NIEM Naming and Design Rules (NDR) Version 6.0

Project Specification Draft 01

27 January 2025

This stage:

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 (Authoritative)

Previous stage:

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 (Authoritative)

Latest stage:

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 (Authoritative)

Open Project:

OASIS NIEMOpen OP

Project Chair:

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

NTAC Technical Steering Committee Chairs:

Brad Bolliger (brad.bolliger@ey.com), EY

James Cabral (jim@cabral.org), Individual

Scott Renner (sar@mitre.org), MITRE

Editors:

James Cabral (jim@cabral.org), Individual

Tom Carlson (Thomas.Carlson@gtri.gatech.edu), Georgia Tech Research Institute

Scott Renner (sar@mitre.org), MITRE

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.
- Conformance Targets Attribute Specification (CTAS) Version 3.0. Edited by Tom Carlson. 22 February 2023. OASIS Project Specification 01. https://docs.oasis-open.org/niemopen/ctas/v3.0/ps01/ctas-v3.0-ps01.html. Latest stage: https://docs.oasis-open.org/niemopen/ctas/v3.0/ctas-v3.0.html.

Abstract:

Work in progress.

Status:

This document was last revised or approved by the Project Governing Board of the OASIS NIEMOpen OP on the above date. The level of approval is also listed above. Check the “Latest stage” location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Open Project (OP) are listed at http://www.niemopen.org/.

Comments on this work can be provided by opening issues in the project repository or by sending email to the project’s public comment list: niemopen@lists.oasis-open-projects.org. List information is available at https://lists.oasis-open-projects.org/g/niemopen.

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product’s prose narrative document(s), the content in the separate plain text file prevails.

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 [RFC 2119] and [RFC 8174] 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 Scott Renner. 1 January 2025. 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

---

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 Glossary

1.1.1 Definitions of terms

Term	Definition
Absolute URI	A Uniform Resource Identifier (URI) with scheme, hierarchical part, and optional query, but without a fragment; a URI matching the grammar syntax `<absoluteURI>` as defined by [RFC 3986].
Adapter class	A class that contains only properties from a single external namespace. [*see §4.4*]
Adapter type	An XSD type definition that encapsulates external components for use within NIEM. *(see §9.4)*
Appinfo namespace	A namespace defined by a schema document that provides additional semantics for components in the XSD representation of a model. *(see §9.7)*
Association class	A class that represents a specific relationship between objects. *(see §4.4)*
Attribute property	A data property represented in XSD as an attribute declaration. *(see §4.8)*
Augmentation	The means by which a designer of one namespace adds properties to a class defined in a different namespace. *(see §3.7, §4.15)*
Augmentation element	An element in an XML message that is a container for one or more augmentation properties. *(see §4.15.2)*
Augmentation point element	An abstract element declaration that provides a place for augmentation properties within the XSD representation of an augmented class. *(see §4.15.2)*

Term	Definition
Augmentation property	A property added by one namespace to an augmented class in another namespace.*(see §4.15)*
Augmentation type	An XSD type definition for an augmentation element. *(see §4.15.2)*
Cardinality	The number of times a property may/must appear in an object.
Class	A definition of an entity in a model; that is, a real-world object, concept, or thing*(see §3.4, §4.4)*
Code list datatype	A datatype in which each valid value is also a string in a code list. *(see §4.12)*
Code list	A set of string values, each having a known meaning beyond its value, each representing a distinct conceptual entity. *(see §4.12)*
Conforming namespace	A namespace that satisfies all of the applicable rules in this document; a reference namespace, extension namespace, or subset namespace. *(see §6.1)*
Conforming schema document	A schema document that satisfies all of the applicable rules in this document.*(see §6.1)*
Conforming schema document set	A schema document set that satisfies all of the applicable rules in this document.*(see §6.1)*
Data definition	A text definition of a 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 literal value in a message.
Documented component	A CMF object or XSD schema component that has an associated data definition.
Element property	An object property, or a data property that is not an attribute property; represented in XSD by an element declaration. *(see §4.8)*
Extension namespace	A namespace defining components that are intended for reuse, but within a more narrow scope than those defined in a reference namespace. *(see §3.6)*
Extension schema document	A schema document that is the XSD representation of an extension namespace.
External attribute	An attribute declaration in external schema document.
External component	A component defined by an external schema document. *(see §9.4)*
External namespace	Any namespace defined by a schema document that is not a conforming namespace, the structures namespace, or the XML namespace http://www.w3.org/XML/1998/namespace. *(see §3.6)*

Term	Definition
External schema document	A schema document that defines an external namespace. *(see §3.6)*
Literal class	A class that contains no object properties, one or more attribute properties, and exactly one element property. *(see §4.4)*
Literal property	The element property in a literal class.
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 namespace, or has a word sense that is in some way unclear. *(see §4.16)*
Message	A package of data shared at runtime; a sequence of bits that convey information to be exchanged or shared; an instance of a message type. *(see §3.1.1)*
Message designer	A person who creates a message type and message format from an information requirement, so that an instance message at runtime will contain all the facts that need to be conveyed.
Message developer	A person who 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 that conform to a message type. *(see §3.1.2)*
Message model	A data model intended to precisely define the mandatory and optional content of messages and the meaning of that content. *(see §3.1.3)*
Message object	The initial object in a message.
Message specification	A collection of related message formats and message types. *(see §3.1.4)*
Message type	A specification of the information content of messages. *(see §3.1.3)*
Model file	The CMF representation of a NIEM model; a message that conforms to the CMF message type. *(see §3.5, §6.1)*
Namespace	A collection of uniquely-named components, managed by an authoritative source. *(see §3.6)*
NCName	A non-colonized name, matching the grammar syntax `<NCName>` as defined by [XML Namespaces].
Object class	Represents a class of objects defined by a NIEM model. *(see §4.4)*
Proxy type	An XSD complex type definition with simple content that extends one of the simple types in the XML Schema namespace with `structures:SimpleObjectAttributeGroup`. *(see §9.5)*
Relationship property	A property that provides information about the relationship between its parent and grandparent objects. *(see §4.6, §5.5)*
Reference attribute property	An attribute property that contains a reference to an object in a message. *(see §4.8)*

Term	Definition
Reference namespace	A namespace containing components that are intended for the widest possible reuse.*(see §3.6)*
Reference schema document	The XSD representation of a reference namespace. *(see §9.8)*
Reuse model	A data model entirely comprised of reference namespaces and extension namespaces; a model intended to make the agreed definitions of a community available for reuse.
Schema	An artifact that can be used to assess the validity of a message; in XML Schema for XML messages, JSON Schema for JSON messages. *(see §3.1.2)*
Schema document set	A collection of schema documents that together are capable of validating an XML document.*(see §10.2)*
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.
Structures namespace	A namespace that provides base types and attributes for the XSD representation of NIEM models.*(see §3.6)*
Subset namespace	A subset of the components in a reference or extension namespace.*(see §3.6)*
Subset rule	Any data that is valid for a subset namespace must also be valid for its reference namespace or extension namespace, and must have the same meaning.*(see §8.4)*
Subset schema document	A schema document for a subset namespace. *(see §9.10)*

Terms imported from *Extensible Markup Language (XML) 1.0 (Fourth Edition)* [XML]:

Term	Definition
Document element	An element, no part of which appears in the content of another element; preferred synonym for *root element*.
XML document	A data object is an XML document if it is well-formed, as defined in this specification. (Section 2, Documents)

Terms imported from *XML Information Set (Second Edition)* [XML Infoset]:

Term	Definition
Attribute	An *attribute information item*, as defined by Section 2.3: *Attribute Information Items*.
Element	An *element information item*, as defined by Section 2.2, *Element Information Items*.

Terms imported from [XML Schema Structures]:

Term	Definition

Term	Definition
Attribute declaration	As defined by *Section 2.2.2.3, Attribute Declaration*.
Base type definition	A type definition used as the basis for an extension or restriction.*(see Section 2.2.1.1, Type Definition Hierarchy)*
Complex type definition	As defined by *Section 2.2.1.3, Complex Type Definition**.
Element declaration	As defined by *Section 2.2.2.1, Element Declaration*.
Schema component	The generic term for the building blocks that comprise the abstract data model of the schema*(see Section 2.2, XML Schema Abstract Data Model)*
Schema document	As defined by *Section 3.1.2, XML Representations of Components*, which states, “A document in this form (i.e. a element information item) is a schema document.”
Simple type definition	As defined by *Section 2.2.1.2, Simple Type Definition*.
Valid	As defined by *Section 2.1, Overview of XML Schema*, which states, “The word valid and its derivatives are used to refer to clause 1 above, the determination of local schema-validity.”
XML Schema	A set of schema components. *(see Section 2.2, XML Schema Abstract Data Model)*
XML Schema definition language (XSD)	As defined by *Abstract*, which states, “XML Schema: Structures specifies the XML Schema definition language, which offers facilities for describing the structure and constraining the contents of XML 1.0 documents, including those which exploit the XML Namespace facility.”

Terms imported from NIEM Conformance Targets Attribute Specification [CTAS-v3.0]:

Term	Definition
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. *(see §6.1)*
Conformance target identifier	An internationalized resource identifier (IRI) that uniquely identifies aconformance target.
Effective conformance targets attribute	The first occurrence of the attribute {https://docs.oasis-open.org/niemopen/ns/specification/conformanceTargets/6.0/}conformanceTargets, in document order.
Effective conformance target identifier	An internationalized resource identifier reference that occurs in the document’seffective conformance targets attribute.

1.1.2 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
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
NBAC	NIEMOpen Business Architecture Committee
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
RFC	Request For Comments
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
XML	Extensible Markup Language
XSD	XML Schema Definition

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.

This document is not intended as a user guide. Training materials for message designers and developers will be available at www.niemopen.org.

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 activities.

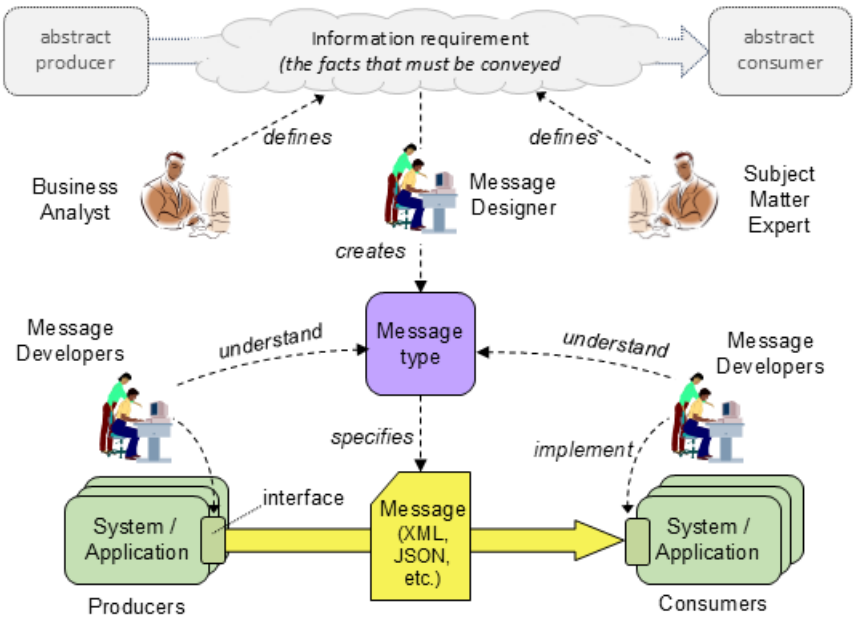


Figure 2-1: User roles and activities

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 amessage 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	

Section	Manager	Business Analyst	Message Designer	Message Developer
6. Conformance		x	x	x
7. Rules for model components		x	x	
8. Rules for namespaces		x	x	
9. Rules for schema documents			x	
10. Rules for models			x	
11. Rules for message types and message formats			x	x
12. Rules for XML messages			x	x
13. Rules for JSON messages			x	x
14. RDF interpretation of NIEM models and messages			x	

Table 2-2: Relevant document sections by user role

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 A, 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 this section:

- *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 of a message 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

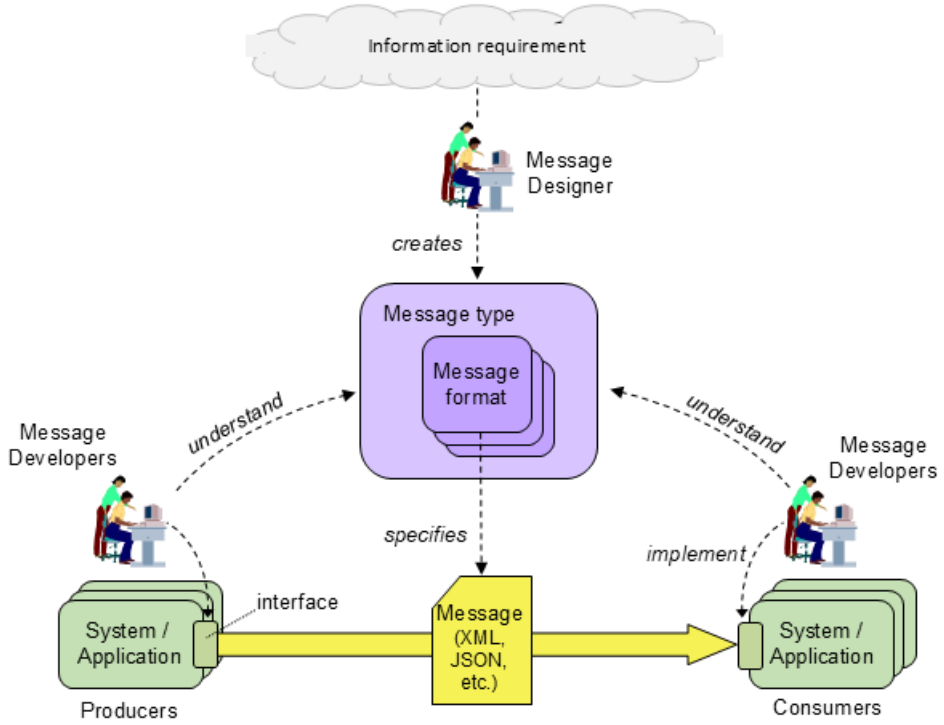


Figure 3-1: Message types, message formats, and messages

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 implement 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 represents the information content of the message. Example 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, long lines may be truncated, and some portions omitted and/or replaced with ellipses (…).)

