of the object properties is provided in ref code list specification.

Name	Definition	Card	Ord	Range
CodeListBinding	A data type for connecting simple content defined by an XML Schema component to a a column of a code list.			
CodeListURI	A universal identifier for a code list.	1	-	xs:anyURI
CodeListColumnName	A local name for a code list column within a code list.	0..1	-	xs:string
CodeListConstrainingIndicator	True when a code list binding constrains the validity of a code list value, false otherwise.	0..1	-	xs:boolean

A CodeListBinding object is represented in XSD as a `clsa:SimpleCodeListBinding` element in an `xs:appinfo` element. Figure 4-22 shows the representation of a CodeListBinding object, first in CMF, then in XSD.

```
<CodeListBinding>
  <CodeListURI>http://api.nsgreg.nga.mil/geo-political/GENC/2/3-11</CodeListURI>
  <CodeListConstrainingIndicator>false</CodeListConstrainingIndicator>
</CodeListBinding>
---------------
<xs:simpleType name="CountryAlpha2CodeSimpleType">
  <xs:annotation>
    <xs:documentation>A data type for country codes.</xs:documentation>
    <xs:appinfo>
      <clsa:SimpleCodeListBinding codeListURI="http://api.nsgreg.nga.mil/geo-political/GENC/2/3-11"
```

```
constrainingIndicator="false"/>
    </xs:appinfo>
```

Figure 4-22: CodeListBinding object in CMF and XSD

The following table shows the mapping between CodeListBinding representations in CMF and XSD.

CMF	XSD
CodeListURI	`clsa:SimpleCodeListBinding/@codeListURI`
CodeListColumnName	`clsa:SimpleCodeListBinding/@columnName`
CodeListConstrainingIndicator	`clsa:SimpleCodeListBinding/@constrainingIndicator`

4.15 Augmentation class

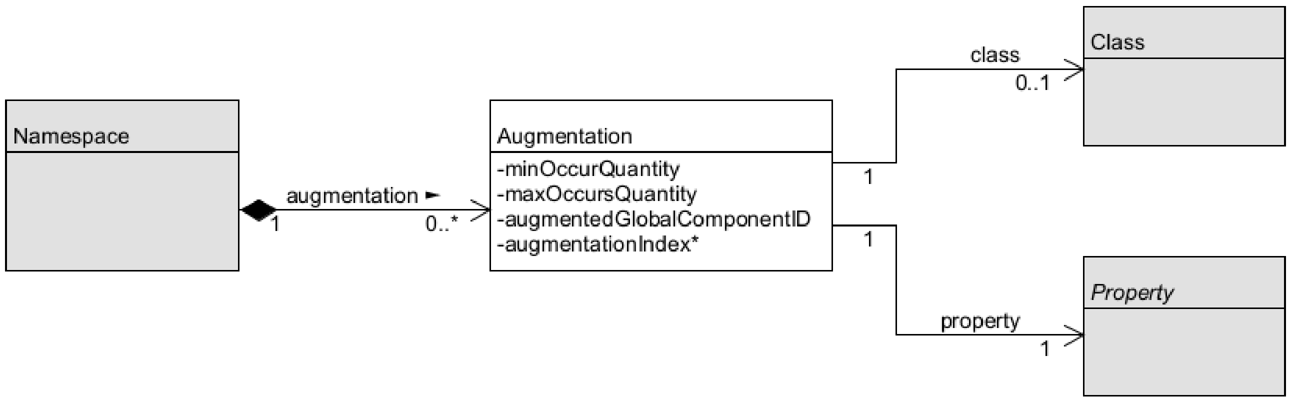


Figure 4-23: Augmentation class diagram

Developers of domain schemas and other schemas that build on and extend the NIEM release schemas need to be able to define additional characteristics of common types. For example, the *NIEM Justice* domain, which addresses justice and public safety concerns, considers the following elements to be characteristics of a person, as defined by `nc:PersonType`:

- `j:PersonAdultIndicator`
- `j:PersonForeignNationalIndicator`

There are several approaches that could be used by a domain to add elements to a common type. One method is to have each domain create a subclass of nc:PersonType that adds elements and attributes for the needed content. Some of the problems with this approach include:

- It results in numerous, domain-specific specializations of `nc:PersonType`, each with common content and extension-specific content.
- There is no method for a message designer to bring these types back together into a single type that carries the desired properties. XML Schema does not support multiple inheritance, so there would be no way to join together `nc:PersonType, ` j:PersonType`, and ` im:PersonType`.
- There is no standard or easy way for the developer to express that the various element instances of the various person types represent the same person, or which parts of those instances are required to be populated; does each person restate the name and birth-date, or is that handled by just one instance?

NIEM's alternative to subclassing is *augmentation*. This is the NIEM mechanism allowing the author of one namespace (the *augmenting namespace*) to add a property to a class in another namespace (the *augmented namespace*) -- without making any change to the augmented namespace. For example:

- `https://docs.oasis-open.org/niemopen/ns/model/domains/justice/6.0/` is an augmenting namespace
- `https://docs.oasis-open.org/niemopen/ns/model/niem-core/6.0/` is an augmented namespace
- `j:PersonAdultIndicator` is an *augmentation property*
- `nc:PersonType` is an *augmented class*

The XSD representation of an augmentation is complex, and is explained below. In CMF, an augmentation is represented as an AugmentationRecord object belonging to the augmenting namespace. In this way, each namespace object contains a complete list of all the augmentations it makes.

Name	Definition	Card	Ord	Range
Augmentation	AugmentationRecordType	A data type for a class that is augmented with a property by a namespace.		
minOccurs	MinOccursQuantity	The minimum number of times a property may occur within an object of a class.	1	-
maxOccurs	MaxOccursQuantity	The maximum number of times a property may occur within an object of a class.	1	-
index	AugmentationIndex	The ordinal position of an augmentation property that is part of an augmentation type.	0..1	-
global	AugmentedGlobalComponentID	The identifer for a NIEM version and kind of component that is the target of this global augmentation.	0..1	-
class	Class	The augmented class.	0..1	-
property	Property	The augmentation property .	1	-

For example, augmentation of `nc:PersonType` with `j:PersonAdultIndicator` and `j:PersonForeignNationalIndicator` by the justice namespace results in the following CMF:

```
<Namespace>
  <NamespaceURI>https://docs.oasis-open.org/niemopen/ns/model/domains/justice/6.0/</NamespaceURI>
  <NamespacePrefix>j</NamespacePrefix>
  <AugmentationRecord>
    <Class structures:ref="nc.PersonType" xsi:nil="true"/>
    <Property structures:ref="j.PersonAdultIndicator" xsi:nil="true"/>
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>unbounded</MaxOccursQuantity>
    <AugmentationIndex>0</AugmentationIndex>
  </AugmentationRecord>
  <AugmentationRecord>
    <Class structures:ref="nc.PersonType" xsi:nil="true"/>
    <Property structures:ref="j.PersonForeignNationalIndicator" xsi:nil="true"/>
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>unbounded</MaxOccursQuantity>
    <AugmentationIndex>1</AugmentationIndex>
  </AugmentationRecord>
</Namespace>
```

Figure 4-24: Augmentation object in CMF

In CMF, the augmentation property appears in the augmented Class object -- like any other property, except that the augmenting namespace is recorded. For example, augmentation of `nc:PersonType` with `j:PersonAdultIndicator` and `j:PersonForeignNationalIndicator` by the justice namespace results in the following CMF:

```
<Class>
  <Name>PersonType</Name>
  <Namespace structures:ref="nc" xsi:nil="true"/>
  <!-- documentation and other PropertyAssociation objects omitted -->
  <PropertyAssociation>
    <DataProperty structures:ref="j.PersonAdultIndicator" xsi:nil="true"/>
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>unbounded</MaxOccursQuantity>
    <AugmentingNamespace>
      https://docs.oasis-open.org/niemopen/ns/model/domains/justice/6.0/
    </AugmentingNamespace>
  </PropertyAssociation>
  <PropertyAssociation>
    <DataProperty structures:ref="j.PersonForeignNationalIndicator" xsi:nil="true"/>
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>unbounded</MaxOccursQuantity>
    <AugmentingNamespace>
      https://docs.oasis-open.org/niemopen/ns/model/domains/justice/6.0/
    </AugmentingNamespace>
  </PropertyAssociation>
</Class>
```

Figure 4-25: AugmentingNamespace property in CMF

4.15.1 Augmentations in NIEM XSD

Augmentations are represented in NIEM XSD in four different ways, depending on the augmenting property and the augmented class:

- The augmenting property may be represented in XSD by an attribute or by an element.
- The augmented class may be represented in XSD by:
  - a type with child elements -- that is, a complex type with complex content (abbreviated "CCC type")
  - a type with an atomic value -- that is, a complex type with simple content (abbreviated "CSC type")

The four combinations iare described in the following sections:

	Augmenting Property	Augmenting Property
Augmented Class	*Element*	*Attribute*
Type with child elements (CCC type)	Section 4.15.2 and Section 4.15.3	Section 4.15.4
Type with an atomic value (CSC type)	Section 4.15.5	Section 4.15.6

4.15.2 Augmenting a CCC type with an augmentation element and type

Every CCC type in a reference or extension schema document contains an ·*augmentation point element*·, which provides a place for any augmentation properties.

```
<xs:complexType name="PersonType">
  <xs:annotation>
    <xs:documentation>A data type for a human being.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      <xs:sequence>
        <xs:element ref="nc:PersonBirthDate" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="nc:PersonName" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="nc:PersonAugmentationPoint" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="PersonAugmentationPoint" abstract="true">
  <xs:annotation>
    <xs:documentation>An augmentation point for PersonType.<</xs:documentation>
  </xs:annotation>
</xs:element>
```

Figure 4-26: An augmentation point element

An ·augmentation point element· is an abstract element declaration having the same name as its augmented base type, with the final "Type" replaced with "AugmentationPoint", and in the same namespace.

For example, `nc:PersonAugmentationPoint` is the augmentation point element for the augmentation base type `nc:PersonType`.

Because an augmentation point element is abstract, it cannot appear in messages - it is a placeholder for element substitution only.

The augmentation point element may be constrained in subset or message schema documents, or omitted if not needed in a particular ·message specification·.

One way to augment a CCC type with an element is to define:

- An ·augmentation type· to contain the augmenting properties, and
- An ·augmentation element· having the augmentation type, and substitutable for the augmentation point element in the augmented type.

Figure 4-22 shows the XSD for a namespace augmenting `nc:PersonType` with two properties. (The corresponding CMF is shown in figures 4-19 and 4-20 above.)

```
<xs:complexType name="PersonAugmentationType">
  <xs:complexContent>
    <xs:extension base="structures:AugmentationType">
      <xs:sequence>
        <xs:element ref="j:PersonAdultIndicator" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="j:PersonForeignNationalIndicator" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="PersonAugmentation" type="j:ExampleAugmentationType" substitutionGroup="nc:PersonAugmentationPoint"/>
```

Figure 4-27: Augmenting a CCC type via an augmentation element and type

In an XML message, the augmentation element is only a container for the augmenting properties; it has no meaning of its own. Figure 4-23 shows a NIEM XML message with this augmentation.

```
<nc:Person>
  <nc:PersonFullName>Tommy Atkins</nc:PersonFullName>
  <j:PersonAugmentation>
    <j:PersonAdultIndicator>true</j:PersonAdultIndicator>
    <j:PersonForeignNationalIndicator>true</j:PersonForeignNationalIndicator>
  </j:PersonAugmentation>
</nc:Person>
```

Figure 4-28: An XML message from a CCC type with an augmentation element

There is no need for the augmentation element in a NIEM JSON message. It does not appear in the equivalent JSON message, shown in figure 4-24 below.

```
{
  "nc:Person": {
    "nc:PersonFullName": "Tommy Atkins",
    "j:PersonAdultIndicator": true,
    "j:PersonForeignNationalIndicator": true
  }
}
```

Figure 4-29: A JSON message from a CCC type with an augmentation element

4.15.3 Augmenting a CCC type with an ordinary element

Another way to augment a CCC type with an element is to substitute an ordinary element (that is, one that does not have an augmentation type) for the augmentation point. In this case there is no augmentation element in the XML message. Figure 4-25 shows the XSD from the augmenting namespace (`http://example.com/s4figs/`). It also shows the CMF for this augmentation. Figure 4-26 shows an XML message with this augmentation, and the equivalent JSON message.

```
<xs:element name="PersonFictionalIndicator" type="niem-xs:boolean"
    substitutionGroup="nc:PersonAugmentationPoint">
  <xs:annotation>
    <xs:documentation>True if a person is a character in a work of fiction.</xs:documentation>
  </xs:annotation>
</xs:element>
----------------
<Namespace>
  <NamespaceURI>http://example.com/s4figs/</NamespaceURI>
  <NamespacePrefix>ex</NamespacePrefix>
  <DocumentationText>Example namespace for NDR6</DocumentationText>
  <AugmentationRecord>
    <Class structures:ref="nc.PersonType" xsi:nil="true"/>
    <DataProperty structures:ref="ex.PersonFictionalCharacterIndicator" xsi:nil="true"/>
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
  </AugmentationRecord>
</Namespace>
<Class>
  <Name>PersonType</Name>
  <Namespace structures:ref="nc" xsi:nil="true"/>
  <!-- documentation and other PropertyAssociation objects omitted -->
  <PropertyAssociation>
    <DataProperty structures:ref="j.PersonFictionalIndicator" xsi:nil="true"/>
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>unbounded</MaxOccursQuantity>
    <AugmentingNamespace>http://example.com/s4figs/</AugmentingNamespace>
  </PropertyAssociation>
</Class>
<DataProperty>
  <Name>PersonFictionalIndicator</Name>
  <Namespace>ex</Namespace>
  <DocumentationText>True if a person is a character in a work of fiction.</DocumentationText>
  <Datatype structures:ref="xs:boolean" xsi:nil="true"/>
</DataProperty>
```

Figure 4-30: Augmenting a CCC type by substituting an ordinary element

```
<nc:Person>
  <nc:PersonFullName>Peter Wimsey</nc:PersonFullName>
  <ex:PersonFictionalIndicator>true</ex:PersonFictionalIndicator>
</nc:Person>
---------------
{
  "nc:Person": {
    "nc:PersonFullName": "Peter Wimsey",
    "ex:PersonFictionalIndicator": true
  }
}
```

Figure 4-31: Equivalent XML and JSON messages from a CCC type augmented by an ordinary element

4.15.4 Augmenting a CCC type with an attribute

A CCC type is augmented with an attribute property by addding an `xs:attribute` element to the complex type definition. This must be done in a subset schema document for the augmented namespace (and not the reference or extension schema document). The `xs:attribute` element must include *appinfo* to specify the augmenting namespace. Figure 4-27 shows the XSD representation of `nc:PersonType` augmented with the attribute `ex:detectiveIndicator`, and the CMF for this augmentation. Figure 4-28 shows an XML message that includes this augmentation, and the equivalent JSON message.

```
<xs:complexType name="PersonType">
  <xs:annotation>
    <xs:documentation>A data type for a human being.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      <xs:sequence>
        <xs:element ref="nc:PersonBirthDate" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="nc:PersonName" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="nc:PersonAugmentationPoint" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="ex:detectiveIndicator
          appinfo:augmentingNamespace="http://example.com/s4figs"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
----------------
<Class>
  <Name>PersonType</Name>
  <Namespace structures:ref="nc" xsi:nil="true"/>
  <!-- documentation and other PropertyAssociation objects omitted -->
  <PropertyAssociation>
    <DataProperty structures:ref="ex:detectiveIndicator" xsi:nil="true"/>
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>unbounded</MaxOccursQuantity>
    <AugmentingNamespace>http://example.com/s4figs/</AugmentingNamespace>
  </PropertyAssociation>
</Class>
```