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

<
Example 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 called the message object. For example, the message object in example 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 object in the serialized graph to point to an object elsewhere in the graph. This mechanism supports cycles and avoids duplication in the graph data structure. (See section 5.2.)

Every message is an instance of a message format. A conforming message must satisfy the rules in section 12 and section 13. 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. Example 3-3 shows a portion of the schemas for the two example messages in example 3-2.

```
<xs:complexType name="RequestType">
  <xs:sequence>
    <xs:element ref="msg:RequestID"/>
    <xs:element ref="msg:RequestedItem"/>
  </xs:sequence>
</xs:complexType>
<xs:element name="Request" type="msg:RequestType"/>
```

```
{
  "msg:RequestType": {
    "type": "object",
    "properties": {
      "msg:RequestID": {"$ref": "#/properties/msg:RequestID"},
      "msg:RequestedItem": {"$ref": "#/properties/msg:RequestedIt
    },
    "required": [
      "msg:RequestID",
      "msg:RequestedItem"
    ]
  },
  "msg:Request": {
    "$ref": "#/definitions/msg:RequestType"
  }
}
```

Example 3-3: Example 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, Java Architecture for XML Binding (JAXB).

A message format belongs to exactly one message type. A conforming message format must satisfy the rules in section 11; 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 XML message and the JSON message in example 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 is the means through which the message designer precisely defines the mandatory and optional content of conforming messages and the meaning of that content. This model is expressed in either of NIEM's two model representations, which are described in section 3.4 and section 3.5, and fully defined in section 4. Example 3-4 shows a portion of the message model for the two message formats in example 3-3.

```
<xs:complexType name="ItemType" appinfo:referenceCode="NONE">
  <xs:annotation>
    <xs:documentation>A data type for an article or thing.
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      <xs:sequence>
        <xs:element ref="nc:ItemName"/>
        <xs:element ref="nc:ItemQuantity"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="ItemName" type="nc:TextType">
  <xs:annotation>
    <xs:documentation>A name of an item.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="RequestedItem" type="nc:ItemType">
  <xs:annotation>
    <xs:documentation>A specification of an item request.</xs
  </xs:annotation>
</xs:element>
```

```
<Class structures:id="nc.ItemType">
  <Name>ItemType</Name>
  <Namespace structures:ref="nc" xsi:nil="true"/>
  <DocumentationText>A data type for an article or th
  <ReferenceCode>NONE</ReferenceCode>
  <ChildPropertyAssociation>
    <DataProperty structures:ref="nc.ItemName" xsi:nil="tr
    <MinOccursQuantity>1</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
  </ChildPropertyAssociation>
  <ChildPropertyAssociation>
    <DataProperty structures:ref="nc.ItemQuantity"
    <MinOccursQuantity>1</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
  </ChildPropertyAssociation>
</Class>
<DataProperty structures:id="nc.ItemName">
  <Name>ItemName</Name>
  <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>
```

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

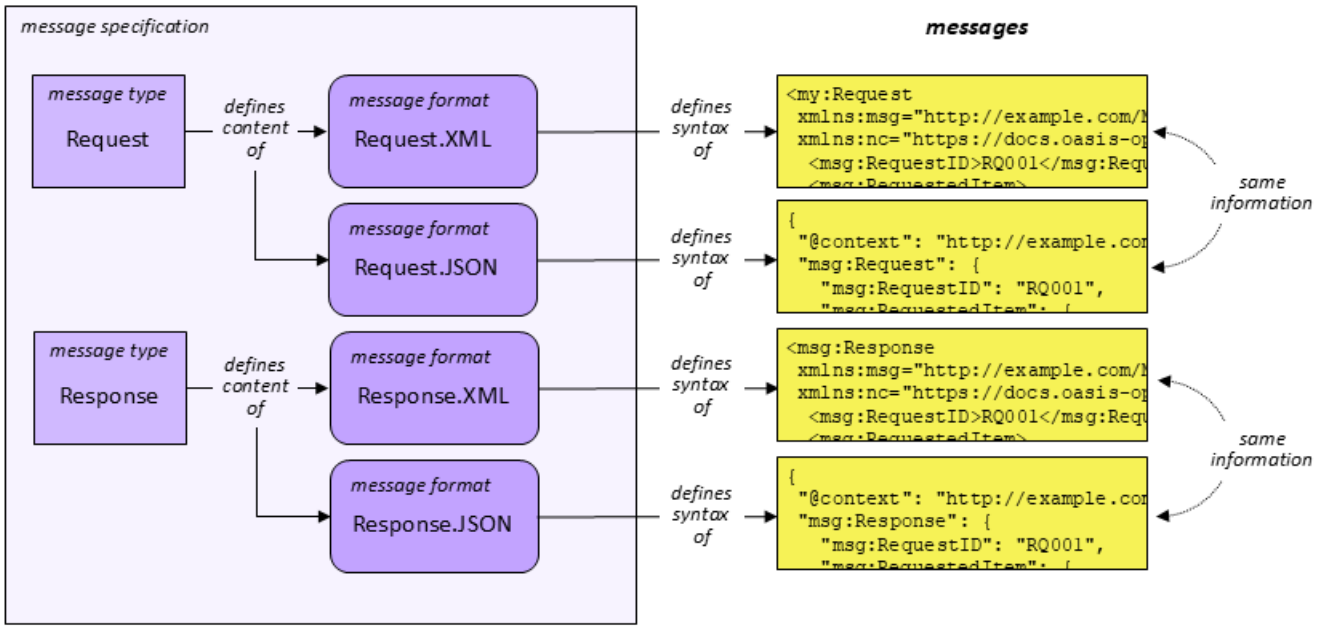
In addition to the message model, a message type also declares the initial property of conforming messages. In a conforming message, the message object is always the value of the initial property. For example, the message type for the message in example 3-2 declares that the initial property is `msg:Request`.

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. (Message designers may also compose these schemas by hand, if desired.)

A conforming message type must satisfy all of the rules in section 11.

3.1.4 Message specification

A message specification is a collection of related message types. For instance, a Request message type might be paired with a Response message type 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 example 3-5.



Example 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
- 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 allowing 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.

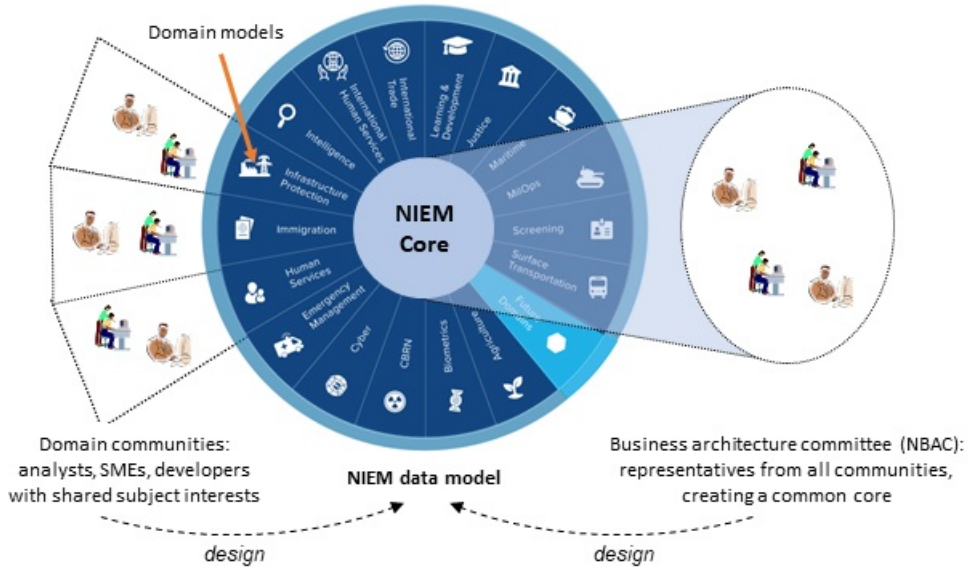


Figure 3-6: NIEM communities and data models

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

NIEM's policy of easy model extension supports easy reuse of community data models. Because a community model does not need to be complete for the union of all needs, each community may focus its effort on its common needs, where the effort of agreement has the highest value. Data definitions that are not common, that are needed only for a particular message appear only as extensions in that message type, and need be learned only by the message developers who implement it. Model extensions are further described in section 3.7.

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 — conventions for data element names and documentation

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.

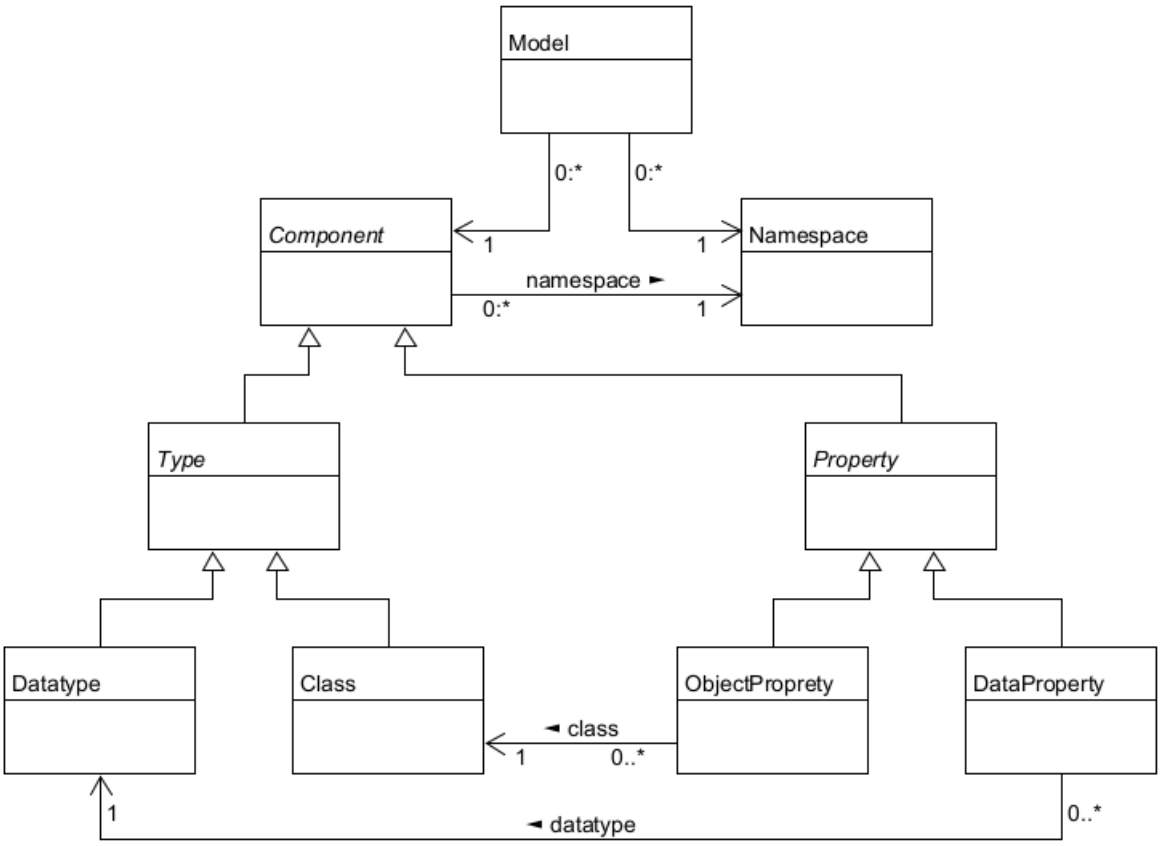


Figure 3-7: High-level view of the NIEM metamodel

- 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 example 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 objects. A *data property* defines a relationship between an object and a literal value. In example 3-4, `nc:ItemType` is the name of a class.

- A *datatype* defines the allowed values of a corresponding *literal value* in a message. In example 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.