Figure 4-32: Augmenting a CCC type with an attribute

```
<nc:Person>
  <nc:PersonFullName>Peter Wimsey</nc:PersonFullName>
  <ex:detectiveIndicator>true</ex:detectiveIndicator>
</nc:Person>
---------------
{
  "nc:Person": {
    "nc:PersonFullName": "Peter Wimsey",
    "ex:detectiveIndicator": true
  }
}
```

Figure 4-33: Equivalent XML and JSON messages from a CCC type augmented with an attribute

4.15.5 Augmenting a CSC type with an attribute

A CSC type is also augmented with an attribute property by addding an `xs:attribute` to the complex type definition. This must be done in a subset schema document for the augmented namespace. The attribute reference must include *appinfo* to specify the augmenting namespace. Figure 4-29 shows the XSD representation of `nc:DirectionCodeType` augmented with the attribute `ex:magneticIndicator`, and the equivalent CMF. Figure 4-30 shows part of an XML message that includes this augmentation, and the equivalent JSON message.

```
  <xs:complexType name="DirectionCodeType">
    <xs:annotation>
      <xs:documentation>A data type for compass directions.</xs:documentation>
    </xs:annotation>
    <xs:simpleContent>
      <xs:extension base="nc:DirectionCodeSimpleType">
```

```
        <xs:attribute ref="ex:magneticIndicator"
            appinfo:augmentingNamespace="http://example.com/s4figs/">
        <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
---------------
<Namespace>
  <NamespaceURI>http://example.com/s4figs/</NamespaceURI>
  <NamespacePrefix>ex</NamespacePrefix>
  <DocumentationText>Example namespace for NDR6</DocumentationText>
  <AugmentationRecord>
    <Class structures:ref="nc.DirectionCodeType" xsi:nil="true"/>
    <DataProperty structures:ref="ex.magneticIndicator" xsi:nil="true"/>
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
  </AugmentationRecord>
</Namespace>
<Class>
  <Name>DirectionCodeType</Name>
  <Namespace>nc</Namespace>
  <DocumentationText>A data type for compass directions.</DocumentationText>
  <PropertyAssociation>
    <DataProperty structures:ref="nc:DirectionCodeLiteral" xsi:nil="true"/>
    <MinOccursQuantity>1</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
  </PropertyAssociation>
  <PropertyAssociation>
    <DataProperty structures:ref="ex:magneticIndicator">
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
    <AugmentingNamespace>http://example.com/s4figs/</AugmentingNamespace>
  </PropertyAssociation>
</Class>
```

Figure 4-34: Augmenting a CSC type with an attribute

```
<nc:RelativeLocationDirectionCode ex:magneticIndicator="true">E</nc:RelativeLocationDirectionCode>
---------------
{
  "nc:RelativeLocationDirectionCode": {
    "nc:DirectionCodeLiteral": "E",
    "ex:magneticIndicator": true
  }
}
```

Figure 4-35: Equivalent XML and JSON messages from a CSC type augmented with an attribute

4.15.6 Augmenting a CSC type with an ordinary element

XML simple content cannot contain a child element. A CSC type is augmented with an element property by adding a *reference attribute* to the complex type definition. A reference attribute is a pointer to an element in a message. Figure 4-31 shows the XSD representation of `nc:DirectionCodeType` augmented with the element `nc:Metadata`, and the equivalent CMF.

```
  <xs:complexType name="DirectionCodeType">
    <xs:annotation>
      <xs:documentation>A data type for compass directions.</xs:documentation>
    </xs:annotation>
    <xs:simpleContent>
      <xs:extension base="nc:DirectionCodeSimpleType">
        <xs:attribute ref="nc:metadataRef"
            appinfo:augmentingNamespace="http://example.com/s4figs/">
        <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:attribute name="metadataRef" type="xs:IDREFS">
    <xs:annotation>
      <xs:documentation>
        A list of metadata objects that apply to a node or object represented by an XML element.
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>
---------------
<Namespace>
  <NamespaceURI>http://example.com/s4figs/</NamespaceURI>
  <NamespacePrefix>ex</NamespacePrefix>
  <DocumentationText>Example namespace for NDR6</DocumentationText>
  <AugmentationRecord>
    <Class structures:ref="nc.DirectionCodeType" xsi:nil="true"/>
    <DataProperty structures:ref="nc:metadataRef" xsi:nil="true"/>
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
  </AugmentationRecord>
</Namespace>
<Class>
  <Name>DirectionCodeType</Name>
  <Namespace>nc</Namespace>
  <DocumentationText>A data type for compass directions.</DocumentationText>
  <PropertyAssociation>
    <DataProperty structures:ref="nc:DirectionCodeLiteral" xsi:nil="true"/>
    <MinOccursQuantity>1</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
  </PropertyAssociation>
  <PropertyAssociation>
    <DataProperty structures:ref="nc:metadataRef">
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
    <AugmentingNamespace>http://example.com/s4figs/</AugmentingNamespace>
  </PropertyAssociation>
</Class>
<DataProperty structures:id="nc.metadataRef">
  <Name>privacyAssertionRef</Name>
  <Namespace structures:ref="nc" xsi:nil="true"/>
  <DocumentationText>
    A list of metadata objects that apply to a node or object represented by an XML element.
  </DocumentationText>
  <Datatype structures:ref="xs.IDREFS" xsi:nil="true"/>
  <AttributeIndicator>true</AttributeIndicator>
</DataProperty>
```

Figure 4-36: Augmenting a CSC type with an element

A property with a name ending in "Ref" is a reference attribute. It refers to an element in the same namespace with the same name, first letter in uppercase, and "Ref" deleted; for example, "nc:metadataRef" points to an "nc:Metadata" element. Figure 4-32 shows part of an XML message that includes the above augmentation, and the equivalent JSON message.

```
<nc:RelativeLocationDirectionCode nc:metadataRef="#MD01">E</nc:RelativeLocationDirectionCode>
<nc:Metadata structures:id="#MD01">
  <nc:Comment>Our best guess</nc:Comment>
</nc:Metadata>
---------------
{
  "nc:RelativeLocationDirectionCode": {
    "nc:DirectionCodeLiteral": "E",
    "nc:metadataRef": { "@id": "#MD01" }
  },
  "nc:Metadata": {
    "nc:Comment": "Our best guess"
  }
}
```

Figure 4-37: Equivalent XML and JSON messages from a CSC type augmented with an attribute

4.15.7 Global element augmentations

4.15.8 Global attribute augmentations

4.16 LocalTerm

A *·local term·* is a word, phrase, acronym, or other string of characters that is used in the name of a namespace component, but that is not defined in OED, or that has a non-OED definition in this namespace, or has a word sense that is in some way unclear. An instance of the LocalTerm class captures the namespace author's definition of such a local term. For example, the Justice domain namespace in the NIEM model has a LocalTerm object defining the name "CLP" with documentation "Commercial Learners Permit".

Name	Definition	Card	Ord	Range
LocalTerm	A data type for the meaning of a term that may appear within the name of a model component.			
TermName	The name of the local term.	1	-	xs:token
DocumentationText	A human-readable text definition of a data model component or term, or the documentation of a namespace.	0..1	-	TextType
TermLiteralText	A meaning of a local term provided as a full, plain-text form.	0..1	-	xs:string
SourceURI	A URI that is an identifier or locator for an originating or authoritative document defining a local term.	0..*	-	xs:anyURI
SourceCitationText	A plain text citation of, reference to, or bibliographic entry for an originating or authoritative document defining a local term.	0..*	-	xs:string

A LocalTerm object is represented in XSD by a `appinfo:LocalTerm` element within `xs:appinfo` element in the `xs:schema` element. Figure 4-33 shows the representation of a LocalTerm object in CMF and XSD.

```
<LocalTerm>
  <TermName>2D</TermName>
  <TermLiteralText>Two-dimensional</TermLiteralText>
</LocalTerm>
<LocalTerm>
  <TermName>3D</TermName>
  <DocumentationText>Three-dimensional</DocumentationText>
</LocalTerm>
<LocalTerm>
  <TermName>Test</TermName>
  <DocumentationText>only for test purposes</DocumentationText>
  <SourceURI>http://example.com/1 http://example.com/2</SourceURI>
  <SourceCitationText>citation #1</SourceCitationText>
  <SourceCitationText>citation #2</SourceCitationText>
</LocalTerm>
---------------
<xs:appinfo>
  <appinfo:LocalTerm term="2D" literal="Two-dimensional"/>
  <appinfo:LocalTerm term="3D" definition="Three-dimensional"/>
  <appinfo:LocalTerm term="Test" definition="only for test purposes" sourceURIs="http://example.com/1 http://example.com/2">
    <appinfo:SourceText>citation #1</appinfo:SourceText>
    <appinfo:SourceText>citation #2</appinfo:SourceText>
  </appinfo:LocalTerm>
</xs:appinfo>
```

Figure 4-38: Example LocalTerm objects in CMF and XSD

The following table shows the mapping between LocalTerm object representations in CMF and XSD.

CMF	XSD
TermName	`appinfo:LocalTerm/@term`
DocumentationText	`appinfo:LocalTerm/@definition`
TermLiteralText	`appinfo:LocalTerm/@literal`
SourceURI	Each URI in the `appinfo:LocalTerm/@sourceURIs` list
SourceCitationText	`appinfo:LocalTerm/appinfo:SourceText`

4.17 TextType

An instance of the TextType class combines a string property with a language property.

Name	Definition	Card	Ord	Range
TextType	A data type for a character string with a language code.			
TextLiteral	A literal value that is a character string.	1	-	xs:string
lang	A name of the language of a character string.	0..1	-	xs:language

5. Data modeling patterns

> NOTE: I think the sections on container objects and representation terms belong here, along with any other modeling patterns we want to describe. NDR 5 buries these things in section 10, "Rules for NIEM modeling", but I don't think that makes sense in the NDR 6 organization. This might also be a good place to talk about metadata in NIEM 6.

6. Conformance

This document defines conformance for namespaces, schema documents, models, and messages. These are the conformance targets for NIEM; that is, these are the kinds of artifact for which conformance may be asserted. For each conformance target, this document specifies a set of conformance claims, called rules, which must be fulfilled by a conforming artifact. Rules are normative, and are written with the capitalized **[RFC 2119]** keywords MUST, MUST NOT, etc.

NIEM does not define conformance for applications, systems, databases, or tools. It is therefore impossible for any of these to properly claim "NIEM conformance". However, they *may* properly claim to generate conforming messages or to employ conforming models.

NIEM defines *conformance* with the rules in this document, but it does not define *compliance*. The distinction is based on assessment authority: Anyone may assess conformance with rules. Compliance is assessed by an authority who can compel change or withhold approval. The authoritative assessment in a compliance evaluation is out of scope for NIEMOpen.

The rules in this document are designed to be used with or without the component definitions in the NIEM model. These rules define conformance to the *NIEM architecture*. Conformance to the *NIEM model* is a separate thing, and is not specified by this document.

6.1 Conformance targets

The conformance targets specifed in this document are listed below. The rules for each conformance target appear in the given sections.

- *Namespace:* A ·conforming namespace· is a namespace that fulfils all of the applicable rules in section 7. The rules for this conformance target apply to both the CMF and XSD representations of a namespace. (In CMF, this is a Namespace object in a ·model file·. In XSD, this is a ·schema document·.) The rules for all conforming namespaces are in:

  Section 7.1: Rules for names of namespace components
  Section 7.2: Rules for documentation of namespace components
  Section 7.3: Rules for properties of namespaces

  - *Reference namespace:* Additional rules for the ·reference namespace· conformance target are in *section 7.4*.
  - *Extension namespace:* Additional rules for the ·extension namespace· conformance target are in *section 7.5*.
  - *Subset namespace:* Additional rules for the ·subset namespace· conformance target are in *section 7.6*.

- *Schema document*: A ·conforming schema document· is a ·schema document· that fulfils all applicable rules of this section. The rules for this conformance target apply only to the XSD representation of a namespace. The rules for all conforming schema documents are found in:

  Section 8.1: Rules for the NIEM profile of XSD
  Section 8.2: Rules for type definitions
  Section 8.3: Rules for attribute and element declarations
  Section 8.4: Rules for adapters and external components
  Section 8.5: Rules for proxy types
  Section 8.6: Rules for augmentations
  Section 8.7: Rules for machine-readable annotations

  - *Reference schema document:* Additional rules for the ·reference schema document· conformance target are in *section 8.8*.
  - *Extension schema document:* Additional rules for the ·extension schema document· conformance target are in *section 8.9*.
  - (Additional rules for ·subset schema documents·) are not required.)

- *Model*: A conforming model fulfils all of the rules in section 9. There are two representations for NIEM models, CMF and XSD.

  - *Model file:* A ·model file· is a ·message· that conforms to the CMF ·message type·. Additional rules for this conformance target are in section 9.1.

    - *Schema document set*: A ·conforming schema document set· is a ·schema document set· that fulfils all applicable rules in section 9. Additional rules for this conformance target are in section 9.2.

- *XML message*: Rules applying to a message in XML format.

- *JSON message:* Rules applying to a message in JSON format.

6.2 Conformance target assertions

It is often helpful for an artifact to contain an assertion of the kind of thing it is supposed to be. These assertions can inform both people and tools. The *Conformance Targets Attribute Specification* **[CTAS]** defines an attribute that, when it appears in an XML document, asserts the document conforms to one or more conformance targets. Specifically, this is the ·effective conformance targets attribute·, which is the first occurrence of the attribute `{https://docs.oasis-open.org/niemopen/ns/specification/conformanceTargets/6.0/}conformanceTargets`, in document order.

For XSD, NIEMOpen makes use of **[CTAS]** to indicate whether a ·schema document· is intended to represent a reference, extension, or subset namespace. For example, a ·reference schema document· contains the conformance target assertion shown in figure 5-1 below:

```
<xs:schema
  targetNamespace="https://docs.oasis-open.org/niemopen/ns/model/niem-core/6.0/"
  xmlns:ct="https://docs.oasis-open.org/niemopen/ns/specification/conformanceTargets/6.0/"
  xmlns:nc="https://docs.oasis-open.org/niemopen/ns/model/niem-core/6.0/"
  ct:conformanceTargets="https://docs.oasis-open.org/niemopen/ns/specification/XNDR/6.0/#ReferenceSchemaDocument"
  version="1"
  xml:lang="en-US">
```

Figure 5-1: Conformance target assertion in XSD

In CMF, the `ConformanceTargetURI` property indicates whether a Namespace object represents a reference, extension, or subset namespace. For example, the Namespace object equivalent to the namespace in figure 5-1 is shown below:

```
<Namespace structures:id="nc">
  <NamespaceURI>https://docs.oasis-open.org/niemopen/ns/model/niem-core/6.0/</NamespaceURI>
  <NamespacePrefixText>nc</NamespacePrefixText>
  <DocumentationText>NIEM Core.</DocumentationText>
  <ConformanceTargetURI>
    https://docs.oasis-open.org/niemopen/ns/specification/XNDR/6.0/#ReferenceSchemaDocument
  </ConformanceTargetURI>
  <NIEMVersionText>6</NIEMVersionText>
  <NamespaceVersionText>1</NamespaceVersionText>
  <NamespaceLanguageName>en-US</NamespaceLanguageName>
</Namespace>
```

Figure 5-2: Conformance target in CMF

6.3 Conformance testing

Automated testing of most rules is possible. Some rules require human evaluation.

- Many rules for CMF namespaces and models can be tested by **[CMFTool]**, a free and open-source tool provided by NIEMOpen.

- Many rules for schema documents may be tested by the Schematron rules provided in **[TODO]**.

- Messages must be valid when assessed against the schema of their ·message format·. Many of the rules applicable to all messages are encoded into these schemas when the schemas are generated from the ·message type· by NIEMOpen developer tools; see **[Tools]**.