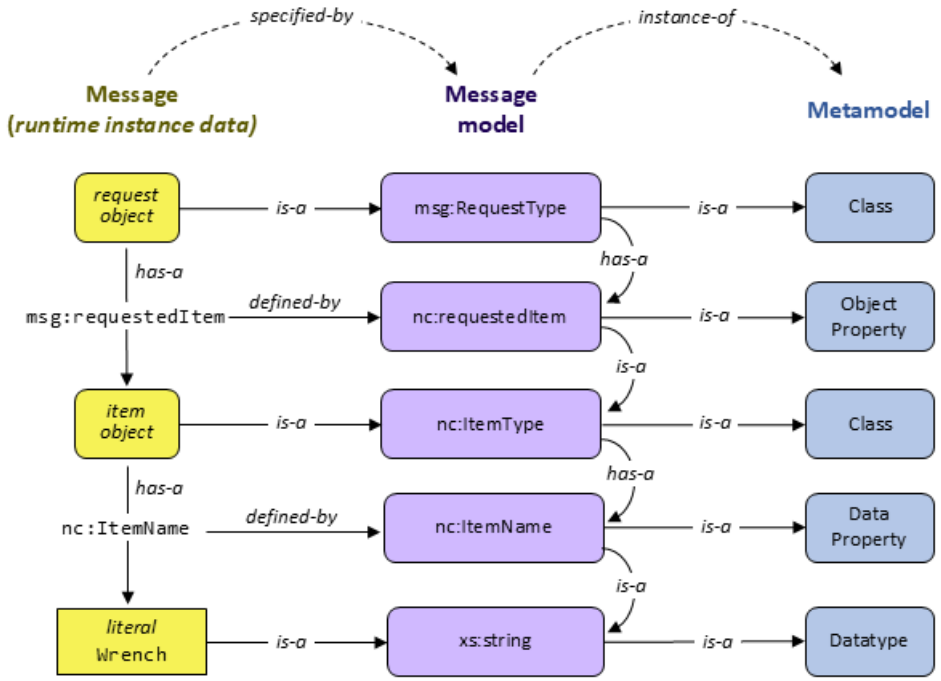


Figure 3-8: Message, message model, and metamodel relationships

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 conformingmessage and also defines the meaning of that content. For example, 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 software tools.)

Every version of NIEM uses 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. For this reason, NIEM 6 introduces the Common Model Format (CMF), which is 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 part of a message specification, which is published 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, example 3-9 shows a portion of the message model from example 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 articl
  <ChildPropertyAssociation>                                         |     "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,
  </ChildPropertyAssociation>                                        |       "cmf:MaxOccursQuantity": 1
  <ChildPropertyAssociation>                                         |     },
    <DataProperty structures:ref="nc.ItemQuantity" xsi:nil="true     |     "cmf:PropertyAssociation": {
    <MinOccursQuantity>1</MinOccursQuantity>                         |       "cmf:DataProperty": { "@id": "#nc.ItemQuantity" }
    <MaxOccursQuantity>1</MaxOccursQuantity>                         |       "cmf:MinOccursQuantity": 1,
  </ChildPropertyAssociation>                                        |       "cmf:MaxOccursQuantity": 1
</Class>                                                             |     }
                                                                     |   }
                                                                     | }
```

Example 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 to represent 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 must 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.

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, once a specific version of a namespace is published, it can no longer be modified. Updates must go into a new version of the namespace. All published versions of a namespace continue to be valid in support of older exchanges.

In addition, note that a message specification contains its own copy of the schemas that they depend upon. Therefore new versions of a model or a namespace do not affect existing exchanges. Exchange partners may decide to upgrade to a new version of NIEM if they decide it suits their needs, but only if they choose to do so, and only on their own timeline. The NIEM release schedule does not force adopters to keep in sync.

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.

  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. 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 §9.4)*.

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.

3.7 Model extensions

Reuse of a community data model typically supplies some but not all of the necessary data definitions. Model extension allows a model designer to supply the missing definitions. NIEM has two forms of model extension: subclassing and

augmentation.

In a *subclass*, a namespace designer creates a new class in his own namespace to represent a special kind of thing. The new class shares all of the properties of its parent class, and adds properties belonging only to the new class. For example, in the NIEM model, `nc:Vehicle` is a subclass of `nc:Conveyance`. Like any Conveyance, a Vehicle may have the `nc:ConveyanceEngineQuantity` property, but only Vehicles have the `nc:VehicleSeatingQuantity` property; other Conveyances do not.

In an *augmentation*, a namespace designer creates additional properties for a class that is defined in a different namespace. Here the designer is not creating a new class for a new kind of thing. Instead, he is providing properties which could have been defined by the original class designer, but in fact were not. For example, the designers of the NIEM Justice domain have augmented `nc:PersonType` with the `j:PersonSightedIndicator` property, because for the members of the Justice domain it is useful to record whether a person is able to see, even though to the NIEM community as a whole, adding this property to NIEM Core is not worth the trouble.

In general, augmentations are preferred over subclassing. At present the NIEM metamodel does not support multiple inheritance. If several domains were to create a subclass of `nc:PersonType`, there would be no way for a message designer to combine in his message model the properties of a person from NIEM Justice, NIEM Immigration, etc. That combination is easily done with augmentations.

---

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.

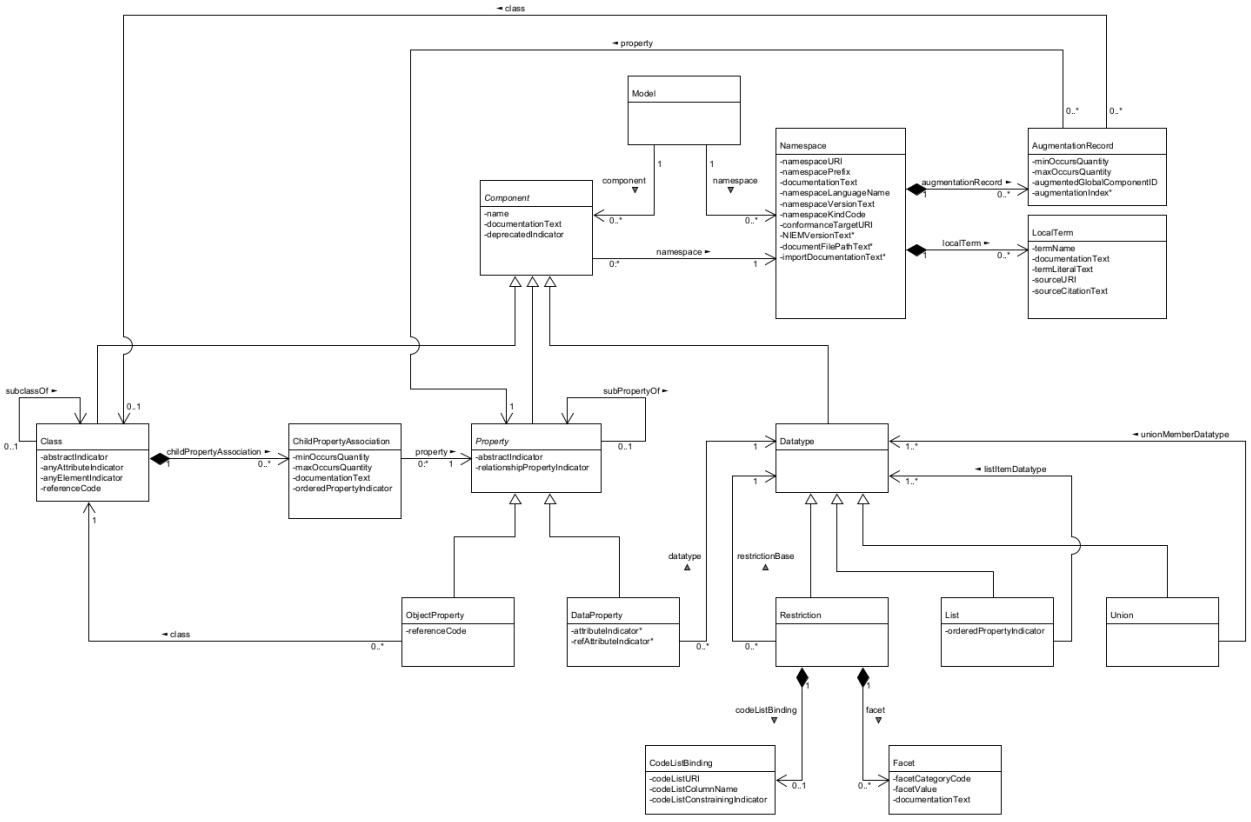


Figure 4-1:The NIEM metamodel

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

Table 4-2: Definition of columns in metamodel property tables

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 7.2). 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

Table 4-3: Definition of columns in CMF-XSD mapping tables

4.1 Model

A Model object represents a NIEM model.

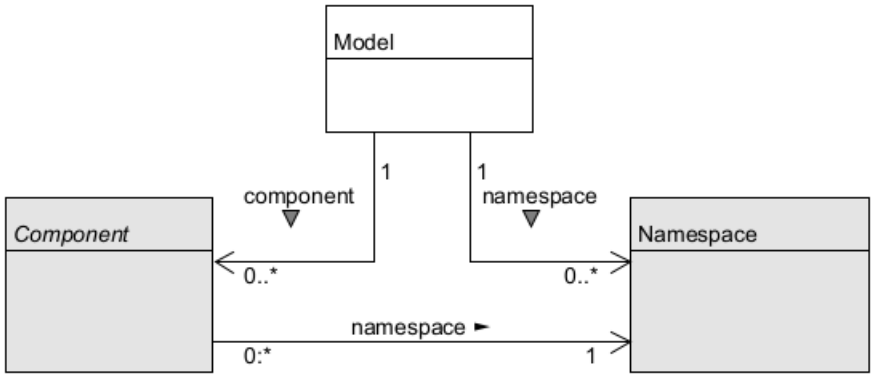


Figure 4-4: Model class diagram

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

Table 4-5: Properties of the Model object class

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.

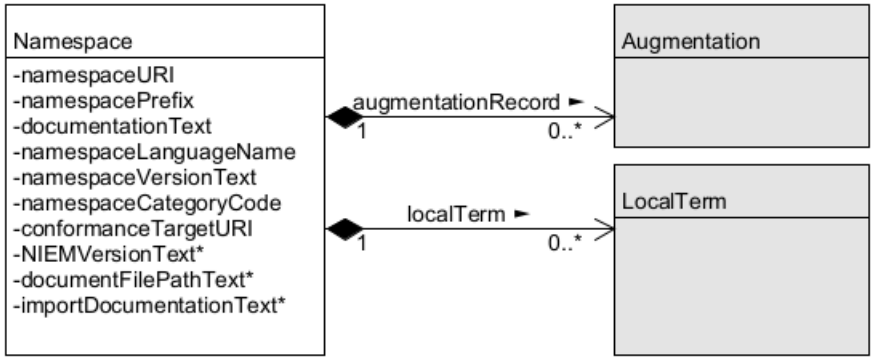


Figure 4-6: Namespace class diagram

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

Name	Definition	Card	Ord	Range
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

Table 4-7: Properties of the Namespace object class

In XSD, an instance of the Namespace class is represented by the `<xs:schema>` element in a schema document. Example 4-8 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/NDR/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/NDR/6.0/#ReferenceSchemaDocument"
  version="ps02"
  xml:lang="en-US">
  <xs:annotation>
    <xs:documentation>NIEM Core.</xs:documentation>
  </xs:annotation>
</xs:schema>
```

Example 4-8: 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`

Table 4-9: Namespace object properties in CMF and XSD

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.

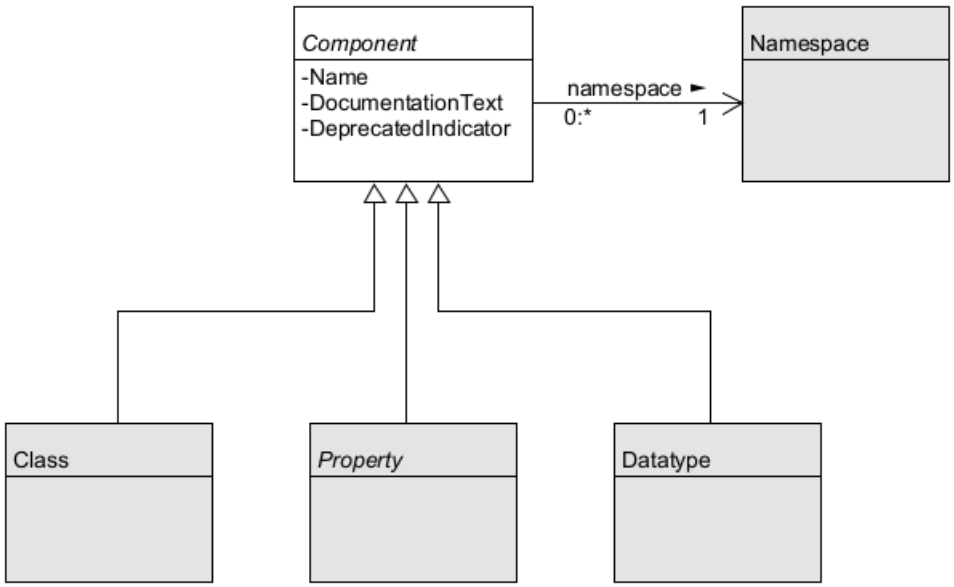


Figure 4-10: Component class diagram

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
Namespace	The namespace of a data model component.	1	-	NamespaceType

Table 4-11: Properties of the Component abstract class

In XSD, the common properties of a Component object are represented by a complex type definition or an element or attribute declaration. Example 4-12 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>
```

Example 4-12: Component object (abstract) in CMF and XSD

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

CMF	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

Table 4-13: Component object properties in CMF and XSD

4.4 Class

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

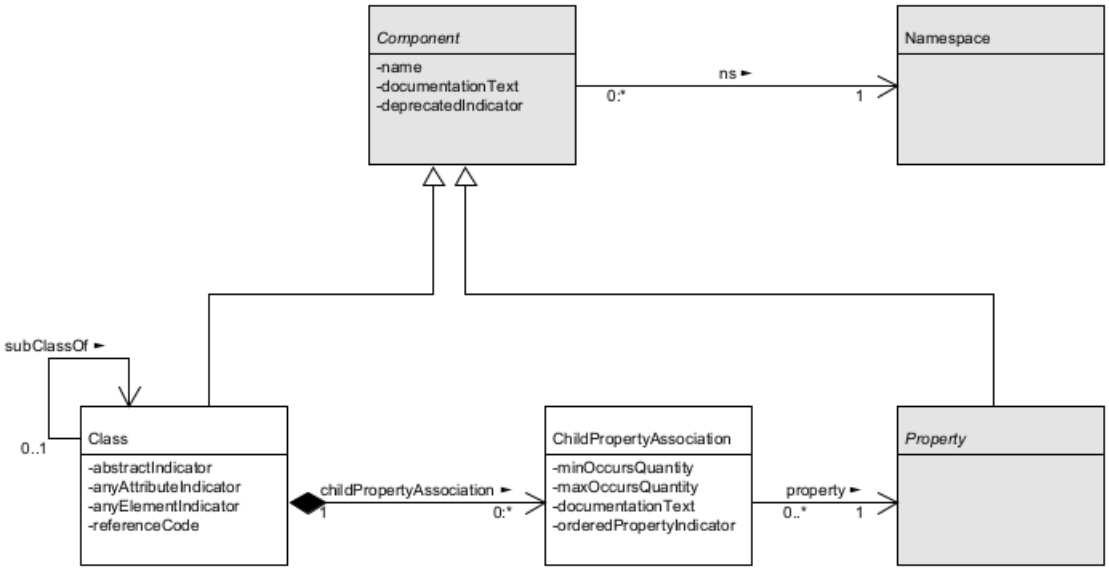


Figure 4-14: Class and ChildPropertyAssociation class diagram

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
AnyAttributeIndicator	True when instances of a class may have arbitrary attribute properties in addition to those specified by ChildPropertyAssociation.	0..1	-	xs:boolean
AnyElementIndicator	True when instances of a class may have arbitrary element properties in addition to those specified by ChildPropertyAssociation.	0..1	-	xs:boolean

Name	Definition	Card	Ord	Range
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

Table 4-15: Properties of the Class object class

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.

Table 4-16: ReferenceCode code list

Class objects may be categorized into four groups, as follows:

- An object class contains one or more properties from a conforming namespace. An object class has a name ending in "Type". Most class objects fall into this category.
- 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 9.4.)
- 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".
- A literal class contains no object properties, at least one attribute property, and exactly one element property. A literal class has a name ending in "Type".

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, example 4-17 shows an XML element with complex content, and also the equivalent in a 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
                                                        |   }
                                                        | }
```

Example 4-17: Instance of a class in XML and JSON

These classes are represented in XSD as a complex type with complex content ("CCC type"); that is, a type with child elements. Example 4-18 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"/>
  <ChildPropertyAssociation>
    <DataProperty structures:ref="ex.MassUnitCode" xsi:nil="true"/>
    <MinOccursQuantity>1</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
  </PropertyAssociation}>
  <ChildPropertyAssociation>
    <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>
```

Example 4-18: A 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`
AnyAttributeIndicator	`xs:anyAttribute`
AnyElementIndicator	`xs:any`
ReferenceCode	`xs:complexType/@appinfo:referenceCode`
SubClassOf	`xs:complexType/xs:complexContent/xs:extension/@base`
ChildPropertyAssociation	`xs:complexType/xs:complexContent/xs:extension/xs:sequence/xs:element` or `xs:complexType/xs:complexContent/xs:extension/xs:attribute`

Table 4-19: Class object object properties in CMF and XSD

Instances of a literal class are represented as an element with simple content and attributes in XML. Example 4-20 below shows an XML and JSON instance of a literal class.

```
<ex:ItemWeightMeasure ex:massUnitCode="KGM">
  22.5
</ex:ItemWeightMeasure>
```

```
{
  "ex:ItemWeightMeasure": {
    "ex:massUnitCode": "KGM",
    "ex:WeightMeasureLiteral": 22.5
  }
}
```

Example 4-20: Instance of a literal class in XML and JSON

A literal class is represented in XSD as a complex type with simple content (“CSC type”) and attributes. This is illustrated in example 4-21 below, which shows a literal class defining the class of the `ItemWeightMeasure` property in example 4-20 above.

```
  <Class structures:id="ex.WeightMeasureType">
    <Name>WeightMeasureType</Name>
    <Namespace structures:ref="ex" xsi:nil="true"/>
    <ChildPropertyAssociation>
      <DataProperty structures:ref="ex.massUnitCode" xsi:nil="true"/>
      <MinOccursQuantity>1</MinOccursQuantity>
      <MaxOccursQuantity>1</MaxOccursQuantity>
    </ChildPropertyAssociation>
    <ChildPropertyAssociation>
      <DataProperty structures:ref="ex.WeightMeasureLiteral" xsi:nil="true"/>
      <MinOccursQuantity>1</MinOccursQuantity>
      <MaxOccursQuantity>1</MaxOccursQuantity>
    </ChildPropertyAssociation>
  </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>
```

Example 4-21: A literal class object in CMF and XSD (CSC type)

A literal 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 literal class, or as a separate element in the XML message.

A literal class always has at least one attribute property. 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
ChildPropertyAssociation	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
Property	The property that occurs in the class.	1	-	PropertyType

Table 4-22: Properties of the ChildPropertyAssociation object class

A ChildPropertyAssociation object is represented in XSD as an element or attribute reference within a complex type definition. Example 4-23 shows the representation of two PropertyAssociation objects, first in CMF, and then in XSD.

```
<ChildPropertyAssociation>
  <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>
</ChildPropertyAssociation>
<ChildPropertyAssociation>
  <DataProperty structures:ref="nc:personNameCommentText" xsi:nil="true"/>
  <MinOccursQuantity>0</MinOccursQuantity>
  <MaxOccursQuantity>1</MaxOccursQuantity>
</ChildPropertyAssociation>
---------------
<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"/>
```

Example 4-23: 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`

Table 4-24: ChildPropertyAssociation object properties in CMF and XSD

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.

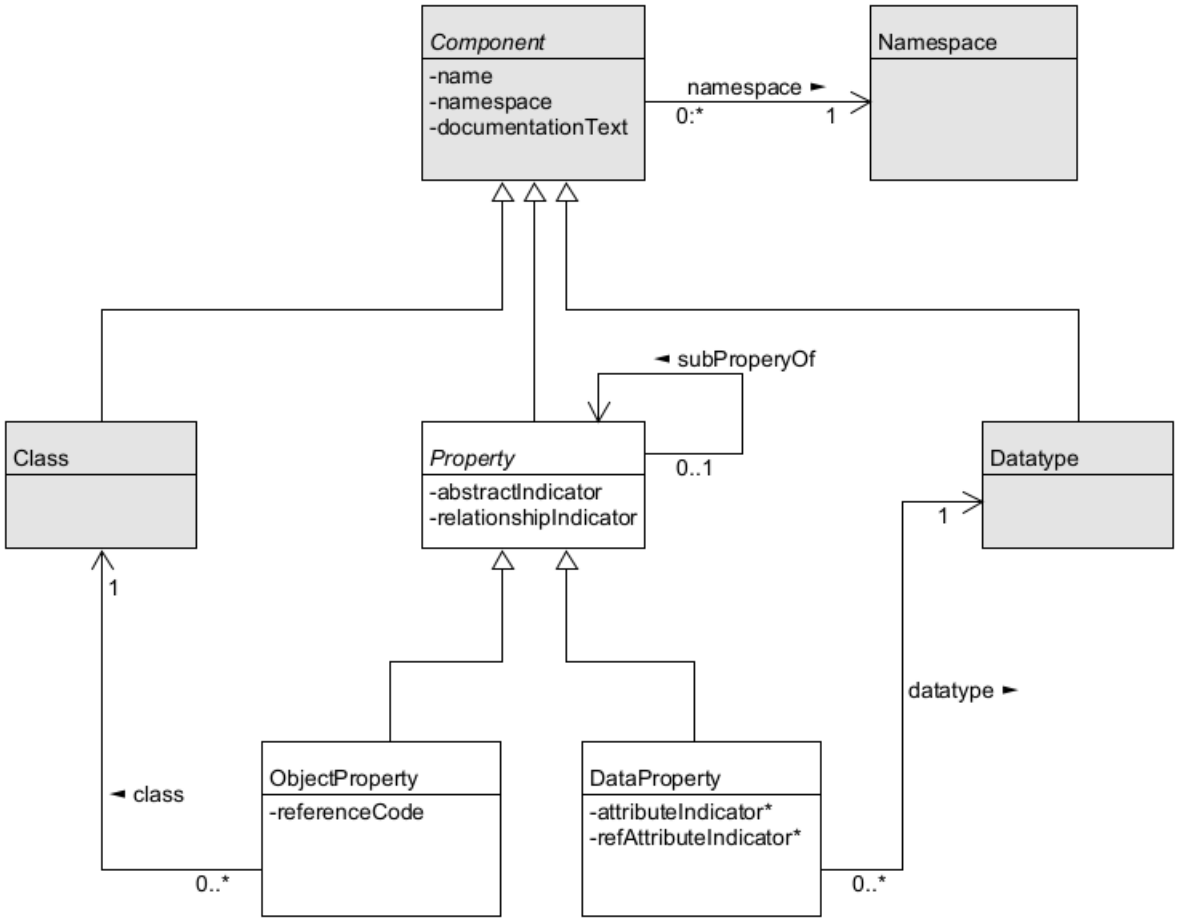


Figure 4-25: Property class diagram

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 relationship property, a property that applies to the relationship between its parent and grandparent objects.	0..1	-	xs:boolean
SubPropertyOf	A property of which a property is a subproperty.	0..1	-	PropertyType

Table 4-26: Properties of the Property abstract class

Apart from the message object, every object in a message is a child property of another object, and typically provides information about that object. A relationship property instead provides information about the relationship between its parent and grandparent objects. Section 5.5 provides an example.

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
Class	The class of this object property.	1	-	ClassType

Table 4-27: Properties of the ObjectProperty object class

An ObjectProperty object is represented in XSD as an element declaration with a type that is a Class object.Example 4-28 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>
```

Example 4-28: 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`

Table 4-29: ObjectProperty object properties in CMF and XSD

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 reference attribute property.	0..1	-	xs:boolean
Datatype	The datatype of this data property.	1	-	DatatypeType

Table 4-30: Properties of the DataProperty object class

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

A reference attribute property is an attribute property that contains one or more identifiers for message objects of a known class. It is interpreted as an [object reference] to each object thus identified. Object references and identiflers are described in section 5.3, and reference attribute properties in section 5.3.6.

A DataProperty object is represented in XSD as an attribute declaration, or as an element declaration with a type that is a Datatype object. Example 4-31 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>
```

Example 4-31: 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`

CMF	XSD
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`

Table 4-32: DataProperty object properties in CMF and XSD

4.9 Datatype

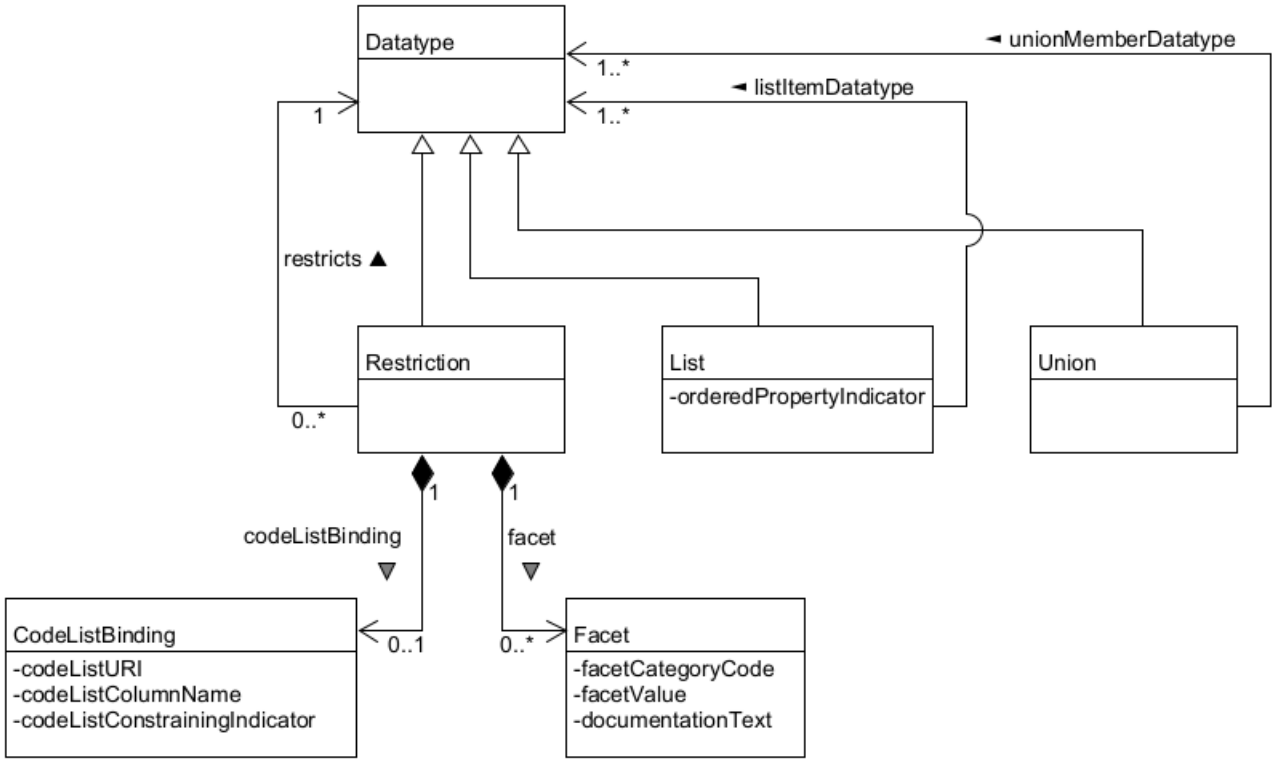


Figure 4-33: Datatype classes

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, example 4-34 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>
```

Example 4-34: 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 literal values.