- The rules in this document that require human evaluation are marked with TODO.

7. Rules for namespaces

These rules apply to both the CMF and XSD representations of conforming namespaces. In CMF, the representation is a Namespace object in a CMF model file. In XSD, it is a schema document.

- Rules for names of components appear in section 7.1
- Rules for documentation of components appear in section 7.2
- Rules for the properties of namespaces appear in section 7.3
- Additional rules applying only to ·reference namespaces· appear in section 7.4
- Additional rules applying only to ·extension namespaces· appear in section 7.5
- Additional rules applying only to ·subset namespaces· appear in section 7.6

7.1 Rules for names of namespace components

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.

These rules apply to all namespaces. In a CMF representation, they apply to the Name property of a Component object. In an XSD representation, they apply to the `{}name` attribute of a complex type definition, element declaration, or attribute declaration.

7.1.1 Rules based on kind of component

Rule 7-1: Class and Datatype components MUST have a name ending in "Type"; Property components MUST NOT.

This rule immediately distinguishes Property components from Class and Datatype components. In an XSD representation, it also avoids naming collisions between type definitions and element/attribute declarations.

Rule 7-2: A component MUST NOT have a name ending in "Augmentation", "AugmentationPoint", or "AugmentationType".

XSD components with these names appear only in the XSD representation of a model. These XSD components are not themselves model components.

7.1.1.1 Rules for names of Class components

Rule 7-3: An ·adapter class· MUST have a name ending in "AdapterType"; all other components MUST NOT.

Rule 7-4: An ·association class· MUST have a name ending in "AssociationType"; all other components MUST NOT.

Rule 7-5: An ·atomic class· with a ·literal property· that has a ·code list datatype· MUST have a name ending in "CodeType"; all other Class components MUST NOT.

These rules immediately distinguish special Class components from ordinary. Rule 7-5 handles an unusual case in XSD. A code list in XSD is represented as a complex type with simple content. This usually corresponds to a Datatype component; however, when that complex type definition includes attribute properties, then it corresponds to a Class component.

7.1.1.2 Rules for names of Datatype components

Rule 7-6: A component with a name ending in "SimpleType" MUST be a Datatype.

A Datatype with a name ending in "SimpleType" is sometimes needed for a ·literal property·, or for a member type in a List or Union component.

Rule 7-7: A Datatype object with a name that ends in "CodeSimpleType" MUST be a ·code list datatype·.

Rule 7-8: A ·code list datatype· MUST have a name ending in "CodeType" or "CodeSimpleType"; all other Datatype components MUST NOT.

The component representing a ·code list· is usually a Datatype. However, when the XSD represenation of a code list includes attributes, it is a Class.

7.1.1.3 Rules for names of Property components

Rule 7-9: A Property object having an AbstractIndicator property with the value `true` SHOULD have a name ending in "Abstract" or "Representation"; all other components SHOULD NOT.

A property name ending in "Abstract" reminds message designers that it cannot be used directly but must be specialized. A property name ending in "Representation" is an instance of the ·representation pattern·, described in section 5.

Rule 7-10: A Property with an ·association class· MUST have a name ending in "Association"; all other components MUST NOT.

Rule 7-11: A Property with a Class or Datatype that represents a ·code list· MUST have a name ending in "Code"; all other components MUST NOT.

Rule 7-12: The ·literal property· of an ·atomic class· MUST have a name ending in "Literal"; all other components MUST NOT.

Component names ending in "Literal" only occur in the CMF representation of an ·atomic class·.

Rule 7-13: A Property that is a ·reference attribute· property MUST have a name ending in "Ref"; all other components MUST NOT.

7.1.2 Rules for composition of component names

Rule 7-14: Except as otherwise provided in this document, the name of a model component MUST be composed of words from the English language, using the prevalent U.S. spelling, as provided by the Oxford English Dictionary **[OED]**.

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.

NIEM supports internationalization in several ways, described in **[I18N]**. NIEM allows (but does not encourage) component names that are not from the English language in extension schema documents.

Rule 7-15: The name of a model component MUST be entirely composed of specified characters.

- Upper-case letters (A–Z)
- Lower-case letters (a–z)
- Digits (0–9)
- Underscore (_)
- Hyphen (-)
- Period (.)

Other characters, such as unicode characters outside the ASCII character set, are explicitly prohibited from the name of an XML Schema component defined by the schema.

Rule 7-16: The name of a model component MUST use the camel case formatting convention.

Camel case is the convention of writing compound words or phrases with no spaces and an initial lowercase or uppercase letter, with each remaining word element beginning with an uppercase letter. *UpperCamelCase* is written with an initial uppercase letter, and *lowerCamelCase* is written with an initial lowercase letter.

Rule 7-17: The name of an attribute property MUST begin with a lowercase character.

Rule 7-18: The name of a model component that is not an attribute property MUST begin with an uppercase character.

Rule 7-19: The characters hyphen (-), underscore (_) MUST NOT appear in a component name unless used as a separator between parts of a word, phrase, or value, which would otherwise be incomprehensible without the use of a separator. The character period (.) MUST NOT appear in a component name unless as a decimal within a numeric value, or unless used as a separator between parts of a word, phrase, or value, which would otherwise be incomprehensible without the use of a separator.

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.

7.1.3 General component naming rules from ISO 11179-5

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 11179-5]** to construct NIEM data component names.

Rule 7-20: 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 7-21: 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 7-22: Articles, conjunctions, and prepositions MUST NOT be used in NIEM component names except where they are required for clarity or by standard convention.

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.

7.1.4 Property nameing rules from ISO 11179-5

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 7-23: Except as specified elsewhere in this document, the name of a property object MUST be formed by the composition of object class qualifier terms, object class term, property qualifier terms, property term, representation qualifier terms, and representation term, as detailed in Annex A of **[ISO 11179-5]**.

For example, the NIEM component name `AircraftFuselageColorCode` is composed of the following:

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

7.1.4.1 Object-class term

Rule 7-24: The object-class term of a NIEM component MUST consist of a term identifying a category of concepts or entities.

NIEM adopts an object-oriented approach to representation of data. Object classes represent what **[ISO 11179-5]** 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.

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. An example of a conept term is Activity. An example of an entity term is Vehicle.

7.1.4.2 Property term

Rule 7-25: A property term MUST describe or represent a characteristic or subpart of an entity or concept.

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. It describes the central meaning of the component.

7.1.4.3 Qualifer terms

Rule 7-26: Multiple qualifier terms MAY be used within a component name as necessary to ensure clarity and uniqueness within its namespace and usage context.

Rule 7-27: The number of qualifier terms SHOULD be limited to the absolute minimum required to make the component name unique and understandable.

Rule 7-28: The order of qualifiers MUST NOT be used to differentiate components.

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.

7.1.4.4 Representation term

The representation terms for a property name serve 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.

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 6-1, Property representation terms, below, 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.

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 continuous or recurring period of time, of a duration greater than or equal to a day.
	Time	A particular point in the progression of time within an unspecified 24-hour day.
	Duration	An amount of time; the length of a time span.
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 **[RFC 3986]**.
Indicator		A list of two mutually exclusive Boolean values that express the only possible states of a property.

Primary Representation Term	Secondary Representation Term	Definition
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 relative speed of change or progress.
	Percent	A representation of a unitless ratio, expressed as parts of a hundred, with 100 percent representing a ratio of 1 to 1.
Quantity		A counted number of non-monetary 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.
List		A sequence of values. This representation term is used in tandem with another of the listed representation terms.
Abstract		An element that may represent a concept, rather than a concrete property. This representation term may be used in tandem with another of the listed representation terms.
Representation		An element that acts as a placeholder for alternative representations of the value of a type (see Section 5, The Representation pattern, above). **TODO**

Table 6-1: Property representation terms

Rule 7-29: If any word in the representation term is redundant with any word in the property term, one occurrence SHOULD be deleted.

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.

Rule 7-30: The name of a data property SHOULD use an appropriate representation term as found in table 6-1, Property representation terms.

Rule 7-31: The name of an object property that corresponds to a concept listed in table 6-1, Property representation terms, SHOULD use a representation term from that table.

Rule 7-32: The name of an object property that does not correspond to a concept listed in table 6-1, Property representation terms SHOULD NOT use a representation term.

7.1.5 Acronyms, abbreviations, and jargon

Rule 7-33: A component name SHOULD use the abbreviations shown in the table below.

Abbreviation	Full Meaning
ID	Identifier
URI	Uniform Resource Identifier

Rule 7-34: A ·local term· MAY be used in the name of a component within its namespace.

A ·local term· is a word, phrase, acronym, or other string of characters that is defined within a namespace by a LocalTerm object.

Rule 7-35: In CMF, a LocalTerm object MUST have a DocumentationText property, or a TermLiteralText property, or both. In XSD, a `LocalTerm` element MUST have a `@definition` attribute, or a `@literal` attribute, or both.

7.2 Rules for documentation of namespace components

NIEM models are composed of data components 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?

·Reference namespaces· and ·extension namespaces· provide the authoritative definition of the components they contain. These definitions include:

1. The structural definition of each component, expressed as CMF objects or XSD schema components. Where possible, meaning is expressed in this way.

2. A text definition of each component, describing what the component means. The term used in this specification for such a text definition is *data definition*.

A ·data definition· is the DocumentText property of a CMF object, or the content of the first occurrence of the element `xs:documentation` that is an immediate child of an occurrence of an element `xs:annotation` that is an immediate child of an XSD schema component.

A ·documented component· is a CMF object or XSD schema component that has an associated data definition.

7.2.1 Rules for documented components

Rule 7-36: In CMF, a Namespace object MUST be a documented component. In XSD, the `xs:schema` element MUST be a documented component.

Rule 7-37: In CMF, a Component object MUST be a documented component. In XSD, a type definition, element declaration, or attribute declaration MUST be a documented component

Rule 7-38: In CMF, a Facet object MUST be a documented component. In XSD, an enumeration facet MUST be a documented component.

Rule 7-39: In CMF, the language name for the first instance of the DocumentationText property in any Namespace or Component object MUST be `en-US`. In XSD, the first occurance of `xs:documentation` within `xs:annotation` MUST be within the scope of an occurance of `xml:lang` with a value of `en-US`. In each case, subsequent instances, if provided, MUST have a different language name.

A model file or schema document always contains data definitions in US English. It may contain equivalent data definitions in other languages.

7.2.2 Rules for data definitions

Rule 7-40: Words or synonyms for the words within a data definition MUST 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 7-41: An object class MUST 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 7-42: An object class MUST NOT be redefined within the definitions of the components that represent properties or subparts of that entity or class.

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 11179-4]** 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 11179-4]** are not violated. This further enhances brevity and precision.

Rule 7-43: A data definition SHOULD NOT contain explicit representational or data typing information such as number of characters, classes of characters, range of mathematical values, etc., unless the very nature of the component can be described only by such information.

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, but the primary source of representational information should come from the XML Schema definition of the types themselves. A developer should try to keep a component's data definition decoupled from its representation.

7.2.3 Data defintion rules from ISO 11179-4

These rules are adopted from **[ISO 11179-4]**, *Information technology — Metadata registries: Formulation of data definitions*

Rule 7-44: Each data definition MUST conform to the requirements for data definitions provided by **[ISO 11179-4]** Section 5.2, *Requirements*; namely, a data definition MUST:

- 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

Rule 7-45: Each data definition SHOULD conform to the recommendations for data definitions provided by **[ISO 11179-4]** Section 5.2, *Recommendations*; namely, 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

7.2.4 Data definition opening phrases

In order to provide a more consistent voice across NIEM, a model built from requirements from many different sources, component data definitions should begin with a standard opening phrase, as defined below.

7.2.4.1 Opening phrases for properties

These rules apply to Property objects in CMF, and to element and attribute declarations in XSD.

Rule 7-46: The data definition for an abstract property SHOULD begin with the standard opening phrase "A data concept...".

Rule 7-47: The data definition for a property that has an association type and is not abstract SHOULD begin with the standard opening phrase "An (optional adjectives) (relationship|association)...".

Rule 7-48: The data definition for a property with a date representation term SHOULD begin with the standard opening phrase "(A|An) (optional adjectives) (date|month|year)...".

Rule 7-49: The data definition for a property with a quantity representation term SHOULD begin with the standard opening phrase "An (optional adjectives) (count|number)...".

Rule 7-50: The data definition for a property with a picture representation term SHOULD begin with the standard opening phrase "An (optional adjectives) (image|picture|photograph)".

Rule 7-51: The data definition for a property with an indicator representation term SHOULD begin with the standard opening phrase "True if ...; false (otherwise|if)...".

Rule 7-52: The data definition for a property with an identification representation term SHOULD begin with the standard opening phrase "(A|An) (optional adjectives) identification...".

Rule 7-53: The data definition for a property with a name representation term SHOULD begin with the standard opening phrase "(A|An) (optional adjectives) name...".

Rule 7-54: The data definition for a property SHOULD begin with the standard opening phrase "(A|An)".

7.2.4.2 Opening phrases for classes

These rules apply to Class objects in CMF, and to complex type definitions in XSD.

Rule 7-55: The data definition for an association class SHOULD begin with the standard opening phrase "A data type for (a relationship|an association)...".

Rule 7-56: The data definition for a class SHOULD begin with the standard opening phrase "A data type..."

7.3 Rules for properties of namespaces

Rule 7-57: The namespace MUST have an identifier, which MUST match the production `<absolute-URI>` as defined by **[RFC 3986]**. In CMF, the namespace identifier is the value of the NamespaceURI property in a Namespace object. In XSD, the namespace identifier is the value of `@targetNamespace` in the `<xs:schema>` element.

Rule 7-58: The namespace MUST have a defined prefix, which MUST match the production *NCName* as defined by **[XML Namespaces]**. In CMF, the prefix is the value of the NamespacePrefix property in a Namespace object. In XSD, the prefix is defined by a namespace binding for the target namespace URI.

Rule 7-59: The namespace MUST have a version, which MUST NOT be empty. In CMF, the version is the value of the NamespaceVersionText property in a Namespace object. In XSD, the version is the value of `@version` in the `<xs:schema>` element.

Rule 7-60: The namespace MUST have a default language, which MUST be a well-formed language tag as defined by **[RFC 4646]**. In CMF, the default language is the value of the NamespaceLanguageName property in a Namespace object. In XSD, the default language is the value of `@xml:lang` in the `<xs:schema>` element.

7.4 Rules for reference namespaces

Rule 7-61: A ·reference namespace· MUST assert the conformance target identifer `https://docs.oasis-open.org/niemopen/ns/specification/XNDR/6.0/#ReferenceSchemaDocument`; all other namespaces MUST NOT. In CMF, this is a value of the ConformanceTargetURI property in the Namespace object. In XSD, this is the ·effective conformance target attribute· of the schema document (*cf.*§6.2).

The conformance target identifier ends in "ReferenceSchemaDocument" instead of "ReferenceNamespace" for historical reasons.

Rule 7-62: In CMF, a Class object with a Namespace that is a ·reference namespace· MUST NOT have the TODO property. In XSD, the ·schema document· for the ·reference namespace· MUST NOT contain the element `xs:any` or `xs:anyAttribute`.

Wilcards are permitted in ·extension namespaces·, but not in ·reference namespaces· or subsets of ·reference namespaces·.

Rule 7-63: A Class object in a ·reference namespace· MUST be augmentable. In the CMF representation, the Class object MUST have an AugmentableIndicator property with a value of `true`. In the XSD representation, the complex type definition MUST contain exactly one element use of its ·augmentation point· element.

To promote reuse, classes defined in ·reference namespaces· and ·extension namespaces· are always augmentable. In a subset of these namespaces, message designers may specify that a class cannot be augmented.