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

Table 4-35: Properties of the List object class

A List object is represented in XSD as a complex type definition that extends a simple type definition that has an `xs:list` element. Example 4-36 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 of integers.</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>
```

Example 4-36: 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`

CMF	XSD
DeprecatedIndicator	`xs:complexType/@appinfo:deprecated`
ListItemDatatype	`xs:simpleType/xs:list/@itemType`
OrderedPropertyIndicator	`xs:complexType/@appinfo:orderedPropertyIndicator`

Table 4-37: List object properties in CMF and XSD

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

Table 4-38: Properties of the Union object class

A Union object is represented in XSD as a complex type definition that extends a simple type definition that has an `xs:union` element. Example 4-39 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>
```

Example 4-39: 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`

CMF	XSD
UnionMemberDatatype	`xs:simpleType/xs:union/@memberTypes`

Table 4-40: Union object properties in CMF and XSD

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 literal values defined by a data type to a a column of a code list.	0..1	-	CodeListBindingType

Table 4-41: Properties of the Restriction object class

A Restriction object is represented in XSD as a complex type with simple content that contains an `xs:restriction` element. Example 4-42 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>
```

Example 4-42: 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/`*facet-element*
CodeListBinding	`xs:complexType/xs:annotation/xs:appinfo/clsa:SimpleCodeListBinding`

Table 4-43: Restriction object properties in CMF and XSD

A code list is a set of string values, each having a known meaning beyond its value, each representing a distinct conceptual entity. These code values 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 list datatypes have an XSD representation composed of `xs:enumeration` values. Code list datatypes 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

Table 4-44: Properties of the Facet object class

The range of the `FacetCategoryCode` property is a code list. The twelve codes correspond to the twelve constraining facets in XML Schema Structures; 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.Example 4-45 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>
```

Example 4-45: 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`

Table 4-46: Facet object properties in CMF and XSD

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 [Code Lists].

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

Table 4-47: Properties of the CodeListBinding object class

A CodeListBinding object is represented in XSD as a `clsa:SimpleCodeListBinding` element in an `xs:appinfo` element. Example 4-48 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="
    </xs:appinfo>
```

Example 4-48: 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`

Table 4-49: CodeListBinding object properties in CMF and XSD

4.15 Augmentation class

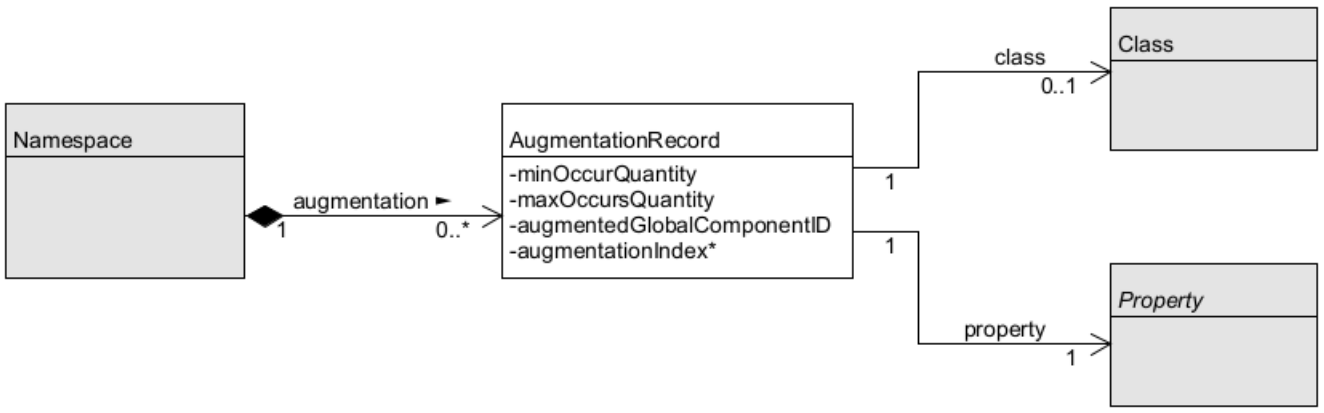


Figure 4-50: Augmentation class diagram

Augmentation 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, the model designers for the NIEM Justice domain have augmented the `nc:PersonType` class with the `j:PersonSightedIndicator` property. Then:

- https://docs.oasis-open.org/niemopen/ns/model/domains/justice/6.0/ is the augmenting namespace
- https://docs.oasis-open.org/niemopen/ns/model/niem-core/6.0/ is the augmented namespace
- `j:PersonSightedIndicator` 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
AugmentationRecord	A data type for a class that is augmented with a property by a namespace.			
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
AugmentationIndex	The ordinal position of an augmentation property that is part of an augmentation type.	0..1	-	xs:integer
GlobalClassCode	A code for a kind of class (object, association, or literal), such that every class in a model of that kind is augmented with a property	0..1	-	GlobalClassCodeType
Class	An augmented class.	0..1	-	ClassType
Property	An augmentation property .	1	-	PropertyType

Table 4-51: Properties of the Augmentation object class

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

```
<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.PersonSightedIndicator" xsi:nil="true"/>
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>unbounded</MaxOccursQuantity>
    <AugmentationIndex>1</AugmentationIndex>
  </AugmentationRecord>
</Namespace>
```

Example 4-52: Augmentation object in CMF

A *global augmentation* adds a property to every class of a specified kind in the model. In CMF, a global augmentation is represented by an AugmentationRecord object with a GlobalClassCode property and no Class property. For example, a global augmentation adding `my:PrivacyCode` to every every object class results in the following CMF for the augmenting namespace.

```
<Namespace>
  <NamespaceURI>http://example.com/MyNamespace/</NamespaceURI>
  <NamespacePrefix>my</NamespacePrefix>
  <AugmentationRecord>
    <Property structures:ref="my.PrivacyCode"/>
    <MinOccursQuantity>1</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
    <AugmentationIndex>0</AugmentationIndex>
    <GlobalClassCode>OBJECT</GlobalClassCode>
  </AugmentationRecord>
</Namespace>
```

Example 4-53: Global augmentation in CMF

A global AugmentationRecord object has no Class property (because it applies to every class). The range of the `GlobalClassCode` property is a code list with the following codes and meanings:

Code	Definition
OBJECT	A code for an augmentation property that applies to all object classes.
ASSOCIATION	A code for an augmentation property that applies to all association classes in the model.
LITERAL	A code for an augmentation property that applies to all datatypes and literal classes in the model. *(see §4.15.5)*

Table 4-54: GlobalClassCode code list

4.15.1 Augmentations in NIEM XSD

The XSD representation of an augmentation is complex, and varies based on two factors:

1. Whether the augmentation property is an attribute property or an element property
2. Whether the model is a message model In a message model, attribute augmentations appear in the schema documents for both the augmenting namespace and the augmented namespace. (See section 4.15.4: *Attribute augmentations in message models*)

4.15.2 Augmenting a class with an element property in XSD

In XSD, a class with element properties is represented by a complex type definition with complex content (a “CCC type”). For example, `nc:PersonType` is represented as the following CCC type definition (some properties are omitted for simplicity):

```
<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>
```

Example 4-55: Example complex type definition with complex content (CCC type)

Every CCC type contains an augmentation point element. This is an abstract element declaration in the same namespace, having the same name as the type which contains it, with the final “Type” replaced with “AugmentationPoint”. Because it is abstract, an augmentation point element cannot appear in a message; it is only a placeholder for element substitution. For

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

```
<xs:element name="PersonAugmentationPoint" abstract="true">
  <xs:annotation>
    <xs:documentation>An augmentation point for PersonType</xs:documentation>
  </xs:annotation>
</xs:element>
```

Example 4-56: Example augmentation point element declaration

In the XSD representation of a model, a namespace augments a CCC type with an element property by defining an augmentation type and an augmentation element. Together these define a container element for the desired augmentation properties that is substitutable for the augmentation point element. For example, example 4-56 shows the XSD for the NIEM Justice namespace augmenting `nc:PersonType` with two properties, and example 4-57 shows an XML message with that augmentation. (The CMF corresponding to the XSD is shown in example 4-52.)

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

Example 4-57: Augmenting a class with an augmentation type and element in XSD

```
<nc:Person>
  <nc:PersonBirthDate>
    <nc:Date>2021-09-11</nc:Date>
  </nc:PersonBirthDate>
  <nc:PersonName>
    <nc:PersonFullName>John Doe</nc:PersonFullName>
  </nc:PersonName>
  <j:PersonAugmentation>
    <j:PersonAdultIndicator>true</j:PersonAdultIndicator>
    <j:PersonSightedIndicator>true</j:PersonSightedIndicator>
  </j:PersonAugmentation>
</nc:Person>
```

Example 4-58: Example message with an augmentation element

All of the augmentations in the XSD representation of the NIEM model use the above approach. There is an alternative approach, in which a namespace augments a CCC type without defining an augmentation type. This is done by making an element property substitutable for the augmentation point element. For example, the namespace http://example.com/Characters could augment `nc:PersonType` with a `PersonFictionalCharacterIndicator` property via the XSD in example 4-59.

```
<xs:element name="PersonFictionalCharacterIndicator" 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>
```

Example 4-59: Augmenting a class with an element property in XSD

```
<nc:Person>
  <nc:PersonBirthDate>
    <nc:Date>2021-09-11</nc:Date>
  </nc:PersonBirthDate>
  <nc:PersonName>
    <nc:PersonFullName>John Doe</nc:PersonFullName>
  </nc:PersonName>
  <chars:PersonFictionalCharacterIndicator>true</nc:PersonFictionalCharacterIndicator>
</nc:Person>
```

Example 4-60: Example message showing augmentation with an element property

The CMF corresponding to the XSD in example 4-59 is shown below. Since there is no augmentation type in the XSD, the AugmentationRecord object does not have an AugmentationIndex property to show the position of the augmentation property within that type.

```
<Namespace>
  <NamespaceURI>http://example.com/Characters/1.0</NamespaceURI>
  <NamespacePrefix>chars/NamespacePrefix>
  <DocumentationText>Example namespace for NDR6.</DocumentationText>
  <AugmentationRecord>
    <Class structures:ref="nc.PersonType" xsi:nil="true"/>
    <DataProperty structures:ref="chars.PersonFictionalCharacterIndicator" xsi:nil="true"/>
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
  </AugmentationRecord>
</Namespace>
```

Example 4-61: CMF for an element property augmentation

4.15.3 Augmenting a literal class or datatype with an element property in XSD

In the XSD representation of a model, a complex type definition with simple content (“CSC type”) can represent either a literal class or a datatype. It is not possible to directly augment either kind of CSC type with an element property, because element properties are only possible within a CCC type. The desired effect is instead accomplished by augmenting the literal class or datatype with a reference attribute property. These are described in section 5.3.6. Note that augmenting a datatype with an attribute necessarily converts it into a literal class; see section 5.1.)

4.15.4 Augmenting a class with an attribute property in XSD

In the XSD representation of a model, a namespace augments a class with an attribute property by writing application information into the namespace schema document. For example, example 4-62 shows the XSD for the *Characters* namespace augmenting `nc:PersonType` with the attribute property `chars:genre`, and example 4-63 shows an XML message with that augmentation.

```
<xs:schema
  targetNamespace="http://example.com/Characters/1.0/"
  xmlns:myChars="http://example.com/Characters/1.0/"
  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/NDR/6.0/#ExtensionSchemaDocument"
  version="1.0"
  xml:lang="en-US">
  <xs:annotation>
    <xs:documentation>Example Characters namespace for NDR6.</xs:documentation>
    <xs:appinfo>
      <appinfo:Augmentation class="nc:PersonType" property="myChars:genre"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:attribute name="genre" type="xs:token">
    <xs:annotation>
      <xs:documentation>A name of a genre of fiction applicable to a fictional character.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:schema>
```

Example 4-62: Augmenting a class with an attribute property in XSD

```
<nc:Person myChars:genre="mystery">
  <nc:PersonBirthDate>
    <nc:Date>1890-10-15</nc:Date>
  </nc:PersonBirthDate>
  <nc:PersonName>
    <nc:PersonFullName>Peter Death Bredon Wimsey</nc:PersonFullName>
  </nc:PersonName>
  <chars:PersonFictionalCharacterIndicator>true</nc:PersonFictionalCharacterIndicator>
</nc:Person>
```

Example 4-63: Example message showing an attribute property augmentation

4.15.5 Global augmentations in XSD

Global augmentation with an element property is represented in XSD by creating an *augmentation element* substitutable for `structures:ObjectAugmentationPoint` or `structures:AssociationAugmentationPoint`. For example, example 4-64 shows the XSD for the *Privacy* namespace augmenting all object classes with the `priv:Restriction` element property; example 4-65 shows part of an XML message with that augmentation.