Rule 7-64: In CMF, a Class object or an ObjectProperty object in a ·reference namespace· MUST NOT contain a ReferenceCode property of `ID`, `URI`, or `NONE`. In XSD, a type definition or an element declaration in a ·reference namespace· MUST NOT have an `@appinfo:referenceCode` property of `ID`, `URI`, or `NONE`.

To promote reuse, object properties defined in ·reference namespaces· and ·extension namespaces· are always referencable. In a subset of these namespaces, message designers may specify that some properties must be referenced via IDREF, or by URI, or must appear inline.

Rule 7-65: A component that is used in a ·reference namespace· MUST be defined in a ·reference namespace·.

7.5 Rules for extension namespaces

Rule 7-66: An ·extension namespace· MUST assert the conformance target identifer `https://docs.oasis-open.org/niemopen/ns/specification/XNDR/6.0/#ExtensionSchemaDocument`; all other namespaces MUST NOT. In

CMF, this is a value of the ConformanceTargetURI property in the Namespace object. In XSD, this is the ·effective conformance target attribute· of the schema document (*cf*§6.2).

Rule 7-67: A Class object in an ·extension namespace· MUST be augmentable. In the CMF representation, the Class object MUST have an AugmentableIndicator property with a value of `true`. In the XSD representation, the complex type definition MUST contain exactly one element use of its ·augmentation point· element.

Rule 7-68: In CMF, a Class object or an ObjectProperty object in an ·extension namespace· MUST NOT have a ReferenceCode property of `ID`, `URI`, or `NONE`. In XSD, a type definition or an element declaration in an ·extension namespace· MUST NOT have an `@appinfo:referenceCode` property of `ID`, `URI`, or `NONE`.

7.6 Rules for subset namespaces

Rule 7-69: A ·subset namespace· must assert the conformance target identifer `https://docs.oasis-open.org/niemopen/ns/specification/XNDR/6.0/#SubsetSchemaDocument`. In CMF, this is a value of the ConformanceTargetURI property in the Namespace object. In XSD, this is the ·effective conformances target attribute· of the schema document (*cf*§6.2.

Rule 7-70: A representation of a ·reference namespace· or ·extension namespace· with the same identifer as the ·subset namespace· MUST exist.

It is helpful when a ·message specification· includes the representation of the ·reference namespace· or ·extension namespace·, as this facilitates automated validation of certain rules; however, this is not required, so long as the canonical representation exists somewhere.

Rule 7-71: A subset namespace MUST NOT extend the valid range of a component in the corresponding ·reference namespace· or ·extension namespace·.

Rule 7-72: With the exception of an ·augmentation property·, a ·subset namespace· MUST NOT contain a component not found in the corresponding ·reference namespace· or ·extension namespace·.

Rule 7-73: The data definition of a component in a ·subset namespace· MUST NOT be different than the data definition of the component in its ·reference namespace· or ·extension namespace·.

A subset namespace must not change the text definition of the components it selects.

Rule 7-74: The ChildPropertyAssociation object for an ·augmentation property· MUST declare each ·augmenting namespace·. In the CMF representation the ChildPropertyAssociation object for the ·augmentation property· MUST have an AugmentingNamespace property containing the URI of each ·augmenting namespace·. In the XSD representation, the attribute reference for an ·augmentation property· MUST have the attribute `appinfo:augmentingNamespace` containing the URI of each augmenting namespace.

8. Rules for schema documents

This section contains rules that apply only to the XSD representation of NIEM models; that is, to ·reference schema documents·, ·extension schema documents·, and ·subset schema documents·.

Rule 8-1: The schema document MUST be a conformant document as defined by **[CTAS]**.

Rule 8-2: The ·document element· of the XML document, and only the ·document element·, MUST own an attribute `{https://docs.oasis-open.org/niemopen/ns/specification/conformanceTargets/6.0/}conformanceTargets`.

8.1 Rules for the NIEM profile of XSD

The W3C XML Schema Language provides many features that allow a developer to represent a data model many different ways. 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. The rules in this section establish a profile of XML Schema for NIEM-conformant schemas by forbidding use of the problematic constructs.

Note that ·external schema documents· (i.e., non-NIEM-conformant schema documents) do not need to obey the rules set forth in this section. So long as schema components from external schema documents are adapted for use with NIEM according to the modeling rules in section 8.4: Rules for adapters and external components, they may be used as they appear in the external standard, even if the schema components themselves violate the rules for NIEM-conformant schemas.

Rule 8-3: The XSD representation of a namespace MUST be a ·schema document·, as defined in [XML Schema Structures].

Rule 8-4: The ·document element· of the XSD representation of a namespace MUST be `xs:schema`.

> NOTE: Do we still need this rule? Isn't it part of 8-3?

Rule 8-5: A schema document MUST NOT contain any of the following elements:

- `xs:notation`
- `xs:all`
- `xs:unique`
- `xs:key`
- `xs:keyref`
- `xs:group`
- `xs:attributeGroup`
- `xs:redefine`
- `xs:include`

Rule 8-6: A schema component MUST NOT have an attribute `{}base` with a value of any of these types:

- `xs:ID`
- `xs:IDREF`
- `xs:IDREFS`
- `xs:anyType`
- `xs:anySimpleType`
- `xs:NOTATION`
- `xs:ENTITY`
- `xs:ENTITES`
- any type in the XML namespace `http://www.w3.org/XML/1998/namespace`

Rule 8-7: A schema component MUST NOT have an attribute `{}itemType` with any of the following values:

- `xs:ID`
- `xs:IDREF`
- `xs:anySimpleType`
- `xs:ENTITY`

Rule 8-8: A schema component MUST NOT have an attribute `{}memberTypes` with any of the following values:

- `xs:ID`
- `xs:IDREF`
- `xs:IDREFS`
- `xs:anySimpleType`
- `xs:ENTITY`
- `xs:ENTITIES`

Rule 8-9: A schema component MUST NOT have an attribute `{}type` with any of the following types:

- `xs:ID`
- `xs:IDREF`
- `xs:IDREFS`
- `xs:anySimpleType`
- `xs:ENTITY`
- `xs:ENTITIES`

Rule 8-10: A complex type definition MUST NOT have mixed content.

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, and so *·external schema documents·* may include mixed content and may be used with NIEM.

Rule 8-11: A complex type definition MUST have a `xs:complexContent` or a `xs:simpleContent` child element

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.

Rule 8-12: The base type of a complex type with complex content MUST have complex content.

This rule addresses a peculiarity of the XML Schema definition language, which allows a complex type to be constructed using xs:complexContent, and yet is derived from a complex type that uses xs:simpleContent. These rules ensure that each type has the content style indicated by the schema.

Rule 8-13: An untyped elememt or an element of type `xs:anySimpleType` MUST be abstract.

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.

Rule 8-14: An element type MUST NOT be in the XML Schema namespace or the XML namespace.

Rule 8-15: An element type that is not `xs:anySimpleType` MUST NOT be a simple type.

Rule 8-16: An attribute declaration MUST have a type.

Rule 8-17: An element declaration MUST NOT have an attribute `{}default` or `{}fixed`.

Rule 8-18: An element `xs:sequence` MUST have a `minOccurs` and `maxOccurs` of 1.

Rule 8-19: An element `xs:choice` MUST be a child of `xs:sequence`.

Rule 8-20: An element `xs:choice` MUST have a `minOccurs` and `maxOccurs` of 1.

Rule 8-21: An XML comment SHOULD NOT appear in the schema.

Since XML comments are not associated with any specific XML Schema construct, there is no standard way to interpret comments. XML Schema annotations should be preferred for meaningful information about components. NIEM specifically defines how information should be encapsulated in NIEM-conformant schemas via xs:annotation elements. Comments do not correspond to any metamodel object.

Rule 8-22: A child of element `xs:documentation` MUST be text or an XML comment.

Rule 8-23: An element `xs:import` MUST have an attribute `{}namespace`.

An import that does not specify a namespace is enabling references to components without namespaces. NIEM requires that all components have a defined namespace. It is important that the namespace declared by a schema be universally defined and unambiguous.

Rule 8-24: An element `xs:import` MUST specifiy a schema document, which MUST be a local resource.

The schema document may be specified by a {}schemaLocation attribute in the xs:import element, or by XML Catalog resolution of the {}namespace attribute, or both. Requiring a local resource ensures that the component definitions are known and fixed.

8.2 Rules for type definitions

Rule 8-25: A type definition that does not define a ·*proxy type*· MUST have a name ending in "Type"; all other XSD components MUST NOT.

Use of the representation term Type immediately identifies XML types in a NIEM-conformant schema and prevents naming collisions with corresponding XML elements and attributes. The exception for proxy types ensures that simple NIEM-compatible uses of base XML Schema types are familiar to people with XML Schema experience (*cf*§8.5).

Rule 8-26: A simple type definition MUST have a name ending in "SimpleType"; all other XSD components MUST NOT.

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".

Rule 8-27: A complex type definition MUST be a Class component, a Datatype component, or a ·proxy type·.

Rule 8-28: An element `xs:sequence` MUST be a child of `xs:extension`.

Rule 8-29: An element `xs:sequence` MUST be a child of `xs:extension` or `xs:restriction`.

Restriction is allowed in an extension schema document, but not in reference schema document.

Rule 8-30: A type definition MUST be top-level.

All XML Schema top-level types (children of the document element) are required by XML Schema to be named. By requiring these components to be top level, they are forced to be named and are globally reusable.

Rule 8-31: A complex type definition MUST be an object type, an association type, an adapter type, or an augmentation type.

The rules in this document use the name of a type as the key indicator of the type's category. This makes the rules much simpler than doing a deep examination of each type (and its base types) to identify its category. For complex types, the names follow a pattern:

- Name ends with AdapterType → type represents an ·adapter class·. (see Rule 7-3)
- Name ends with AssociationType → type represents an ·association class·. (see Rule 7-4)
- Name ends with AugmentationType → type is an ·augmentation type·.
- Otherwise → type is the XSD representation of an ·object class·.

Rule 8-32: A type with complex content that does not represent an ·adapter class·, an ·association class·, or an ·augmentation type· MUST be derived from structures:ObjectType or from another object type.

Rule 8-33: A type definition that represents an ·adapter class· MUST be derived from `structures:AdapterType`.

Rule 8-34: A type definition that represents an ·association class· MUST be derinved from `structures:AssociationType` or from another ·association class·.

Rule 8-35: A type definition that is an ·augmentation type· MUST be derived from `structures:AugmentationType`.

Rule 8-36: A complex type definition with simple content MUST include `structures:SimpleObjectAttributeGroup`.

Rule 8-37: The base type definition of a type definition MUST have the target namespace or the XML Schema namespace or a namespace that is imported as conformant.

Rule 8-38: An attribute or element reference MUST have the target namespace or a namespace that is imported as conformant.

Rule 8-39: An attribute group reference MUST be `structures:SimpleObjectAttributeGroup`.

In ·conformant schema documents·, use of attribute groups is restricted. The only attribute group defined by NIEM for use in conformant schemas is `structures:SimpleObjectAttributeGroup`. This attribute group provides the attributes necessary for identifiers and references.

Rule 8-40: A complex type definition MUST NOT have an element use of an augmentation element declaration, or an element declaration that is in the substitition group of an augmentation point element declaration.

Augmentation elements do not correspond to a model component, and must not be used as a property in any class.

Rule 8-41: The item type of a list simple type definition MUST have a target namespace equal to the target namespace of the XML Schema document within which it is defined, or a namespace that is imported as conformant by the schema document within which it is defined.

Rule 8-42: Every member type of a union simple type definition MUST have a target namespace that is equal to either the target namespace of the XML Schema document within which it is defined or a namespace that is imported as conformant by the schema document within which it is defined.

8.3 Rules for attribute and element declarations

Rule 8-43: The name of an element declaration or attribute declaration MUST NOT end in "Literal".

Literal properties appear only in the CMF representation of an ·atomic class·.

Rule 8-44: An attribute declaration or element declaration MUST be top-level.

Rule 8-45: An element declaration MUST NOT have a simple type.

Rule 8-46: The type definition of an attribute or element declaration MUST have a target namespace that is the target namespace, or a namespace that is imported as conformant.

Rule 8-47: An element substitution group MUST have either the target namespace or a namespace that is imported as conformant.

8.4 Rules for adapters and external components

Rule 8-48: An `xs:import` element importing an external schema document MUST own the attribute `appinfo:externalImportIndicator` with a value of `true`.

An ·external schema document· is any schema document that is not

- a ·reference schema document·, or
- an ·extension schema document·, or
- a ·subset schema document·, or
- a schema document that has the structures namespace as its target namespace, or
- a schema document that has the XML namespace as its target namespace.

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 schema documents.

A schema component defined by an external schema document may be called an external component. A NIEM-conformant type may use external components in its definition. There are two ways to integrate external components into a NIEM-conformant schema:

- An ·adapter class· may be constructed from externally-defined elements and attributes. A goal of this method is to represent, as a single unit, a set of data that embodies a single concept from an external standard.

- A type that is not an external adapter type, and which is defined by an extension or subset schema document, may incorporate an externally-defined attribute.

Rule 8-49: An `xs:import` element importing an external schema document MUST be a documented component.

A NIEM-conformant schema has well-known documentation points. Therefore, a schema that imports a NIEM-conformant namespace need not provide additional documentation for the imported namespace. However, when an external schema document is imported, appropriate documentation must be provided on the xs:import element. This ensures that documentation for all external schema documents will be both available and accessible in a consistent manner.

Rule 8-50: An ·adapter type· MUST have a name ending in "AdapterType"; all other type definitions MUST NOT.

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

An external 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 document. Most likely, the external schema document 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 document, and the external adapter type may use those components.

In normal (conformant) type definition, a reference to an attribute or element is a reference to a documented component. Within an external 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 8-51: An external adapter type definition MUST be a complex type definition with complex content that extends structures:ObjectType, and that uses xs:sequence as its top-level compositor.

Rule 8-52: An element reference that appears within an external adapter type MUST have a target namespace that is imported as external.

Rule 8-53: An external adapter type definition MUST NOT be a base type definition.

Rule 8-54: An external attribute use MUST be a documented component with a non-empty data definition.

Rule 8-55: An attribute use schema component MUST NOT have an {attribute declaration} with an ID type.

NIEM schemas use `structures:id` to enable references between components. Each NIEM-defined complex type in a reference or extension schema document must incorporate a definition for `structures:id`. [XML] Section 3.3.1, Attribute Types entails that a complex type may have no more than one ID attribute. This means that an external attribute use must not be an ID attribute.

The term "attribute use schema component" is defined by **[XML Schema Structures]** Section 3.5.1, The Attribute Use Schema Component. Attribute type ID is defined by**[XML]** Section 3.3.1, Attribute Types.

Rule 8-56: An external attribute use MUST be a documented component with a non-empty data definition.

8.5 Rules for proxy types

Rule 7-75: The XSD declaration of a ·proxy type· MUST have the same name as the simple type it extends.

A ·proxy type· is an XSD complex type definition with simple content that extends one of the simple types in the XML Schema namespace with `structures:SimpleObjectAttributeGroup`; for example:

```
<xs:complexType name="string">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

A ·proxy type· is not a model component. It is a convenience complex type definition wrapper for a simple type in the XML Schema namespace; for example, `niem-xs:token` is a proxy type for `xs:token`. Unlike other complex type definitions, proxy types have the same local name as the builtin simple type. This is done to make conformant schemas more understandable to people that are familiar with the names of the XML Schema namespace simple types.

Rule 8-57: A proxy type MUST have the designated structure. It MUST use `xs:extension`. It MUST NOT use `xs:attribute`. It MUST include exactly one `xs:attributeGroup` reference, which must be to `structures:SimpleObjectAttributeGroup`.

8.6 Rules for augmentations

Rule 8-58: The XSD definition of an ·augmentation type· MUST have a name ending in "AugmentationType"; all other XSD components MUST NOT.

Rule 8-59: The XSD declaration of an ·augmentation element· MUST have a name ending in "Augmentation"; all other XSD components MUST NOT.

Rule 8-60: The XSD declaration of an ·augmentation point element· MUST have a name ending in "AugmentationPoint"; all other XSD components MUST NOT.

Rule 8-61: The data definition for an augmentation point element SHOULD begin with standard opening phrase "An augmentation point...".

Rule 8-62: The data definition for an augmentation element SHOULD begin with the standard opening phrase "Supplements..." or "Additional information about...".

Rule 8-63: The data definition for an augmentation type SHOULD begin with the standard opening phrase "A data type (that supplements|for additional information about)...".

Rule 8-64: A schema document containing an element declaration for an ·augmentation point element· MUST also contain a type definition for its augmented base type.

For example, a schema document with an element declaration for `FooAugmentationPoint` must also contain a type definition for `FooType`.

Rule 8-65: An augmentation point element MUST have no type.

Rule 8-66: An augmentation point element MUST have no substitution group.

Rule 8-67: An augmentation point element MUST only be referenced by its base type.

For example, the `FooAugmentationPoint` element must not be included in any type other than `FooType`.

Rule 8-68: An augmentation point element particle MUST have attribute `minOccurs` equal to 0 and attribute `maxOccurs` set to unbounded.

Rule 8-69: An augmentation point element particle MUST be the last element occurrence in the content model of its augmentable type.

8.7 Rules for machine-readable annotations

Rules for non-appinfo annotations TODO

NIEM defines a single namespace that holds components for use in NIEM-conformant schema application information, represented by the URI `https://docs.oasis-open.org/niemopen/ns/model/appinfo/6.0/`. This namespace is referred to as the ·appinfo namespace·.

Rule 8-70: An attribute in the ·appinfo namespace· MUST be owned by an element with a namespace name `http://www.w3.org/2001/XMLSchema` .

Rule 8-71: A child of element `xs:appinfo` MUST be an element, a comment, or whitespace text.

Rule 8-72: An element that is a child of `xs:appinfo` MUST have a namespace name.

Rule 8-73: An element that is a descendent of `xs:appinfo` MUST NOT have the XML Schema namespace.

Rule 8-74: A schema component that has an attribute appinfo:deprecated with a value of true MUST be a deprecated component.

Rule 8-75: When the element `appinfo:LocalTerm` appears in a schema document, it MUST be application information on an element `xs:schema`.

8.8 Rules for reference schema documents

Rule 8-76: A ·reference schema document· MUST NOT have an attribute `{}final`.

Rule 8-77: A simple type constraining facet in a ·reference schema document· MUST NOT have an attribute `{}fixed`.

Rule 8-78: In a ·reference schema document·, a complex type definition with simple content schema component MUST have a derivation method of extension.

Rule 8-79: A ·reference schema document· MUST NOT contain the attribute `{}block` or `{}blockDefault`.

Rule 8-80: A ·reference schema document· MUST NOT contain the attribute `{}final` or `{}finalDefault`.

Rule 8-81: An element declaration in a ·reference schema document· MUST have the `{nillable}` property with a value of true.

Properties in a reference or extension namespace are always referencable, in order to maximize reuse. Message designers may make some properties un-referencable in a namespace subset.

Rule 8-82: A ·reference schema document· MUST NOT contain the element `xs:choice`.

8.9 Rules for extension schema documents

Rule 8-83: An element declaration in an ·extension schema document· MUST have the `{nillable}` property with a value of true.

9. Rules for models

These rules apply to both the CMF and XSD representations of a model.

Rule 9-1: Every namespace in a model MUST be one of the following:

- a ·conforming namespace·; that is, a ·reference namespace·, ·extension namespace·, or ·subset namespace·
- an ·external namespace·
- the ·structures namespace·
- the XML namespace, `http://www.w3.org/XML/1998/namespace`
- the XSD namespace, `http://www.w3.org/2001/XMLSchema`.

Rule 9-2: A model MUST NOT contain two namespaces with the same prefix.

In a model there is always a one-to-one match between namespace prefix and namespace URI.

9.1 Rules for model files

Rule 9-3: A model MUST NOT contain two namespaces with the same identifier.

This is impossible in an XSD representation of a model.

9.2 Rules for schema document sets

A ·schema document set· is a collection of ·schema documents· that together are capable of validating an XML document.

Rule 9-4: The ·schema documents· in a ·schema document set· MUST be exactly those determined by the following procedure:

- Beginning with the empty set
- Add one or more specified initial ·schema documents·
- As each ·schema document· is added, find each `<xs:import>` element contained therein, and add the ·schema document· specified by that element to the set.

Schema assembly is underspecified in **[XML Schema]**. But a specification that defines message conformance in terms of schema validation must have some way to establish the schema used to assess validity. Otherwise no one can be certain what conforms. This rule establishes the needed certainty.

Most schema document sets are established by a single extension schema document, with all other needed schema documents brought in by `xs:import` elements. But it is also allowable to include every document as an initial schema document. Or to have a single initial document with no namespace, containing nothing but `xs:import` elements for each document in the set.

Rule 9-5: The members of a ·schema document set· MUST NOT contain two `xs:import` elements that have the same `{}namespace` attribute but specify different schema documents.

XML Schema permits conflicting imports, but the result is underspecified, and often causes errors that are very hard to detect and diagnose.

Rule 9-6: The members of a ·schema document set· MUST NOT contain two `xs:import` elements with the same namespace but different values for `appinfo:externalImportIndicator`.

Rule 9-7: The members of a ·schema document set· MUST NOT contain two `xs:import` elements with non-empty ·data definitions· that are different.

An ·external schema document· is usually imported once in a ·schema document set·, and imports of other ·schema documents· are usually not documented, so this rule is rarely applicable.

Rule 9-8: There MUST be a one-to-one match between namespace prefix and namespace URI among all the members of a ·schema document set·.

XML Schema permits a schema document set to contain

- schema document A containing `xmlns:foo="http://example.com/MyFoo/"`

- schema document B containing `xmlns:bar="http://example.com/MyFoo/"`
- schema document C containing `xmlns:foo="http://example.com/MyBar/"`

This is not allowed in NIEM XSD. There is always a one-to-one match between namespace prefix and URI in CMF.

Rule 9-9: A ·schema document set· MUST be complete; that is, it MUST contain the definition of every schema component referenced by any component defined by the schema set.

A ·schema document set· defines an XML Schema that may be used to validate an XML document. This rule ensures that a schema document set under consideration contains definitions for everything that it references; it has everything necessary to do a complete validation of XML documents, without any unresolved references. Note that some tools may allow validation of documents using partial schemas, when components that are not present are not exercised by the XML document under validation. Such a schema does not meet qualify as a conformant schema document set.

Rule 9-10: A ·schema document set· MUST include the ·structures namespace· as it is defined in Appendix B of this document.

This rule further enforces uniform and consistent use of the NIEM structures namespace, without addition. Users are not allowed to insert types, attributes, etc. that are not specified by this document.

10. Rules for message types and message formats

Rule 10-1: A ·message type· MUST declare the class and property for the ·initial node· of conforming ·messages·.

A ·message model· alone is insufficient to completely define the content of conforming ·messages·. The ·message model· defines the content of several properties, but does not say which of those properties are required in a conforming ·message·.

For example, the ·message type· for the ·message· in figure 3-2 (reproduced below) must declare that the initial property is `msg:Request` and the initial class is `msg:RequestType`. Otherwise, the single element `<nc:ItemName>Wrench</nc:ItemName>` would be a valid ·message·, because it is valid according to the ·message model·.

```
<msg:Request                                                | {
 xmlns:nc="https://docs.oasis-open.org/niemopen/ns/model/   |   "@context": "http://example.com/ReqRes/Request/JSON/1.0",
 xmlns:msg="http://example.com/ReqRes/1.0/">                |   "msg:Request": {
  <msg:RequestID>RQ001</msg:RequestID>                      |     "msg:RequestID" : "RQ001",
  <msg:RequestedItem>                                       |     "msg:RequestedItem": {
    <nc:ItemName>Wrench</nc:ItemName>                       |       "nc:ItemName": Wrench",
    <nc:ItemQuantity>10</nc:ItemQuantity>                   |       "nc:ItemQuantity": 10
  </msg:RequestedItem>                                      |      }
</msg:Request>                                              |   }
                                                            | }
```

This document does not specify any particular syntax for the declaration.

Rule 10-2: The ·schema· for a ·message format· MUST validate exactly those ·messages· that conform to the format's ·message type·.

This is the only conformance rule for the XML Schema in an XML message format, or the JSON Schema in a JSON message format. NIEMOpen provides free and open-source software to generate conforming schemas from the message type. Developers are also free to construct those schemas by hand.

11. Rules for messages

Beyond this point THERE ARE DRAGONS!

12.2 Identifiers and references

Nested elements, shown above, are sufficient to represent simple data that takes the form of a tree. However, the use of nested elements has limitations; expression of all relationships via nested elements is not always possible. Situations that cause problems include:

- Cycles: some object has a relationship that, when followed, eventually circles back to itself. For example, suppose that Bob has a sister relationship to Sue, who has a brother relationship back to Bob. This is not a tree, and so needs some representation other than just nested elements.
- Reuse: multiple objects have a relationship to a common object. For example, suppose Bob and Sue both have a mother relationship to Sally. Expressed via nested elements, this would result in a duplicate representation of Sally.

NIEM provides two different ways to solve this problem: the use of local references pointing to local identifiers, and the use of uniform resource identifiers (URIs). These two methods are similar, and can interoperate, but have distinctions, as described by Section XX Differentiating reference-to-identifier links and use of URIs, below.

Rule 12-1: An element MUST NOT have more than one attribute that is `structures:id`, `structures:ref`, or `structures:uri`.

12.2.1 Local identifiers and references

The XML specifications define ID and IDREF attributes, which act as references in XML data. This is supported by XML Schema, and NIEM uses ID and IDREF as one way to reference data across data objects. Under this framework:

- Within an XML document, each value of any attribute of type `xs:ID` must be unique. For example, if an element has an attribute of type `xs:ID` with the value of Bob, then there may not be any other attribute in the document that is of type `xs:ID` that also has the value Bob. NIEM provides attribute `structures:id` of type `xs:ID` to act as a standard local identifier.
- Within an XML document, the value of any attribute of type `xs:IDREF` must appear somewhere within the document as the value of some attribute of type `xs:ID`. For example, if an attribute of type `xs:IDREF` has the value Bob, then somewhere within that XML document there must be an attribute of type `xs:ID` with the value Bob. NIEM provides attribute `structures:ref` of type `xs:IDREF` as a standard local reference.
- These constraints, that IDs must be unique, and that IDREFs must refer to IDs, are XML constraints, not unique to NIEM.
- There are additional constraints placed on XML documents and XML schemas regarding the use of ID and IDREF attributes. For example, an element may not have two attributes of type ID.

In short, within a NIEM-conformant XML document, an attribute `structures:ref` refers to an attribute `structures:id`. These attributes may appear in an XML document to express that an object that is the value of an element is the same as some other object within the document. For example, in the following example, the user of the weapon (Bart) is the same person that is the subject of the arrest:

```
<j:Arrest>
  <j:ArrestInvolvedWeapon>
    <nc:WeaponUser structures:id="bart">
      <nc:PersonName>
        <nc:PersonGivenName>Bart</nc:PersonGivenName>
      </nc:PersonName>
    </nc:WeaponUser>
  </j:ArrestInvolvedWeapon>
  <j:ArrestSubject>
    <nc:EntityPerson structures:ref="bart" xsi:nil="true"/>
  </j:ArrestSubject>
</j:Arrest>
```

Figure 12-1: Example of `structures:id` and `structures:ref`

Note that rules below establish that relationships established using `structures:id` and `structures:ref` have the exact same meaning as relationships established using nested elements. An information exchange specification may constrain them differently, or prefer one over the other, but from a NIEM perspective, they have the same meaning.

Rule 12-2: The value of an attribute `structures:ref` MUST match the value of an attribute `structures:id` of some element in the XML document.

Although many attributes with ID and IDREF semantics are defined by many vocabularies, for consistency, within a NIEM XML document any attribute `structures:ref` must refer to an attribute `structures:id`, and not any other attribute.

Rule 12-3: Every element that has an attribute `structures:ref` MUST have a referencing element validation root that is equal to the referenced element validation root.

The term "validation root" is defined by **[XML Schema Structures]** *Section 5.2, Assessing Schema-Validity*. It is established as a part of validity assessment of an XML document.

NIEM supports type-safe references; that is, references using `structures:ref` and `structures:id` must preserve the type constraints that would apply if nested elements were used instead of a reference. For example, an element of type `nc:PersonType` must always refer to another element of type `nc:PersonType`, or a type derived from `nc:PersonType`, when using `structures:ref` to establish the relationship.

Rule 12-4: Every element that has an attribute `structures:ref` MUST have a referenced element type definition that is validly derived from the referencing element type definition.

The term "validly derived" is as established by **[XML Schema Structures]**, subsection *Schema Component Constraint: Type Derivation OK (Complex)* within Section 3.4.6, *Constraints on Complex Type Definition Schema Components*.

This rule requires that the type of the element pointed to by a `structures:ref` attribute must be of (or derived from) the type of the reference element.

12.2.2 Uniform resource identifiers in NIEM XML

NIEM supports linked data through the use of uniform resource identifiers (URIs), expressed through the attribute structures:uri in XML documents . This attribute works much like `structures:ref` and `structures:id`, and overlaps somewhat. Linked data introduces key terminology:

- Anything modeled or addressed by an information system may be called a *resource*: people, vehicles, reports, documents, relationships, ideas: anything that is talked about and modeled in an information system is a resource.
- Every resource may have a name, called a uniform resource identifier (URI).

As described in Section 5.4, Unique identification of data objects, above, `structures:uri`, `structures:id`, and `structures:ref` each denote a resource identifier. Although a `structures:ref` must always refer to a `structures:id`, and a value of `structures:id` must be unique within its document, a `structures:uri` may refer to any of `structures:uri`, `structures:ref`, or `structures:id`.

Rule 12-5: The value of an attribute `structures:uri` is a URI-reference, as defined by **[RFC 3986]**, which denotes a resource identifier on the element holding the attribute, in accordance with evaluation consistent with **[RFC 3986]** and **[XML Base]**.

Rule 12-6: The value of an attribute `structures:uri` that is an *absolute URI* according to **[RFC 3986]** MUST be valid according to the rules for its scheme.

Validating parsers do not always test whether an absolute URI follows the rules for its scheme, and so a A successful validation match against the type `xs:anyURI` is necessary but not sufficient.

The following example shows a reference to an absolute URI, using the URN namespace for UUIDs:

```
<example:ArrestMessage>
  <j:Arrest xsi:nil="true"
    structures:uri="urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"/>
</example:ArrestMessage>
```

Figure 12-2: Example of `structures:uri` holding an absolute URI

The following example shows a relative URI, using xml:base to carry the base URI for the document. The person object identified by the structures:uri attribute has the URI http://state.example/scmods/B263-1655-2187.

```
<example:ArrestMessage xml:base="http://state.example/scmods/">
  <j:Arrest>
    <j:ArrestSubject>
      <nc:EntityPerson structures:uri="B263-1655-2187"/>
    </j:ArrestSubject>
  </j:Arrest>
</example:ArrestMessage>
```

Figure 12-3: Example of `structures:uri` holding an relative URI, with an `xml:base`

Rule 12-7: The value of an attribute `structures:id` with a value of *val*, or an attribute `structures:ref` with a value of *val*, denotes a resource identifier on the element holding the attribute, as would be denoted by an attribute `structures:uri` with a value of *#val*.

For example, `structures:id="hello"` and `structures:ref="hello"` each denote the same resource identifier for an element as if it held an attribute `structures:uri="#hello"`.

A set of elements that each have the same resource identifier denote the same object, which has that given identifier. This means that, in an XML representation, the properties of an object may be spread across a set of elements that share an identifier.

The following example contains four references to the same object, which has the identifier `https://state.example/98723987/results.xml#delta`.