```
<xs:complexType name="ObjectAugmentationType">
  <xs:annotation>
    <xs:documentation>A data type for additional information about an object.</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:AugmentationType">
      <xs:sequence>
        <xs:element ref="priv:Restriction"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:element name="ObjectAugmentation" type="priv:ObjectAugmentationType substitutionGroup="structures:ObjectAugmentationPo
  <xs:annotation>
    <xs:documentation>Additional information about an object.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Example 4-64: Global augmentation with an element property in XSD

```
<nc:Person>
  <priv:ObjectAugmentation>
    <priv:Restriction>PII</priv:Restriction>
  </priv:ObjectAugmentation>
  <nc:PersonName>
    <nc:PersonFullName>John Doe</nc:PersonFullName>
  </nc:PersonName>
</nc:Person>
```

Example 4-65: Global augmentation with an element property in XSD

Global augmentation with an attribute property is represented in XSD by writing application information into the augmenting namespace schema document. Instead of specifying the augmented class, this appinfo provides a code from `GlobalAugmentationCodeType`. For example, example 4-66 shows the XSD for the *Privacy* namespace augmenting all object classes with the `priv:classification` attribute property.

```
<xs:schema
  targetNamespace="http://example.com/Privacy/1.0/"
  xmlns:priv="http://example.com/Privacy/1.0/"
  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/NDR/6.0/#ExtensionSchemaDocument"
  version="1.0"
  xml:lang="en-US">
  <xs:annotation>
    <xs:documentation>Example Privacy namespace for NDR6.</xs:documentation>
    <xs:appinfo>
      <appinfo:Augmentation property="priv:classification" globalClassCode="OBJECT"/>
    </xs:appinfo>
  </xs:annotation>
</xs:schema>
```

Example 4-66: Global augmentation with an attribute property in XSD

4.15.6 Attribute augmentations in message models

The XSD representation of a message model must successfully validate all conforming messages. This means the augmented type definition has to include the augmenting attribute property. For example, the highlighted line in example 4-67 shows how the type definition of `nc:PersonType` would include the augmentation property `chars:genre`.

```
<xs:complexType name="PersonType"></code>
  <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="myChars:genre" appinfo:augmentingNamespace="chars"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Example 4-67: Example complex type definition with complex content (CCC type)

The `appinfo:augmentingNamespace` attribute is required; it declares that this attribute reference is an augmentation. The value of the attribute may be either the namespace prefix or URI.

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

Table 4-68: Properties of the LocalTerm object class

A LocalTerm object is represented in XSD by a `appinfo:LocalTerm` element within `xs:appinfo` element in the `xs:schema` element. Example 4-69 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>
```

Example 4-69: 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`

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

Table 4-70: LocalTerm object properties in CMF and XSD

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

Table 4-71: Properties of the TextType object class

5. Data modeling patterns

This section is informative. It explains common patterns in NIEM models and messages.

5.1 Datatypes and literal classes

A model component can be a datatype in one message model and a class in another. This occurs when a message designer creates a subset of a reused literal class, or augments a reused datatype.

Removing attribute properties from a reused literal class can turn it into a datatype. For example, `nc:NumericType` is a literal class in the NIEM model, but in a subset can become a datatype in a message model. In the NIEM model, `nc:NumericType` has one element property and one attribute property. Example 5-1 shows the class representation in CMF and XSD;example 5-2 shows an object of the class in an XML and JSON message.

```
<Class structures:id="nc.NumericType">                    | <xs:complexType name="NumericType">
  <Name>NumericType</Name>                                |   <xs:simpleContent>
  <Namespace structures:ref="nc"                          |     <xs:extension base="niem-xs:decimal">
  <ChildPropertyAssociation>                              |       <xs:attribute ref="nc:toleranceNumeric" use="optional
    <DataProperty structures:ref="nc.NumericLiteral"/>    |     </xs:extension>
    <MinOccursQuantity>1</MinOccursQuantity>              |   </xs:simpleContent>
    <MaxOccursQuantity>1</MaxOccursQuantity>              | </xs:complexType>
  </ChildPropertyAssociation>                             |
  <ChildPropertyAssociation>                              |
    <DataProperty structures:ref="nc.toleranceNumeric"/>  |
    <MinOccursQuantity>0</MinOccursQuantity>              |
    <MaxOccursQuantity>1</MaxOccursQuantity>              |
  </ChildPropertyAssociation>                             |
</Class>                                                  |
```

Example 5-1: A literal class in CMF and XSD

```
<my:Message>                                                    | "my:Message": {
  <my:MaximumNumber nc:toleranceNumeric="2">7<my:MaximumNumber>  |   "my:MaximumNumber": {
</my:Message>                                                   |     "nc:NumericLiteral": "7",
                                                                |     "nc:toleranceNumeric": "2"
                                                                |   }
                                                                | }
```

Example 5-2: Objects of a literal class in an XML and JSON message

If a message designer decides to reuse `nc:NumericType`, and to remove `nc:toleranceNumeric` from the class in his model subset, then `nc:NumericType` becomes a datatype in the subset. Example 5-3 shows the CMF and XSD representations of that subset; example 5-4 shows the resulting data property in an XML and JSON message.

```
<Restriction structures:id="nc.NumericType">          | <xs:complexType name="NumericType">
  <Name>NumericType</Name>                            |   <xs:simpleContent>
  <Namespace structures:ref="nc"                      |     <xs:extension base="niem-xs:decimal"/>
  <RestrictionBase structures:ref="xs:decimal"/>      |   </xs:simpleContent>
</Restriction>                                        | </xs:complexType>
```

Example 5-3: A restriction datatype in a CMF and XSD model subset

```
<my:Message>                                          | "my:Message": {
  <my:MaximumNumber>7<my:MaximumNumber>               |   "my:MaximumNumber": "7"
</my:Message>                                         | }
```

Example 5-4: A data property in an XML and JSON message

Going the other way, augmenting a reused datatype turns it into a literal class. For example, `nc:PersonUnionCategoryCodeType` is a datatype in the NIEM model, and `nc:PersonUnionCategoryCode` is a data property with that datatype. Example 5-5 shows the datatype representation in CMF and XSD; example 5-6 shows the data property in an XML and JSON message.

```
<Restriction structures:id="nc.PersonUnionCategoryCodeType">      | <xs:complexType name="PersonUnionCategoryCodeType">
  <Name>PersonUnionCategoryCodeType</Name>                        |   <xs:simpleContent>
  <Namespace structures:ref="nc" xsi:nil="true"/>                 |     <xs:restriction base="niem-xs:token">
  <RestrictionBase structures:ref="xs.token" xsi:nil="true"/>     |       <xs:enumeration value="civil union"/>
  <Enumeration>                                                   |       <xs:enumeration value="common law"/>
    <StringValue>civil union</StringValue>                        |       <xs:enumeration value="domestic partnership"/>
  </Enumeration>                                                  |       <xs:enumeration value="married"/>
  <Enumeration>                                                   |       <xs:enumeration value="unknown"/>
    <StringValue>common law</StringValue>                         |     </xs:restriction>
  </Enumeration>                                                  |   </xs:simpleContent>
  <Enumeration>                                                   | </xs:complexType>
    <StringValue>domestic partnership</StringValue>               |
  </Enumeration>                                                  |
  <Enumeration>                                                   |
    <StringValue>married</StringValue>                            |
  </Enumeration>                                                  |
  <Enumeration>                                                   |
    <StringValue>unknown</StringValue>                            |
  </Enumeration>                                                  |
</Restriction>                                                    |
```

Example 5-5: A datatype in CMF and XSD

```
<nc:Person>                                                     | "nc:Person": {
  <nc:PersonUnionCategoryCode>married</nc:PersonUnionCategoryCode>  |   "nc:PersonUnionCategoryCode": "married"
</nc:Person>                                                    | }
```

Example 5-6: A data property in an XML and JSON message

A message designer might decide to augment `nc:PersonUnionCategoryCodeType` with metadata to indicate this information is

sometimes privileged. Doing so turns it into a literal class in his model subset. Example 5-7 shows the CMF and XSD representations of that subset; example 5-8 shows the resulting object in an XML and JSON message.

```
<Restriction structures:id="nc.PersonUnionCategoryCodeSimple
  <Name>PersonUnionCategoryCodeSimpleType</Name>
  <Namespace structures:ref="nc"/>
  <RestrictionBase structures:ref="xs.token" xsi:nil="true"/
  <Enumeration>
    <StringValue>civil union</StringValue>
  </Enumeration>
  <Enumeration>
    <StringValue>common law</StringValue>
  </Enumeration>
  <Enumeration>
    <StringValue>domestic partnership</StringValue>
  </Enumeration>
  <Enumeration>
    <StringValue>married</StringValue>
  </Enumeration>
  <Enumeration>
    <StringValue>unknown</StringValue>
  </Enumeration>
</Restriction>
<DataProperty structures:id="nc.PersonCategoryCodeLiteral">
  <Name>PersonUnionCategoryCodeLiteral</Name>
  <Namespace structures:ref="nc"/>
  <Datatype structures:ref="nc.PersonUnionCategoryCodeSimple
</DataProperty>
<Class>
  <Name>PersonUnionCategoryCodeType</Name>
  <Namespace structures:ref="nc"/>
  <ChildPropertyAssociation>
    <DataProperty structures:ref="nc.PersonCategoryCodeLiter
    <MinOccursQuantity>1</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
  </ChildPropertyAssociation>
  <ChildPropertyAssociation>
    <DataProperty structures:ref="my.privileged"/>
    <MinOccursQuantity>0</MinOccursQuantity>
    <MaxOccursQuantity>1</MaxOccursQuantity>
    <AugmentingNamespace>my</AugmentingNamespace>
  </ChildPropertyAssociation>
</Class>
```

```
<xs:simpleType name="PersonUnionCategoryCodeSimpleType">
  <xs:restriction base="xs:token">
    <xs:enumeration value="civil union"/>
    <xs:enumeration value="common law"/>
    <xs:enumeration value="domestic partnership"/>
    <xs:enumeration value="married"/>
    <xs:enumeration value="unknown"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="PersonUnionCategoryCodeType">
  <xs:simpleContent>
    <xs:extension base="nc:PersonUnionCategoryCodeSimpleTyp
      <xs:attribute ref="my:privileged"
          appinfo:augmentingNamespace="my"/>
      <xs:attributeGroup ref="structures:SimpleObjectAttrib
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

Example 5-7: A literal class in a CMF and XSD model subset

```
<nc:Person>
  <nc:PersonUnionCategoryCode
      my:privileged="true">married</nc:PersonUnionCategoryCode>
</nc:Person>
```

```
"nc:Person": {
  "nc:PersonUnionCategoryCode": {
    "nc:PersonUnionCategoryCodeLiteral": "married",
    "my:privileged": "true"
  }
}
```

Example 5-8: An object property with a code list class in an XML and JSON message

The representation of a literal class is complex when compared to the datatype. The JSON message is likewise complicated. Best practice is therefore to avoid augmenting a datatype whenever possible.

5.2 Meaning of NIEM data

The meaning of NIEM data is partly expressed through the hierarchy of nested objects in a message, and partly through the message model's definition of those objects. For example, the meaning of the two equivalent messages in example 3-2 (reproduced below) is described in table 5-9.

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

Message data	Description	Meaning
`<msg:Request>` *or* `"msg:Request":{...}`	The initial property is `msg:Request`. The message model defines the range of this property as the `msg:RequestType` class.	There is an object that is a request for a specified quantity of a named item.
`<msg:RequestID>` *or `"msg:RequestID":...`	The next property is `msg:RequestID`. The message model defines the range of this data property as the `xs:token` datatype.	There is a relationship between the object of `msg:RequestType` and the literal value `RQ001`.
`<msg:RequestedItem>` *or* `"msg:RequestedItem":{...}`	The next property is `msg:RequestedItem`. The message model defines the range of this object property as the `nc:ItemType` class.	There is a relationship between the object of `msg:RequestType` and the object of `nc:ItemType`.
`<nc:ItemName>` *or* `"nc:ItemName":...`	The next property is `nc:ItemName`. The message model defines the range of this data property as the `nc:TextType` datatype.	There is a relationship between the object of `nc:ItemType` and the literal value `Wrench`.
`<nc:ItemQuantity>` *or* `nc:ItemQuantity":...`	The next property is `nc:ItemQuantity`. The message model defines the range of this data property as the `nc:QuantityType` datatype.	There is a relationship between the object of `nc:ItemType` and the literal value `10`.

Table 5-9: Meaning of NIEM data

NIEM is designed so that NIEM data is a form of RDF data. For example, the message data above corresponds to the RDF shown in example 5-10

```
@prefix nc: <https://docs.oasis-open.org/niemopen/ns/model/adapters/niem-xs/6.0/> .
@prefix msg: <http://example.com/ReqRes/1.0/> .
_:n1 a msg:RequestType .
_:n1 msg:RequestID "RQ001".
_:n1 msg:RequestedItem _:n2 .
_:n2 a nc:ItemType .
_:n2 nc:ItemName "Wrench" .
_:n2 nc:ItemQuantity "10" .
```

Example 5-10: RDF interpretation of NIEM data (Turtle syntax)

That RDF data expresses a graph, illustrated by the diagram infigure 5-11.

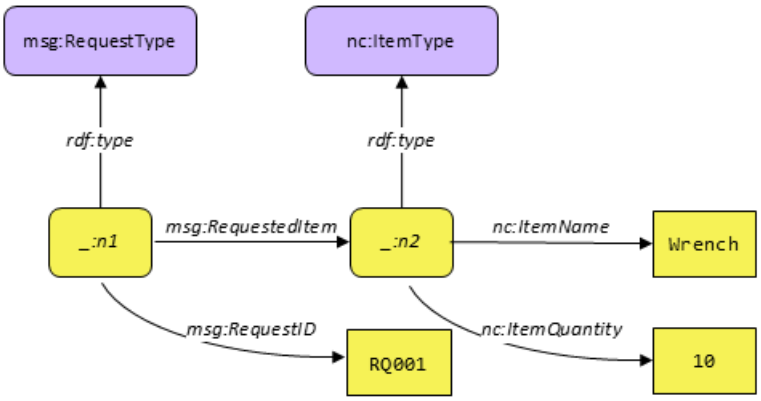


Figure 5-11: Diagram showing meaning of NIEM data

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

Likewise, nothing may be inferred from data that is not present in a NIEM message. It may be absent due to lack of availability, lack of knowledge, or deliberate withholding of information. These cases should be modeled explicitly, if they are required.

5.3 Identifiers and references in NIEM messages

A hierarchy of nested objects (illustrated above) is sufficient to represent simple data that takes the form of a tree. However, this simple representation has limitations, and is not capable of expressing all relationships among objects. 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. These relationships do not form a tree, and require a data structure that is a graph, rather than a simple hierarchy of objects.
- 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 objects, this would result in a duplicate representation of Sally.

NIEM solves these problems through object identifiers and object references. Any object may have an identifier. An object reference can take the place of any object in a message, and is interpreted as if the object with the same identifier actually appeared in that place. The resulting data structure is a graph, not a tree.

For example, in example 5-12 below, there is only one Person object in the message; it has the identifier `JD`, and is a child property of `nc:PersonLocationAssociation`. The `nc:Person` property of the `nc:PersonOrganizationAssociation` object is an object reference. The interpretation is that the person located at the Pentagon is also the person associated with the US Army.

```
<nc:PersonLocationAssociation>                          | "nc:PersonLocationAssociation": {
  <nc:Person structures:id="JD">                        |   "nc:Person": {
    <nc:PersonName>                                     |     "@id": "#JD",
      <nc:PersonFullName>4R</nc:PersonFullName>         |     "nc:PersonName": {
    </nc:PersonName>                                    |       "nc:PersonFullName": "John Doe"
  </nc:Person>                                          |     }
  <nc:Location>                                         |   },
    <nc:LocationName>Pentagon</nc:LocationName>         |   "nc:Location": {
  </nc:Location>                                        |     "nc:LocationName": "Pentagon"
</nc:PersonLocationAssociation>                         |   }
<nc:PersonOrganizationAssociation>                      | },
  <nc:Person structures:ref="JD" xsi:nil="true"/>       | "nc:PersonOrganizationAssociation": {
  <nc:Organization>                                     |   "nc:Person": {
    <nc:OrganizationName>US Army</nc:OrganizationName>  |     "@id": "#JD"
  </nc:Organization>                                    |   },
</nc:PersonOrganizationAssociation>                     |   "nc:Organization": {
                                                        |     "nc:OrganizationName": "US Army"
                                                        |   }
                                                        | }
```

Example 5-12: Example of object references in NIEM XML and JSON

5.3.1 Object references in NIEM XML using structures:id and structures:ref

[XML] defines ID and IDREF attributes; these act as references in XML data. NIEM XML uses ID and IDREF as one way to reference data across data objects.

- `structures:id` is an ID attribute. Its value is an identifier for the object in which it appears. For example, in example 5-12 the value `JD` is an identifier for the `nc:Person` object. According to the rules of XML, an ID value must be unique within the XML document.

  An ID attribute is a *fragment identifier* within the XML document. For example, if the message as a whole has the URI http://example.com/MSG/, then the object identifier `JD` is equivalent to http://example.com/MSG/#JD.

- `structures:ref` is an IDREF attribute. An object with this attribute is a reference to the object with that identifier. For example, in example 5-12, the element `<nc:Person structures:ref="JD" xsi:nil="true"/>` is a reference to the `<nc:Person>` object that has the identifier `JD`.

  The `structures:ref` attribute has type `xs:IDREF`, so according to the rules of XML the message must contain an ID attribute with the same value. This means a `structures:ref` reference can only link to an object within the same message.

  Object references using `structures:ref` must not have content. If the object type has mandatory content, then `xsi:nil="true"` is required.

5.3.2 Object references in NIEM XML using structures:uri

NIEM introduced support for linked data through the use of uniform resource identifiers (URIs), expressed in NIEM XML through the attribute `structures:uri`. In linked data, 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. In NIEM, the objects in a message are the resources; an object identifier is a resource identifier.

The `structures:uri` attribute assigns an object identifier to the element in which it appears. All of the elements having the same identifier refer to a single object, and all of those elements provide property values for that one object. For example, in example 5-13 below, there is only one Person object in the message. The person located at the Pentagon is also the person associated with the US Army; that person is named “John Doe” and also has red hair.

```
<nc:PersonLocationAssociation>                          | "nc:PersonLocationAssociation": {
  <nc:Person structures:uri="#JD">                      |   "nc:Person": {
    <nc:PersonName>                                     |     "@id": "#JD",
      <nc:PersonFullName>John Doe</nc:PersonFullName>   |     "nc:PersonName": {
    </nc:PersonName>                                    |       "nc:PersonFullName": "John Doe"
  </nc:Person>                                          |     }
  <nc:Location>                                         |   },
    <nc:LocationName>Pentagon</nc:LocationName>         |   "nc:Location": {
  </nc:Location>                                        |     "nc:LocationName": "Pentagon"
</nc:PersonLocationAssociation>                         |   }
<nc:PersonOrganizationAssociation>                      | },
  <nc:Person structures:uri="#JD">                      | "nc:PersonOrganizationAssociation": {
    <nc:PersonHairColorText>Red</nc:PersonHairColorText>|   "nc:Person": {
  </nc:Person>                                          |     "@id": "#JD",
  <nc:Organization>                                     |     "nc:PersonHairColorText": "Red"
    <nc:OrganizationName>US Army</nc:OrganizationName>  |   },
  </nc:Organization>                                    |   "nc:Organization": {
</nc:PersonOrganizationAssociation>                     |     "nc:OrganizationName": "US Army"
                                                        |   }
                                                        | }
```

Example 5-13: Example of URI object references in NIEM XML and JSON

The `structures:uri` attribute has the type `xs:anyURI`. Values can be either a *URI-reference* or a *relative-ref*, as defined by [RFC 3986]. A URI-reference can be a URN or URL; for example:

```
  <nc:Person structures:uri="urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"/>
  <nc:Person structures:uri="http://example.com/PersonID/B263-1655-2187"/>
```

If the message as a whole has a URI, then a relative reference is interpreted according to the rules for reference resolution in [RFC 3986]. For example, if the message URI is http://example.com/MSG/, then the relative reference `JD` resolves to http://example.com/MSG/#JD.

A relative resource in `structures:uri` is the same thing as a fragment identifier in `structures:id`, but with a leading `#` character. For example, `structures:uri="#JD"` and `structures:id="JD"` denote the same resource identifier.

5.3.3 Comparison of object references in NIEM XML

- `structures:ref` and `structures:id` must appear within the same message.
- `structures:ref` requires and provides 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 XML document.
- The value of `structures:ref` must appear within the document as the value of an attribute `structures:id`.
- A `structures:uri` can reference any `structures:id` in the same message, or in another message.
- Any `structures:uri` may reference any other `structures:uri`, in the same message, or in another message.

5.3.4 Object references in NIEM JSON using @id

Object references in NIEM JSON use JSON-LD's `@id` keyword. This is equivalent to `structures:uri` in NIEM XML. For example, the following NIEM XML and JSON references mean the same thing and are interpreted in the same way. (There is no JSON equivalent to XML's ID/IDREF attributes.)

```
<nc:Person structures:uri="#JD">
"nc:Person": { "@id": "#JD" }
```

The two JSON objects in example 5-13 that are values of a `nc:Person` key have the same `#JD` value for `@id`. That means

the two JSON objects contain properties of a single NIEM message object, representing a person named “John Doe” who has red hair.

5.3.5 Meaning of inline objects and object references

An important aspect of all of the object reference mechanisms (`structures:ref`, `structures:uri`, and `@id`) is that they all have the same meaning. There is also no difference in meaning between an object that appears inline and an object that appears through a reference.

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

5.3.6 Reference attribute properties

A reference attribute property contains a list of object identifiers, and is interpreted as an object reference to each of the objects thus identified, each a child property of the object containing the reference attribute property. For example, the two XML messages in example 5-14 have the same meaning.

```
<my:Thing nc:metadataRef="m6 m7">                       | <my:Thing>
  <my:ThingName>The Snark</my:ThingName>                |   <my:ThingName>The Snark</my:ThingName>
  <my:ThingLocation>Dismal Valley</my:ThingLocation>    |   <my:ThingLocation>Dismal Valley</my:ThingLocation>
</my:Thing>                                             |   <nc:Metadata>
<nc:Metadata structures:id="m6">                        |     <nc:ConfidencePercent>75</nc:ConfidencePercent>
  <nc:ConfidencePercent>75</nc:ConfidencePercent>       |   </nc:Metadata>
</nc:Metadata>                                          |   <nc:Metadata>
<nc:Metadata structures:id="m7">                        |     <nc:SourceIDText>Bingo-7</nc:SourceIDText>
  <nc:SourceIDText>Bingo-7</nc:SourceIDText>            |   </nc:Metadata>
</nc:Metadata>                                          | </my:Thing>
```

Example 5-14: Reference attribute property and equivalent message in XML

Example 5-15 shows the equivalent JSON message.

```
"my:Thing": {
  "my:ThingName": "The Snark",
  "my:ThingLocation": "Dismal Valley",
  "nc:metadataRef": [
    { "@id": "#m6"},
    { "@id": "#m7"}
  ]
},
"nc:Metadata": [
  {
    "nc:ConfidencePercent": "75",
    "@id": "#m6"
  },
  {
    "nc:SourceIDText": "Bingo-7",
    "@id": "#m7"
  }
]
```

Example 5-15: Reference attribute property in JSON message

The class of these objects is determined by the name of the reference attribute property. For example, an object reference belonging to `nc:metadataRef` must have the class `nc:MetadataType`, or a derived class.*(see rule 12-11.)*

5.4 Metadata and augmentation

Metadata is data about data. The distinction is created by intended use. To the person editing an image, the creation

timestamp is metadata, something he does not need. To the person writing software to sort photos into creation order, the timestamp is the data for his code. One man's metadata is another man's data.

The NIEM model contains a number of classes and properties that are suitable for metadata representations, and any model designer is free to invent new components for this purpose, as needed. A message designer may use these components in his message model, in the same way as any other component. For example, a message designer might, within the components he creates, use `nc:Metadata` to represent a source of information and the level of confidence in that information. Figure 5-16 shows an example of a message in which the designer chose to use `nc:Metadata` as a property within his own `my:ThingType` class.

```
<my:Thing>
  <my:ThingName>The Snark</my:ThingName>
  <my:ThingLocation>Dismal Valley</my:ThingLocation>
  <nc:Metadata>
    <nc:ConfidencePercent>75</nc:ConfidencePercent>
    <nc:SourceIDText>Bingo-7</nc:SourceIDText>
```

Example 5-16: Metadata properties used in a designer's own class

A message designer might also want to record source and confidence in a class reused from another namespace. This is done through augmentation, following one of two patterns. The first is to augment the class with an object property. Example 5-17 shows a message example in which `nc:PersonType` is augmented with `nc:Metadata`.

```
<nc:Person>                                          | "nc:Person": {
  <nc:PersonBirthDate>                               |   "nc:PersonBirthDate": {
    <nc:Date>2021-09-11</nc:Date>                    |     "nc:Date": "2021-09-11"
  </nc:PersonBirthDate>                              |   },
  <nc:PersonName>                                    |   "nc:PersonName": {
    <nc:PersonFullName>John Doe</nc:PersonFullName>  |     "nc:PersonFullName": "John Doe"
  </nc:PersonName>                                   |   },
  <my:PersonAugmentation>                            |   "nc:Metadata": {
    <nc:Metadata">                                   |     "nc:SourceIDText": "Tango-7"
      <nc:SourceIDText>Tango-7</nc:SourceIDText>     |   }
    </nc:Metadata>                                   | }
  </my:PersonAugmentation>                           |
</nc:Person>                                         |
```

Example 5-17: Metadata object property augmenting a reused class

The above augmentation pattern only works for a class with element properties. To add metadata properties to a literal class, the message designer must augment the class with a reference attribute property. Example 5-18 shows a message example in which `nc:PersonNameTextType` is augmented with `nc:metadataRef`.