```
<example:ArrestMessage xml:base="https://state.example/98723987/results.xml">
  <j:Arrest>
    <j:ArrestSubject>
      <nc:EntityPerson structures:id="delta"/>
    </j:ArrestSubject>
  </j:Arrest>
  <j:Arrest>
    <j:ArrestSubject>
      <nc:EntityPerson structures:ref="delta"/>
    </j:ArrestSubject>
  </j:Arrest>
  <j:Arrest>
    <j:ArrestSubject>
      <nc:EntityPerson structures:uri="#delta"/>
    </j:ArrestSubject>
  </j:Arrest>
  <j:Arrest>
    <j:ArrestSubject>
      <nc:EmtotuPerson structures:uri="https://state.example/98723987/results.xml#delta"/>
    </j:ArrestSubject>
  </j:Arrest>
</example:ArrestMessage>
```

Figure 12-4: Example of `structures:id`, `structures:ref`, and `structures:uri` identifying the same object

12.2.3 Differentiating reference-to-identifier links and use of URIs

These two methods are similar, and can interoperate, but have distinctions:

- With ref-to-id links, both structures:ref and structures:id are required to be within the same document.
- With ref-to-id links, both structures:id and structures:ref are required to be validated against the same schema.
- Ref-to-id links provide and require type safety, in that the type of an object pointed to by structures:ref must be consistent with the referencing element’s type declaration.
- The value of structures:id must be unique for IDs within the document.
- The value of structures:ref must appear within the document as the value of an attribute structures:id.
- An attribute structures:uri is a URI-reference that can reference any resource, inside or outside the document.

- A structures:uri can reference any structures:id within the same document, or in another conformant document.
- A structures:uri can reference any structures:ref within the same document, or in another conformant document.
- Any structures:uri may reference any other structures:uri, within the same document, or in another conformant document.

12.2.4 Reference and content elements have the same meaning

An important aspect of the use of nested elements, ref-to-id references, and URI references, is that they all have the same meaning. Expressing a relationship as a nested element, versus as a ref-to-id reference is merely for convenience and ease of serialization. There is no change in meaning or semantics between relationships expressed by sub-elements versus relationships expressed by structures:ref or structures:uri.

Any claim that nested elements represent composition, while references represent aggregation is incorrect. No life cycle dependency is implied by either method. Similarly, any claim that included data is intrinsic (i.e., a property inherent to an object), while referenced data is extrinsic (i.e., a property derived from a relationship to other things), is false. A property represented as a nested element has the exact same meaning as that property represented by a reference.

Rule 12-8: There MUST NOT be any difference in meaning between a relationship established via an element declaration instantiated by a nested element, and that element declaration instantiated via reference.

There is no difference in meaning between relationships established by sub-elements and those established by references. 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 `xs:ID` and `xs: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 within an XML document. Also, any value of `structures:ref` must also appear as the value of an ID.

By this rule, the following three XML fragments have a very similar meaning. The first example, with no reference, shows a witness that is a role of a person.

```
<j:Witness>
  <nc:RoleOfPerson>
    <nc:PersonName>
      <nc:PersonFullName>John Doe</nc:PersonFullName>
    </nc:PersonName>
  </nc:RoleOfPerson>
</j:Witness>
```

Figure 12-5: Example with no reference

The next example, with a backward reference, also expresses a witness object that is a role of a person. It first expresses the person object, then the witness object as a role of a that person, expressed as a reference back to the person.

```
<nc:Person structures:id="c58">
  <nc:PersonName>
    <nc:PersonFullName>John Doe</nc:PersonFullName>
  </nc:PersonName>
</nc:Person>
<j:Witness>
  <nc:EntityPerson structures:ref="c58" xsi:nil="true"/>
</j:Witness>
```

Figure 12-6: Example with a backward reference

The final example, with a forward reference shows a witness as a role of a person, with a separate person object expressed as a forward reference to the person object that is expressed later, within the definition of the witness.

```
<nc:Person structures:ref="t85" xsi:nil="true"/>
<j:Witness>
  <nc:EntityPerson structures:id="t85">
    <nc:PersonName>
      <nc:PersonFullName>John Doe</nc:PersonFullName>
    </nc:PersonName>
  </nc:RoleOfP
```

Figure 12-7: Example with a forward reference

NIEM-conformant data instances may use either representation as needed, to represent the meaning of the fundamental data. There is no difference in meaning between reference and content 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.

13. Rules for the NIEM profile of JSON-LD

13.1 Components are globally reusable

Applicable Rules:

- Rule 9-10, Simple type definition is top-level (REF, EXT)
- Rule 9-25, Complex type definition is top-level (REF, EXT)
- Rule 9-36, Element declaration is top-level (REF, EXT)
- Rule 9-48, Attribute declaration is top-level (REF, EXT)

Each component defined by a NIEM-conformant JSON Schema may be reused from outside the schema document. Every schema component that is defined by a NIEM-conformant JSON Schema is given an explicit name. These components are defined as neither local nor anonymous.

These components are defined within the top level `properties` and `definitions` blocks.

13.2 Enumeration Data Definitions

Applicable Rule:

- Rule 9-14 Enumeration has data definition (REF, EXT)

Enumerations MUST have `description` objects providing the full value of each enumeration, to preserve the semantics of the codes. In the example below, the code "1" has no semantic meaning on its own; "Fatal Accident" does.

```
"aamva_d20:AccidentSeverityCodeType": {
    "description": "A data type for severity levels of an accident.",
    "type": "string",
    "oneOf": [
        {
            "const": "1",
            "description": "Fatal Accident"
        },
        {
            "const": "2",
            "description": "Incapacitating Injury Accident"
        },
        {
            "const": "3",
            "description": "Non-incapacitating Evident Injury"
        },
        {
            "const": "4",
            "description": "Possible Injury Accident"
        },
        {
            "const": "5",
            "description": "Non-injury Accident"
        },
        {
            "const": "9",
            "description": "Unknown"
        }
    ]
}
```

13.3 Normalization for JSON Schema

NIEM property and class names can be lengthy and some development environments can have issues with using the fully qualified names that NIEM uses, e.g. `nc:PersonGivenName`. NIEM allows mapping of these longer fully qualified names to shorter names. The mappings are put in the `@context` object along with namespace mappings.

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "type": "object",
    "@context" : {
        "nc": "http://release.niem.gov/niem/niem-core/5.0/",
        "name": "nc:PersonName",
        "first": "nc:PersonGivenName",
        "middle": "nc:PersonMiddleName",
        "last": "nc:PersonSurName",
        "nametype": "nc:PersonNameType",
        "nametexttype": "nc:PersonNameTextType",
        "propernametexttype": "nc:ProperNameTextType",
        "texttype": "nc:TextType"
    },

    "properties": {
        "name": {
            "description": "A combination of names and/or titles by which a person is known.",
            "type": "array",
            "items": {"$ref": "#/definitions/nametype"}
        },
        "first": {
            "description": "A first name of a person.",
            "$ref": "#/definitions/nametexttype"
        },
        "middle": {
            "description": "A middle name of a person.",
            "type": "array",
            "items": {"$ref": "#/definitions/nametexttype"}
        },
        "last": {
            "description": "A last name or family name of a person.",
            "$ref": "#/definitions/nametexttype"
        }
    },
    "definitions": {
        "nametype": {
            "description": "A data type for a combination of names and/or titles by which a person is known.",
            "type": "object",
            "properties": {
                "first": {"$ref": "#/properties/first"},
                "middle": {"$ref": "#/properties/middle"},
                "last": {"$ref": "#/properties/last"},
            },
            "required": ["last"]
        },
        "nametexttype": {
            "description": "A data type for a name by which a person is known, referred, or addressed.",
            "$ref": "#/definitions/propernametexttype"
        },
        "propernametexttype": {
            "description": "A data type for a word or phrase by which a person or thing is known, referred, or addressed.",
            "$ref": "#/definitions/texttype"
        },
        "texttype": {
            "description": "A data type for a character string.",
            "type": "string"
        }

    }
}
```

See section *14.5 Normalization for JSON Instances*, below, for normalized instances.

14. Rules for NIEM messages in JSON

14.1 Instance Must be Schema-Valid

Applicable Rule:

> Rule 12-1 (Constraint) The XML document MUST be schema-valid, as assessed against a conformant schema document set, composed of authoritative, comprehensive schema documents for the relevant namespaces.

JSON messages MUST also be schema-valid, as assessed against JSON one or more authoritative, comprehensive JSON Schema documents.

The schemas that define the exchange must be authoritative. Each is the reference schema or extension 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 JSON Schema validation must be performed on all JSON instances as they are served or consumed, only that the JSON instances validate if JSON validation is performed. The JSON Schema component definitions specify JSON documents and element information items, and the instances should follow the rules given by the schemas, even when validation is not performed.

14.2 Empty Content Has No Meaning

Applicable Rule:

> Rule 12-2 (Interpretation) Within the instance, the meaning of an element with no content is that additional properties are not asserted. There MUST NOT be additional meaning interpreted for an element with no content.

In this example, neither empty object has any implied meaning:

```
{
    SomeProperty1: "",
    SomeProperty2: []
}
```

14.3 Referencing

Applicable Rules:

- Rule 12-3 Element has only one resource identifying attribute (INS)
- Rule 12-7 `structures:uri` denotes resource identifier (INS)
- Rule 12-8 `structures:id` and `structures:ref` denote resource identifier (INS)

JSON-LD provides a means of identifying resources, both internally and externally, through the use of `@id`. Used on its own, `@id` provides a local identification string that identifies an object. All objects identified with the same string are the same object, linked to each other via the matching `@id` properties.

Combined with `"@type": "@vocab"`, an `@id` becomes an Internationalized Resource Identifier (IRI), linking together all object in an instance with the same IRI.

```
"SomeLocalIdentifiedProperty": {
      "@id": "some-id-string"
    },
"SomeGlobalIdentifiedProperty": {
      "@id": "http://some-uri/",
      "@type": "@vocab"
    }
```

Applicable Rules:

- Rule 12-5 Linked elements have same validation root (INS)
- Rule 12-6 Attribute structures:ref references element of correct type (INS)

- Rule 12-9 Nested elements and references have the same meaning. (INS)

JSON Schema lacks the object oriented classes of CMF or XML Schema, so the concept of type derivation doesn't strictly apply. Despite that limitation, references in JSON instances MUST link objects that are of the same "type" or "derived" from the same "type." People objects need to link to people objects.

For example, an `nc:Person` object can be linked to other objects of `nc:PersonType`, but also to objects derived from `nc:PersonType`, like a `j:MedicalPractitioner` object.

```
"nc:Person": {
    "@id": "ABC123"
    "nc:PersonName": {
        "nc:PersonFullName": "Doctor Who"
    }
},
"j:MedicalPractitioner": {
    "@id": "ABC123"
}
```

It may be helpful to refer to the source CMF to more clearly see types and derivations.

There MUST NOT be any difference in meaning between a relationship established via an element declaration instantiated by a nested element, and that element declaration instantiated via reference. In other words, linking objects together establishes that the object are the same object, and one could be conceptually substituted for the other.

14.4 Order of Properties

Applicable Rule:

- 12-10 Order of properties is expressed via structures:sequenceID (INS)

Because repeated properties are represented as ordered arrays of values in JSON, their order is automatically preserved. Whether this ordering is meaningful is up to the exchange developer and should be documented apart from the JSON Schema. If ordering is meaningful across different objects, for example regarding first, middle, and last names, the exchange developer must devise their own mechanism for documenting this ordering.

14.5 Normalization for JSON Instances

Instance with fully qualified names:

```
{
    "nc:PersonName": [
        {
            "nc:PersonGivenName": "Peter",
            "nc:PersonMiddleName": [
                "Death",
                "Bredon"
            ],
            "nc:PersonSurName": "Wimsey"
        }
    ]
}
```

Instance with short names:

```
{
    "@context" : {
        "nc": "http://release.niem.gov/niem/niem-core/5.0/",
        "name": "nc:PersonName",
        "first": "nc:PersonGivenName",
        "middle": "nc:PersonMiddleName",
        "last": "nc:PersonSurName"
    },

    "name": [
        {
            "first": "Peter",
            "middle": [
                "Death",
                "Bredon"
            ],
            "last": "Wimsey"
        }
    ]
}
```

As with namespace declarations above, the `@context` object MUST exist as part of the specification, but does not need to be included in JSON instances being exchanged. Structure, however, cannot be simplified via this mapping process.

Appendix A. References

This appendix contains the normative and informative references that are used in this document. Any normative work cited in the body of the text as needed to implement the work product must be listed in the Normative References section below. Each reference to a separate document or artifact in this work must be listed here and must be identified as either a Normative or an Informative Reference. Normative references are specific (identified by date of publication and/or edition number or version number) and Informative references are either specific or non-specific.

While any hyperlinks included in this appendix were valid at the time of publication, OASIS cannot guarantee their long-term validity.

A.1 Normative References

The following documents are referenced in such a way that some or all of their content constitutes requirements of this document.

[ClarkNS]

Clark, J. XML Namespaces, 4 February 1999. Available from http://www.jclark.com/xml/xmlns.htm.

[ISO 11179-4]

ISO/IEC 11179-4 Information Technology — Metadata Registries (MDR) — Part 4: Formulation of Data Definitions Second Edition, 15 July 2004. Available from http://standards.iso.org/ittf/PubliclyAvailableStandards/c035346_ISO_IEC_11179-4_2004(E).zip.

[ISO 11179-5]

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.

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, http://www.rfc-editor.org/info/rfc2119.

[RFC8174]

Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, http://www.rfc-editor.org/info/rfc8174.

A.2 Informative References

[I18N]

Renner, S, "Internationalization scenarios for NIEM", June 2022. **Publish as a project note TODO**

[RFC3552]

Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, https://www.rfc-editor.org/info/rfc3552.

[Tools]

Add a reference to a NIEMOpen tools page TODO.

Appendix B. Structures namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="https://docs.oasis-open.org/niemopen/ns/model/structures/6.0/"
  xmlns:structures="https://docs.oasis-open.org/niemopen/ns/model/structures/6.0/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  version="ps02"
  xml:lang="en-US">
  <xs:annotation>
```

```
    <xs:documentation>The structures namespace provides base types and other components for definition of NIEM-conformant XML
schemas.</xs:documentation>
  </xs:annotation>
  <xs:attributeGroup name="SimpleObjectAttributeGroup">
    <xs:attribute ref="structures:id"/>
    <xs:attribute ref="structures:ref"/>
    <xs:attribute ref="structures:uri"/>
    <xs:anyAttribute processContents="strict" namespace="##other"/>
  </xs:attributeGroup>
  <xs:complexType name="AdapterType" abstract="true">
    <xs:annotation>
      <xs:documentation>A data type for a type that contains a single non-conformant property from an external standard for use
in NIEM.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element ref="structures:ObjectAugmentationPoint" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="structures:appliesToParent"/>
    <xs:attribute ref="structures:id"/>
    <xs:attribute ref="structures:ref"/>
    <xs:attribute ref="structures:uri"/>
    <xs:anyAttribute processContents="strict" namespace="##other"/>
  </xs:complexType>
  <xs:complexType name="AssociationType" abstract="true">
    <xs:annotation>
      <xs:documentation>A data type for a relationship between two or more objects, including any properties of that
relationship.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element ref="structures:AssociationAugmentationPoint" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="structures:appliesToParent"/>
    <xs:attribute ref="structures:id"/>
    <xs:attribute ref="structures:ref"/>
    <xs:attribute ref="structures:uri"/>
    <xs:anyAttribute processContents="strict" namespace="##other"/>
  </xs:complexType>
  <xs:complexType name="AugmentationType" abstract="true">
    <xs:annotation>
      <xs:documentation>A data type for a set of properties to be applied to a base type.</xs:documentation>
    </xs:annotation>
  </xs:complexType>
  <xs:complexType name="ObjectType" abstract="true">
    <xs:annotation>
      <xs:documentation>A data type for a thing with its own lifespan that has some existence.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element ref="structures:ObjectAugmentationPoint" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="structures:appliesToParent"/>
    <xs:attribute ref="structures:id"/>
    <xs:attribute ref="structures:ref"/>
    <xs:attribute ref="structures:uri"/>
    <xs:anyAttribute processContents="strict" namespace="##other"/>
  </xs:complexType>
  <xs:element name="AssociationAugmentationPoint" abstract="true">
    <xs:annotation>
      <xs:documentation>An augmentation point for type structures:AssociationType.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="ObjectAugmentationPoint" abstract="true">
    <xs:annotation>
      <xs:documentation>An augmentation point for type structures:ObjectType.</xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:attribute name="appliesToParent" type="xs:boolean" default="true">
    <xs:annotation>
      <xs:documentation>True if this element is a property of its parent; false if it appears only to support referencing.
</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="id" type="xs:ID">
    <xs:annotation>
      <xs:documentation>A document-relative identifier for an XML element.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="ref" type="xs:IDREF">
    <xs:annotation>
      <xs:documentation>A document-relative reference to an XML element.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="uri" type="xs:anyURI">
    <xs:annotation>
      <xs:documentation>An internationalized resource identifier or uniform resource identifier for a node or object.
</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:schema>
```

Appendix C. Index of rules

- Rule 7-55: Standard opening phrase for association class data definition.
- Rule 7-56: Standard opening phrase for class data definition.
- Rule 7-57: Namespace identifier is absolute URI.
- Rule 7-58: Namespace has a prefix.
- Rule 7-59: Namespace has version.
- Rule 7-60: Namespace has language.
- Rule 7-61: Reference namespace asserts conformance.
- Rule 7-62: Reference namespace does not have wildcard.
- Rule 7-63: Classes in reference namespace are augmentable.
- Rule 7-64: Object properties in reference namespace are referenceable.
- Rule 7-65: Reference namespace uses reference namespace components.
- Rule 7-66: Extension namespace asserts conformance.
- Rule 7-67: Classes in extension namespace are augmentable.
- Rule 7-68: Object properties in extension namespace are referenceable.
- Rule 7-69: Subset namespace asserts conformance.
- Rule 7-70: Subset has corresponding reference or extension namespace.
- Rule 7-71: Subset does not extend component range.
- Rule 7-72: Subset does not add components.
- Rule 7-73: Subset does not alter data definition.
- Rule 7-74: Augmentations declare augmenting namespace.
- Rule 8-1: Schema is CTAS-conformant.
- Rule 8-2: Document element has attribute `ct:conformanceTargets`.
- Rule 8-3: Document is a valid schema document.
- Rule 8-4: Document element is `xs:schema`.
- Rule 8-5: Prohibited schema components.
- Rule 8-6: Prohibited base types.
- Rule 8-7: Prohibited list item types.
- Rule 8-8: Prohibited union item types.
- Rule 8-9: Prohibited attribute types.
- Rule 8-10: No mixed content on complex type or complex content.
- Rule 8-11: Complex type content is explicitly simple or complex.
- Rule 8-12: Base type of complex type with complex content must have complex content.
- Rule 8-13: Untyped element is abstract.
- Rule 8-14: Element type not in the XML or XML Schema namespace.
- Rule 8-15: Element type is not simple type.
- Rule 8-16: Attribute declaration has type.
- Rule 8-17: No default or fixed value.
- Rule 8-18: Sequence has minimum and maximun cardinality 1.
- Rule 8-19: `xs:choice` must be child of `xs:sequence`.
- Rule 8-20: Choice has minimum and maximum cardinality 1.
- Rule 8-21: Comment is not recommended.
- Rule 8-22: Documentation element has no element children.
- Rule 8-23: Import has namespace.
- Rule 8-24: Import specifies local resource.
- Rule 8-25: Name of type definitions.
- Rule 8-26: Name of simple type definitions.
- Rule 8-27: Name of complex type definition.
- Rule 8-28: `xs:sequence` must be child of `xs:extension`.
- Rule 8-29: `xs:sequence` must be child of `xs:extension` or `xs:restriction`.
- Rule 8-30: Type definition is top-level.
- Rule 8-31: Complex type has a category.
- Rule 8-32: Object type with complex content is derived from `structures:ObjectType`.
- Rule 8-33: Adapter type derived from `structures:AdapterType`.
- Rule 8-34: Association type derived from `structures:AssociationType`.
- Rule 8-35: Augmentation type derived from `structures:AugmentationType`.
- Rule 8-36: Complex type with simple content has `structures:SimpleObjectAttributeGroup`.

- Rule 8-37: Base type definition defined by conformant schema.
- Rule 8-38: Component reference defined by conformant schema.
- Rule 8-39: Schema uses only known attribute groups.
- Rule 8-40: Augmentation elements are not used directly.
- Rule 8-41: List item type defined by conformant schemas.
- Rule 8-42: Union member types defined by conformant schemas.
- Rule 8-43: No literal properties in XSD.
- Rule 8-44: Declarations are top-level.
- Rule 8-45: Element type is not simple type.
- Rule 8-46: Attribute and element type is from conformant namespace.
- Rule 8-47: Element substitution group defined by conformant schema.
- Rule 8-48: Import of external schema document is labeled.
- Rule 8-49: Import of external namespace has data definition.
- Rule 8-50: Name of adapter type.
- Rule 8-51: Structure of external adapter type definition follows pattern.
- Rule 8-52: Element use from external adapter type defined by external schema documents.
- Rule 8-53: External adapter type not a base type.
- Rule 8-54: External attribute use has data definition.
- Rule 8-55: External attribute use not an ID.
- Rule 8-56: External element use has data definition.
- Rule 7-75: Proxy types.
- Rule 8-57: Proxy type has designated structure.
- Rule 8-58: Name of augmentation types.
- Rule 8-59: Name of augmentation elements.
- Rule 8-60: Name of augmentation point elements.
- Rule 8-61: Standard opening phrase for augmentation point element data definition.
- Rule 8-62: Standard opening phrase for augmentation element data definition.
- Rule 8-63: Standard opening phrase for augmentation type data definition.
- Rule 8-64: Augmentation point element corresponds to its base type.
- Rule 8-65: An augmentation point element has no type.
- Rule 8-66: An augmentation point element has no substitution group.
- Rule 8-67: Augmentation point element is only referenced by its base type.
- Rule 8-68: Augmentation point element use is optional and unbounded.
- Rule 8-69: Augmentation point element use must be last element in its base type.
- Rule 8-70: Appinfo attribute annotates schema component.
- Rule 8-71: `xs:appinfo` children are comments, elements, or whitespace.
- Rule 8-72: Appinfo child elements have namespaces.
- Rule 8-73: Appinfo descendants are not XML Schema elements.
- Rule 8-74: Component marked as deprecated is deprecated component.
- Rule 8-75: LocalTerm appinfo applies to schema.
- Rule 8-76: No simple type disallowed derivation.
- Rule 8-77: No use of "fixed" on simple type facets.
- Rule 8-78: Simple content uses extension.
- Rule 8-79: No disallowed substitutions.
- Rule 8-80: No disallowed derivation.
- Rule 8-81: Element declaration is nillable.
- Rule 8-82: No `xs:choice`.
- Rule 8-83: Element declaration is nillable.
- Rule 9-1: Namespaces are conforming or external.
- Rule 9-2: Unique namespace prefixes.
- Rule 9-3: Unique namespace identifiers.
- Rule 9-4: Composition of schema document set.
- Rule 9-5: Consistent import schema document.
- Rule 9-6: Consistent import labels.
- Rule 9-7: Consistent import documentation.
- Rule 9-8: Namespace prefix is unique.

- Rule 9-9: Schema document set must be complete.
- Rule 9-10: Use structures namespace consistent with specification.
- Rule 10-1: Message type declares initial class and property.
- Rule 10-2: Message format schema matches message type.
- Rule 12-1: NO NAME.
- Rule 12-2: NO NAME.
- Rule 12-3: NO NAME.
- Rule 12-4: NO NAME.
- Rule 12-5: NO NAME.
- Rule 12-6: NO NAME.
- Rule 12-7: NO NAME.
- Rule 12-8: NO NAME.

Appendix D. Mapping NIEM 5 rules to NIEM 6

NIEM 5 Rule	NIEM 6 Rules
Rule 4-1, Schema marked as reference schema document must conform	7-61
Rule 4-2, Schema marked as extension schema document must conform	7-66
Rule 4-3, Schema is CTAS-conformant	8-1
Rule 4-4, Document element has attribute ct:conformanceTargets	8-2
Rule 4-5, Schema claims reference schema conformance target	7-61, 7-69
Rule 4-6, Schema claims extension conformance target	7-66
Rule 5-1, structures:uri denotes resource identifier	*no matching NIEM6 rule*
Rule 7-1, Document is an XML document	8-3
Rule 7-2, Document uses XML namespaces properly	8-3
Rule 7-3, Document is a schema document	8-3
Rule 7-4, Document element is xs:schema	8-4
Rule 7-5, Component name follows ISO 11179 Part 5 Annex A	7-23
Rule 9-1, No base type in the XML namespace	8-6
Rule 9-2, No base type of xs:ID	8-6
Rule 9-3, No base type of xs:IDREF	8-6
Rule 9-4, No base type of xs:IDREFS	8-6
Rule 9-5, No base type of xs:anyType	8-6
Rule 9-6, No base type of xs:anySimpleType	8-6
Rule 9-7, No base type of xs:NOTATION	8-6
Rule 9-8, No base type of xs:ENTITY	8-6
Rule 9-9, No base type of xs:ENTITIES	8-6
Rule 9-10, Simple type definition is top-level	8-30
Rule 9-11, No simple type disallowed derivation	8-76
Rule 9-12, Simple type has data definition	7-37
Rule 9-13, No use of fixed on simple type facets	8-77
Rule 9-14, Enumeration has data definition	7-38
Rule 9-15, No list item type of xs:ID	8-7
Rule 9-16, No list item type of xs:IDREF	8-7
Rule 9-17, No list item type of xs:anySimpleType	8-7
Rule 9-18, No list item type of xs:ENTITY	8-7
Rule 9-19, No union member types of xs:ID	8-8

NIEM 5 Rule	NIEM 6 Rules
Rule 9-20, No union member types of xs:IDREF	8-8
Rule 9-21, No union member types of xs:IDREFS	8-8
Rule 9-22, No union member types of xs:anySimpleType	8-8
Rule 9-23, No union member types of xs:ENTITY	8-8
Rule 9-24, No union member types of xs:ENTITIES	8-8
Rule 9-25, Complex type definition is top-level	8-30
Rule 9-26, Complex type has data definition	7-37
Rule 9-27, No mixed content on complex type	8-10
Rule 9-28, No mixed content on complex content	8-10
Rule 9-29, Complex type content is explicitly simple or complex	8-11
Rule 9-30, Complex content uses extension	*no matching NIEM6 rule*
Rule 9-31, Base type of complex type with complex content must have complex content	8-12
Rule 9-32, Base type of complex type with complex content must have complex content	8-12
Rule 9-33, Simple content uses extension	8-78
Rule 9-34, No complex type disallowed substitutions	8-79
Rule 9-35, No complex type disallowed derivation	8-80
Rule 9-36, Element declaration is top-level	8-44
Rule 9-37, Element declaration has data definition	7-37
Rule 9-38, Untyped element is abstract	8-13
Rule 9-39, Element of type xs:anySimpleType is abstract	8-13
Rule 9-40, Element type not in the XML Schema namespace	8-14
Rule 9-41, Element type not in the XML namespace	8-14
Rule 9-42, Element type is not simple type	8-15, 8-45
Rule 9-43, No element disallowed substitutions	8-79
Rule 9-44, No element disallowed derivation	8-80
Rule 9-45, No element default value	8-17
Rule 9-46, No element fixed value	8-17
Rule 9-47, Element declaration is nillable	8-81, 8-83
Rule 9-48, Attribute declaration is top-level	8-44
Rule 9-49, Attribute declaration has data definition	7-37
Rule 9-50, Attribute declaration has type	8-16
Rule 9-51, No attribute type of xs:ID	8-9
Rule 9-52, No attribute type of xs:IDREF	8-9

NIEM 5 Rule	NIEM 6 Rules
Rule 9-53, No attribute type of xs:IDREFS	8-9
Rule 9-54, No attribute type of xs:ENTITY	8-9
Rule 9-55, No attribute type of xs:ENTITIES	8-9
Rule 9-56, No attribute type of xs:anySimpleType	8-9
Rule 9-57, No attribute default values	8-17
Rule 9-58, No fixed values for optional attributes	8-17
Rule 9-59, No use of element xs:notation	8-5
Rule 9-60, Model group does not affect meaning	*no matching NIEM6 rule*
Rule 9-61, No xs:all	8-5
Rule 9-62, xs:sequence must be child of xs:extension	8-28
Rule 9-63, xs:sequence must be child of xs:extension or xs:restriction	8-29
Rule 9-64, No xs:choice	8-82
Rule 9-65, xs:choice must be child of xs:sequence	8-19
Rule 9-66, Sequence has minimum cardinality 1	8-18
Rule 9-67, Sequence has maximum cardinality 1	8-18
Rule 9-68, Choice has minimum cardinality 1	8-20
Rule 9-69, Choice has maximum cardinality 1	8-20
Rule 9-70, No use of xs:any	7-62
Rule 9-71, No use of xs:anyAttribute	7-62
Rule 9-72, No use of xs:unique	8-5
Rule 9-73, No use of xs:key	8-5
Rule 9-74, No use of xs:keyref	8-5
Rule 9-75, No use of xs:group	8-5
Rule 9-76, No definition of attribute groups	8-5
Rule 9-77, Comment is not recommended	8-21
Rule 9-78, Documentation element has no element children	8-22
Rule 9-79, xs:appinfo children are comments, elements, or whitespace	8-71
Rule 9-80, Appinfo child elements have namespaces	8-72
Rule 9-81, Appinfo descendants are not XML Schema elements	8-73
Rule 9-82, Schema has data definition	7-36
Rule 9-83, Schema document defines target namespace	7-57
Rule 9-84, Target namespace is absolute URI	7-57
Rule 9-85, Schema has version	7-59

NIEM 5 Rule	NIEM 6 Rules
Rule 9-86, No disallowed substitutions	8-79
Rule 9-87, No disallowed derivations	8-80
Rule 9-88, No use of xs:redefine	8-5
Rule 9-89, No use of xs:include	8-5
Rule 9-90, xs:import must have namespace	8-23
Rule 9-91, XML Schema document set must be complete	9-9
Rule 9-92, Namespace referenced by attribute type is imported	*no matching NIEM6 rule*
Rule 9-93, Namespace referenced by attribute base is imported	*no matching NIEM6 rule*
Rule 9-94, Namespace referenced by attribute itemType is imported	*no matching NIEM6 rule*
Rule 9-95, Namespaces referenced by attribute memberTypes is imported	*no matching NIEM6 rule*
Rule 9-96, Namespace referenced by attribute ref is imported	*no matching NIEM6 rule*
Rule 9-97, Namespace referenced by attribute substitutionGroup is imported	*no matching NIEM6 rule*
Rule 10-1, Complex type has a category	8-31
Rule 10-2, Object type with complex content is derived from structures:ObjectType	8-32
Rule 10-3, RoleOf element type is an object type	*no matching NIEM6 rule*
Rule 10-4, Only object type has RoleOf element	*no matching NIEM6 rule*
Rule 10-5, RoleOf elements indicate the base types of a role type	*no matching NIEM6 rule*
Rule 10-6, Instance of RoleOf element indicates a role object	*no matching NIEM6 rule*
Rule 10-7, Import of external namespace has data definition	8-49
Rule 10-8, External adapter type has indicator	8-50
Rule 10-9, Structure of external adapter type definition follows pattern	8-51
Rule 10-10, Element use from external adapter type defined by external schema documents	8-52
Rule 10-11, External adapter type not a base type	8-53
Rule 10-12, External adapter type not a base type	8-53
Rule 10-13, External attribute use only in external adapter type	*no matching NIEM6 rule*
Rule 10-14, External attribute use has data definition	8-54
Rule 10-15, External attribute use not an ID	8-55

NIEM 5 Rule	NIEM 6 Rules
Rule 10-16, External element use has data definition	8-56
Rule 10-17, Name of code type ends in CodeType	7-5, 7-8
Rule 10-18, Code type corresponds to a code list	7-5, 7-8
Rule 10-19, Element of code type has code representation term	7-11
Rule 10-20, Proxy type has designated structure	7-75, 8-57
Rule 10-21, Association type derived from structures:AssociationType	7-4, 8-34
Rule 10-22, Association element type is an association type	7-10
Rule 10-23, Augmentable type has augmentation point element	7-63, 7-67
Rule 10-24, Augmentable type has at most one augmentation point element	7-63, 7-67
Rule 10-25, Augmentation point element corresponds to its base type	8-64
Rule 10-26, An augmentation point element has no type	8-65
Rule 10-27, An augmentation point element has no substitution group	8-66
Rule 10-28, Augmentation point element is only referenced by its base type	8-67
Rule 10-29, Augmentation point element use is optional	8-68
Rule 10-30, Augmentation point element use is unbounded	8-68
Rule 10-31, Augmentation point element use must be last element in its base type	8-69
Rule 10-32, Element within instance of augmentation type modifies base	*no matching NIEM6 rule*
Rule 10-33, Only an augmentation type name ends in AugmentationType	8-58
Rule 10-34, Schema component with name ending in AugmentationType is an augmentation type	8-58
Rule 10-35, Type derived from structures:AugmentationType is an augmentation type	8-35
Rule 10-36, Augmentation element type is an augmentation type	8-59
Rule 10-37, Augmentation elements are not used directly	8-40
Rule 10-38, Metadata type has data about data	*no matching NIEM6 rule*
Rule 10-39, Metadata types are derived from structures:MetadataType	*no matching NIEM6 rule*
Rule 10-40, Metadata element declaration type is a metadata type	*no matching NIEM6 rule*
Rule 10-41, Metadata element has applicable elements	*no matching NIEM6 rule*
Rule 10-42, Name of element that ends in Representation is abstract	7-9
Rule 10-43, A substitution for a representation element declaration is a value for a type	*no matching NIEM6 rule*
Rule 10-44, Schema component name composed of English words	7-14

NIEM 5 Rule	NIEM 6 Rules
Rule 10-45, Schema component name has xml:lang	7-60
Rule 10-46, Schema component names have only specific characters	7-15
Rule 10-47, Punctuation in component name is a separator	7-19
Rule 10-48, Names use camel case	7-16
Rule 10-49, Attribute name begins with lower case letter	7-17
Rule 10-50, Name of schema component other than attribute and proxy type begins with upper case letter	7-18
Rule 10-51, Names use common abbreviations	7-33
Rule 10-52, Local term declaration is local to its schema document	7-34
Rule 10-53, Local terminology interpretation	*no matching NIEM6 rule*
Rule 10-54, Singular form is preferred in name	7-20
Rule 10-55, Present tense is preferred in name	7-21
Rule 10-56, Name does not have nonessential words	7-22
Rule 10-57, Element or attribute name follows pattern	7-23
Rule 10-58, Object-class term identifies concrete category	7-24
Rule 10-59, Property term describes characteristic or subpart	7-25
Rule 10-60, Name may have multiple qualifier terms	7-26
Rule 10-61, Name has minimum necessary number of qualifier terms	7-27
Rule 10-62, Order of qualifiers is not significant	7-28
Rule 10-63, Redundant term in name is omitted	7-29
Rule 10-64, Element with simple content has representation term	7-30
Rule 10-65, Element with complex content has representation term when appropriate	7-31
Rule 10-66, Element with complex content has representation term only when appropriate	7-32
Rule 10-67, Machine-readable annotations are valid	*no matching NIEM6 rule*
Rule 10-68, Component marked as deprecated is deprecated component	8-74
Rule 10-69, Deprecated annotates schema component	8-70
Rule 10-70, External import indicator annotates import	*no matching NIEM6 rule*
Rule 10-71, External adapter type indicator annotates complex type	*no matching NIEM6 rule*
Rule 10-72, appinfo:appliesToTypes annotates metadata element	*no matching NIEM6 rule*
Rule 10-73, appinfo:appliesToTypes references types	*no matching NIEM6 rule*

NIEM 5 Rule	NIEM 6 Rules
Rule 10-74, appinfo:appliesToElements annotates metadata element	*no matching NIEM6 rule*
Rule 10-75, appinfo:appliesToElements references elements	*no matching NIEM6 rule*
Rule 10-76, appinfo:LocalTerm annotates schema	8-75
Rule 10-77, appinfo:LocalTerm has literal or definition	7-35
Rule 10-78, Use structures consistent with specification	9-10
Rule 11-1, Name of type ends in Type	7-1
Rule 11-2, Only types have name ending in Type or SimpleType	7-1, 8-25
Rule 11-3, Base type definition defined by conformant schema	8-37
Rule 11-4, Name of simple type ends in SimpleType	8-26
Rule 11-5, Use lists only when data is uniform	*no matching NIEM6 rule*
Rule 11-6, List item type defined by conformant schemas	8-41
Rule 11-7, Union member types defined by conformant schemas	8-42
Rule 11-8, Name of a code simple type ends in CodeSimpleType	7-7
Rule 11-9, Code simple type corresponds to a code list	7-7
Rule 11-10, Attribute of code simple type has code representation term	7-11
Rule 11-11, Complex type with simple content has structures:SimpleObjectAttributeGroup	8-36
Rule 11-12, Element type does not have a simple type name	8-45
Rule 11-13, Element type is from conformant namespace	8-46
Rule 11-14, Name of element that ends in Abstract is abstract	7-9
Rule 11-15, Name of element declaration with simple content has representation term	7-30
Rule 11-16, Name of element declaration with simple content has representation term	7-30
Rule 11-17, Element substitution group defined by conformant schema	8-47
Rule 11-18, Attribute type defined by conformant schema	8-46
Rule 11-19, Attribute name uses representation term	7-30
Rule 11-20, Element or attribute declaration introduced only once into a type	*no matching NIEM6 rule*
Rule 11-21, Element reference defined by conformant schema	8-38
Rule 11-22, Referenced attribute defined by conformant schemas	8-38
Rule 11-23, Schema uses only known attribute groups	8-39
Rule 11-24, Data definition does not introduce ambiguity	7-40
Rule 11-25, Object class has only one meaning	7-41
Rule 11-26, Data definition of a part does not redefine the whole	7-42

NIEM 5 Rule	NIEM 6 Rules
Rule 11-27, Do not leak representation into data definition	7-43
Rule 11-28, Data definition follows 11179-4 requirements	7-44
Rule 11-29, Data definition follows 11179-4 recommendations	7-45
Rule 11-30, xs:documentation has xml:lang	7-60
Rule 11-31, Standard opening phrase for augmentation point element data definition	8-61
Rule 11-32, Standard opening phrase for augmentation element data definition	8-62
Rule 11-33, Standard opening phrase for metadata element data definition	*no matching NIEM6 rule*
Rule 11-34, Standard opening phrase for association element data definition	7-47
Rule 11-35, Standard opening phrase for abstract element data definition	7-46
Rule 11-36, Standard opening phrase for date element or attribute data definition	7-48
Rule 11-37, Standard opening phrase for quantity element or attribute data definition	7-49
Rule 11-38, Standard opening phrase for picture element or attribute data definition	7-50
Rule 11-39, Standard opening phrase for indicator element or attribute data definition	7-51
Rule 11-40, Standard opening phrase for identification element or attribute data definition	7-52
Rule 11-41, Standard opening phrase for name element or attribute data definition	7-53
Rule 11-42, Standard opening phrase for element or attribute data definition	7-54
Rule 11-43, Standard opening phrase for association type data definition	7-55
Rule 11-44, Standard opening phrase for augmentation type data definition	8-63
Rule 11-45, Standard opening phrase for metadata type data definition	*no matching NIEM6 rule*
Rule 11-46, Standard opening phrase for complex type data definition	7-56
Rule 11-47, Standard opening phrase for simple type data definition	7-56
Rule 11-48, Same namespace means same components	*no matching NIEM6 rule*
Rule 11-49, Different version means different view	*no matching NIEM6 rule*
Rule 11-50, Reference schema document imports reference schema document	7-65
Rule 11-51, Extension schema document imports reference or extension schema document	*no matching NIEM6 rule*
Rule 11-52, Structures imported as conformant	*no matching NIEM6 rule*
Rule 11-53, XML namespace imported as conformant	*no matching NIEM6 rule*
Rule 11-54, Each namespace may have only a single root schema in a schema set	9-5
Rule 11-55, Consistently marked namespace imports	9-6

NIEM 5 Rule	NIEM 6 Rules
Rule 12-1, Instance must be schema-valid	*no matching NIEM6 rule*
Rule 12-2, Empty content has no meaning	*no matching NIEM6 rule*
Rule 12-3, Element has only one resource identifying attribute	12-1
Rule 12-4, Attribute structures:ref must reference structures:id	12-2
Rule 12-5, Linked elements have same validation root	12-3
Rule 12-6, Attribute structures:ref references element of correct type	12-4
Rule 12-7, structures:uri denotes resource identifier	12-5
Rule 12-8, structures:id and structures:ref denote resource identifier	12-7
Rule 12-9, Nested elements and references have the same meaning.	12-8
Rule 12-10, Order of properties is expressed via structures:sequenceID	*no matching NIEM6 rule*
Rule 12-11, Metadata applies to referring entity	*no matching NIEM6 rule*
Rule 12-12, Referent of structures:relationshipMetadata annotates relationship	*no matching NIEM6 rule*
Rule 12-13, Values of structures:metadata refer to values of structures:id	*no matching NIEM6 rule*
Rule 12-14, Values of structures:relationshipMetadata refer to values of structures:id	*no matching NIEM6 rule*
Rule 12-15, structures:metadata and structures:relationshipMetadata refer to metadata elements	*no matching NIEM6 rule*
Rule 12-16, Attribute structures:metadata references metadata element	*no matching NIEM6 rule*
Rule 12-17, Attribute structures:relationshipMetadata references metadata element	*no matching NIEM6 rule*
Rule 12-18, Metadata is applicable to element	*no matching NIEM6 rule*

Appendix E. Index of figures

Appendix W. Safety, Security and Privacy Considerations

(Note: OASIS strongly recommends that Open Projects consider issues that might affect safety, security, privacy, and/or data protection in implementations of their specification and document them for implementers and adopters. For some purposes, you may find it required, e.g. if you apply for IANA registration.

While it may not be immediately obvious how your specification might make systems vulnerable to attack, most specifications, because they involve communications between systems, message formats, or system settings, open potential channels for exploit. For example, IETF [RFC3552] lists “eavesdropping, replay, message insertion, deletion, modification, and man-in-the-middle” as well as potential denial of service attacks as threats that must be considered and, if appropriate, addressed in IETF RFCs.

In addition to considering and describing foreseeable risks, this section should include guidance on how implementers and adopters can protect against these risks.

We encourage editors and OP members concerned with this subject to read *Guidelines for Writing RFC Text on Security Considerations*, IETF [RFC3552], for more information.

Remove this note before submitting for publication.)

Appendix X. Acknowledgments

```
(Note: A Work Product approved by the OP should include a list of people who participated in the
development of the Work Product. This is generally done by collecting the list of names in this appendix.
This list should be initially compiled by the Chair, and any Member of the OP may add or remove their
names from the list by request. Remove this note before submitting for publication.)
```

X.1 Special Thanks

(This is an optional subsection to call out contributions from OP members. If a OP wants to thank non-OP members then they should avoid using the term "contribution" and instead thank them for their "expertise" or "assistance".)

Substantial contributions to this document from the following individuals are gratefully acknowledged:

Participant Name, Affiliation or "Individual Member"

x.2 Participants

(An OP can determine who they list here. It is common practice for OPs to list everyone that was part of the OP during the creation of the document, but this is ultimately an OP decision on who they want to list and not list.)

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Project-name OP Members:

First Name	Last Name	Company
Aubrey	Beach	JS J6
Brad	Bollinger	Ernst & Young
James	Cabral	Individual
Tom	Carlson	GTRI
Chuck	Chipman	GTRI
Mike	Douklias	JS J6
Katherine	Escobar	JS J6
Lavdjola	Farrington	JS J6
Dave	Hardy	JS J6
Mike	Hulme	Unisys
Eric	Jahn	Alexandria Consulting
Dave	Kemp	NSA
Vamsi	Kondannagari	Integral Fed
Shunda	Louis	JS J6
Peter	Madruga	GTRI
Christina	Medlin	GTRI
Joe	Mierwa	Mission Critical Partners
April	Mitchell	FBI
Carl	Nelson	RISS
Scott	Renner	MITRE

First Name	Last Name	Company
Beth	Smalley	JS J6
Duncan	Sparrell	sFractal
Jennifer	Stathakis	NIST
Stephen	Sullivan	JS J6
Josh	Wilson	FBI

Appendix Y. Revision History

Revisions made since the initial stage of this numbered Version of this document may be tracked here.

If revision tracking is handled in another system like github, provide a link to it instead of using this table, if desired.

Revision	Date	Editor	Changes Made
	2024-02-26	Scott Renner	Attempts to be a fairly complete outline of NDR6.
	2024-04-10	Scott Renner	Includes sections 1 and 2. The plan now is to keep filling out this outline until enough sections are complete. Then we will remove the comment stuff and apply the OASIS template.
	2024-04-15	Scott Renner	Revised section 3
	2024-04-24	Scott Renner	New section 4, Section 3 comments from incorporated.
	2024-05-04	Scott Renner	Section 3: better description of message spec, type, format; new figure 3-3, Section 4: comments, new examples and figures; flattened outline
	2024-05-20	Scott Renner	Nearly complete section 5 -- still need section on augmentations
	2024-06-03	Scott Renner	Combined section 4 and 5 into new section 4: "Data models in NIEM" -- metamodel, CMF, and XSD, Outlined section 5: "Modeling rules for NIEM XSD", Moved a lot of XSD stuff from section 3 into the new section 5, Comments incorporated
	2024-06-24	Scott Renner	Incorporated comments, Augmentation section is complete
	2024-07-01	Scott Renner	New subsections in section 2, New text in section 3.6 (canonical representation in XSD and CMF), Lots of new stuff in section 5.1
	2024-07-08	Scott Renner	Conformance is a major section (new section 5), Conformance targets now apply to namespaces, not schema documents, Section 6 is now "Rules for namespaces and models"
	2024-07-10	Scott Renner	Naming rules (section 6) complete
	2024-07-15	Scott Renner	Rules for documentation, extension schemas and models complete, new placeholder section 5 (data modeling patterns)
	2024-07-19	Scott Renner	All NDR 5 schema rules handled, NDR5 to NDR6 crossref appendix added, comments incorporated

Appendix Z. Notices