```
<nc:Person>                                                          | "nc:Person": {
  <nc:PersonBirthDate>                                               |   "nc:PersonBirthDate": {
    <nc:Date>2021-09-11</nc:Date>                                    |     "nc:Date": "2021-09-11"
  </nc:PersonBirthDate>                                              |   },
  <nc:PersonName>                                                    |   "nc:PersonName": {
    <nc:PersonFullName nc:metadataRef="m2">John Doe</nc:PersonFullName>  |     "nc:PersonFullName": "John Doe",
  </nc:PersonName>                                                   |     "nc:metadataRef": "#m2"
</nc:Person>                                                         |   },
<nc:Metadata structures:id="m2">                                     |   "nc:Metadata": {
  <nc:ConfidencePercent>75</nc:ConfidencePercent>                    |     "@id": "#m2",
</nc:Metadata>                                                       |     "nc:ConfidencePercent": "75",
                                                                     |   }
                                                                     | }
```

Example 5-18: Metadata reference attribute augmenting a reused class

5.5 Relationship properties

The value of a property usually provides information about its parent object. For example, the value of

`nc:personNameCommentText` in example 5-19 tells us something about the parent object; namely, that this name is a silly name.

```
<nc:Person>                                                    | "nc:Person": {
  <nc:PersonName nc:personNameCommentText="This is a silly name">  |   "nc:PersonName": {
    <nc:PersonFullName>Bozo the Clown</nc:PersonFullName>      |     "nc:personNameCommentText": "This is a silly na
  </nc:PersonName>                                             |     "nc:PersonFullName": "Bozo the Clown"
</nc:Person>                                                   |   }
                                                               | }
```

Example 5-19: Example of an ordinary property

Sometimes that is not what is needed. For example, in example 5-20, the relationship property `my:isSecret` is not telling us the name “Superman” is a secret. Everybody knows that name! Instead, `my:isSecret` is telling us something about the *relationship* between the name “Superman” and the person object with the other name “Clark Kent”. That relationship is the thing to be kept secret.

```
<nc:Person>                                                | "nc:Person": {
  <nc:PersonName my:isSecret="true">                       |   "nc:PersonName": [
    <nc:PersonFullName>Superman</nc:PersonFullName>        |     {
  </nc:PersonName>                                         |       "nc:PersonFullName": "Superman",
  <nc:PersonName>                                          |       "@annotation": { "my:isSecret": "true" }
    <nc:PersonFullName>Clark Kent</nc:PersonFullName>      |     },
  </nc:PersonName>                                         |     {
</nc:Person>                                               |       "nc:PersonFullName": "Clark Kent"
                                                           |     }
                                                           |   }
                                                           | }
```

Example 5-20: Example of a relationship property

NIEM uses RDF-star to represent relationship properties. Example 5-21 shows the RDF equivalent for the message in example 5-20. Figure 5-22 provides a diagram of that RDF graph.

```
_:n1 nc:PersonName _:n2 .
_:n1 nc:PersonName _:n2 {| "my:isSecret": "true" |} .
_:n2 nc:PersonFullName "Superman" .
_:n3 nc:PersonFullName "Clark Kent" .
```

Example 5-21: RDF-star equivalent for a relationship property

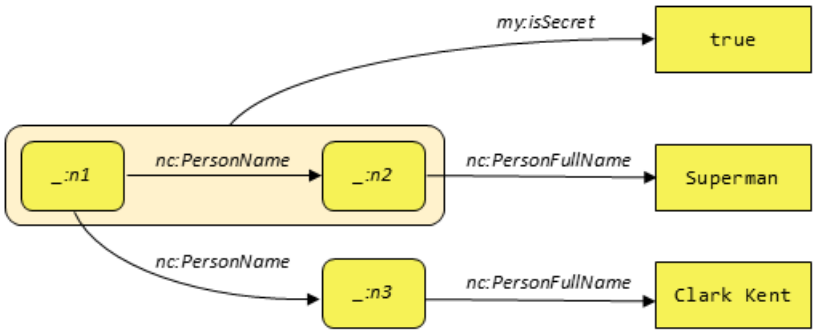


Figure 5-22: RDF-star graph diagram for a relationship property

5.6 Roles

These use `structures:uri` in NIEM 6. Need explanation and example TODO

5.6 Representation pattern

Stuff from NDR 5 section 10.7 TODO

5.7 Container objects

NDR 5 section 10.6 TODO

---

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 specified 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 satisfies all of the applicable rules in this document. 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 component names

  Section 7.2: Rules for component documentation

  Section 8: Rules for namespaces

  - *Reference namespace:* Additional rules for the reference namespace conformance target are in *section 9.8*.
  - *Extension namespace:* Additional rules for the extension namespace conformance target are in *section 9.9*.
  - *Subset namespace:* Additional rules for the subset namespace conformance target are in *section 9.10*.

- *Schema document*: A conforming schema document is a schema document that satisfies all of the applicable rules in this document. 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 9.1: Rules for the NIEM profile of XSD

  Section 9.2: Rules for type definitions

  Section 9.3: Rules for attribute and element declarations

  Section 9.4: Rules for adapters and external components

Section 9.5: Rules for proxy types

Section 9.6: Rules for augmentations

Section 9.7: Rules for machine-readable annotations

- *Reference schema document:* Additional rules for the reference schema document conformance target are in *section 9.8*.
- *Extension schema document:* Additional rules for the extension schema document conformance target are in *section 9.9*.
- *Subset schema document:* Additional rules for the subset schema document conformance target are in *section 9.9*.

- *Model*: A conforming model fulfils all of the rules in section 10. 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 10.1.
  - *Schema document set*: A conforming schema document set is a schema document set that fulfils all applicable rules in section 10. Additional rules for this conformance target are in section 10.2.
- *Message type and message format*: Rules for these conformance targets are in section 11
- *XML message*: Rules applying to a message in XML format are in section 12
- *JSON message:* Rules applying to a message in JSON format are in section 13

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-v3.0] 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-v3.0] 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 example 6-1 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/NDR/6.0/#ReferenceSchemaDocument"
  version="1"
  xml:lang="en-US">
```

Example 6-1: 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 example 6-1 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/NDR/6.0/#ReferenceSchemaDocument
  </ConformanceTargetURI>
  <NIEMVersionText>6</NIEMVersionText>
  <NamespaceVersionText>1</NamespaceVersionText>
  <NamespaceLanguageName>en-US</NamespaceLanguageName>
</Namespace>
```

Example 6-2: Conformance target assertion in CMF

6.3 Conformance testing

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

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

---

7. Rules for model components

These rules apply to model components in both the CMF and XSD representations ofconforming 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 insection 7.1
- Rules for documentation of components appear insection 7.2
- Rules for namespaces appear in section 8

7.1 Rules for component names

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.

Rule 7-1: Attribute and element do not have same uncased name || A namespace MUST NOT include two components with the same uncased name. (NEW)

For example, a namespace may not include both the attribute `commentText` and the element `CommentText`. This would cause problems in case-insensitive environments.

7.1.1 Rules based on kind of component

Rule 7-2: Name of Class, Datatype, and Property components || Class and Datatype components MUST have a name ending in “Type”; Property components MUST NOT. (N5R 11-1,11-2)

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 7-3: Augmentation names are reserved || A component MUST NOT have a name ending in “Augmentation”, “AugmentationPoint”, or “AugmentationType”. (NEW)

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 7-4: Name of adapter classes || An adapter class MUST have a name ending in “AdapterType”; all other components MUST NOT. (NEW)

Rule 7-5: Name of association classes || An association class MUST have a name ending in “AssociationType”; all other components MUST NOT. (N5R 10-21)

Rule 7-6: Name of code list literal classes || A literal class with a literal property that has a code list datatype MUST have a name ending in “CodeType”; all other literal classes MUST NOT. (N5R 10-17,10-18)

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 7-7: Names ending in “SimpleType” || A component with a name ending in “SimpleType” MUST be a Datatype. (NEW)

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 7-8: Names ending in “CodeSimpleType” || A Datatype object with a name that ends in “CodeSimpleType” MUST be a code list datatype. (N5R 11-8,11-9)

Rule 7-9: Name of code list datatypes || A code list datatype MUST have a name ending in “CodeType” or “CodeSimpleType”; all other Datatype components MUST NOT. (N5R 10-17,10-18)

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

7.1.1.3 Rules for names of Property components

Rule 7-10: Name of abstract properties || 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. (N5R 10-42,11-14)

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 7-11: Name of association properties || A Property with an association class MUST have a name ending in “Association”; all other components MUST NOT. (N5R 10-22)

Rule 7-12: Name of code properties || A Property with a Class or Datatype that represents a code list MUST have a name ending in “Code”; all other components MUST NOT. (N5R 10-19,11-10)

Rule 7-13: Name of literal properties in CMF || The literal property of a literal class MUST have a name ending in “Literal”; all other components MUST NOT. (NEW)

Component names ending in “Literal” only occur in the CMF representation of a literal class.

Rule 7-14: Name of representation attributes || A Property that is a reference attribute property property MUST have a name ending in “Ref”; all other components MUST NOT. (NEW)

7.1.2 Rules for composition of component names

Rule 7-15: Component name composed of English words || 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]. (N5R 10-44)

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. NIEM allows (but does not encourage) component names that are not from the English language in extension schema documents.

Rule 7-16: Component names have only specific characters || The name of a model component MUST be entirely composed of specified characters. (N5R 10-46)

- 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 7-17: Component names use camel case || The name of a model component MUST use the camel case formatting convention. (N5R 10-48)

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 7-18: Name of attribute properties begin with lower case letter || The name of anattribute property MUST begin with a lowercase character. (N5R 10-49)

Rule 7-19: Name of components other than attribute properties begin with upper case letter || The name of a model component that is not an attribute property MUST begin with an uppercase character. (N5R 10-50)

Rule 7-20: Punctuation in component name is a separator || 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. (N5R 10-47)

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 7-21: Singular form is preferred in name || 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. (N5R 10-54)

Rule 7-22: Present tense is preferred in name || 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. (N5R 10-55)

Rule 7-23: Name does not have nonessential words || Articles, conjunctions, and prepositions MUST NOT be used in NIEM component names except where they are required for clarity or by standard convention. (N5R 10-56)

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 naming 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 7-24: Property name follows ISO 11179-5 pattern || 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]. (N5R 7-5,10-57)

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 7-25: Object-class term identifies concrete category || The object-class term of a NIEM component MUST consist of a term identifying a category of concepts or entities. (N5R 10-58)

NIEM adopts an object-oriented approach to representation of data. Object classes represent whatISO 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 concept term is Activity. An example of an entity term is Vehicle.

7.1.4.2 Property term

Rule 7-26: Property term describes characteristic or subpart || A property term MUST describe or represent a characteristic or subpart of an entity or concept. (N5R 10-59)

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 Qualifier terms

Rule 7-27: Name may have multiple qualifier terms || Multiple qualifier terms MAY be used within a component name as necessary to ensure clarity and uniqueness within its namespace and usage context. (N5R 10-60)

Rule 7-28: Name avoids unnecessary qualifier terms || The number of qualifier terms SHOULD be limited to the absolute minimum required to make the component name unique and understandable. (N5R 10-61)

Rule 7-29: Order of qualifiers is not significant || The order of qualifiers MUST NOT be used to differentiate components. (N5R 10-62)

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.

PrimaryRepresentationTerm	SecondaryRepresentationTerm	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.

PrimaryRepresentationTerm	SecondaryRepresentationTerm	Definition
	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.
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

Table 7-1: Property representation terms

Rule 7-30: Redundant term in name is omitted || If any word in the representation term is redundant with any word in the property term, one occurrence SHOULD be deleted. (N5R 10-63)

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 7-31: Data property uses representation term || The name of a data property SHOULD use an appropriate representation term as found in table 6-1, Property representation terms. (N5R 10-64, 11-15, 11-16, 11-19)

Rule 7-32: Object property uses representation term when appropriate || 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. (N5R 10-65)

Rule 7-33: Object property uses representation term only when appropriate || 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. (N5R 10-66)

7.1.5 Acronyms, abbreviations, and jargon

Rule 7-34: Names use common abbreviations || A component name SHOULD use the abbreviations shown in the table below. (N5R 10-51)

Abbreviation	Full Meaning
ID	Identifier
URI	Uniform Resource Identifier

Rule 7-35: Local terms usable within their namespace || A local term MAY be used in the name of a component within its namespace. (N5R 10-52)

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

Rule 7-36: Local term has literal or definition || 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. (N5R 10-77)

7.2 Rules for component documentation

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 7-37: Namespace has data definition || In CMF, a Namespace object MUST be a documented component. In XSD, the `xs:schema` element MUST be a documented component. (N5R 9-82)

Rule 7-38: Model component has data definition || 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 (N5R 9-12,9-26,9-37,9-49)

Rule 7-39: Enumeration facet has data definition || In CMF, a Facet object with a FacetCategoryCode of `enumeration` MUST be a documented component. In XSD, an `xs:enumeration` facet MUST be a documented component. (N5R 9-14)

Rule 7-40: Pattern facet has data definition || In CMF, a Facet object with a FacetCategoryCode of `pattern` MUST be a documented component. In XSD, an `xs:pattern` facet MUST be a documented component. (NEW)

Rule 7-41: Documentation is provided in US English || 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 occurrence of `xs:documentation` within `xs:annotation` MUST be within the scope of an occurrence of `xml:lang` with a value of `en-US`. In each case, subsequent instances, if provided, MUST have a different language name. (NEW)

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 7-42: Data definition does not introduce ambiguity || 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. (N5R 11-24)

Rule 7-43: Object class has only one meaning || 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. (N5R 11-25)

Rule 7-44: Data definition of a part does not redefine the whole || An object class MUST NOT be redefined within the definitions of the components that represent properties or subparts of that entity or class. (N5R 11-26)

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 7-45: Do not leak representation into data definition || 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. (N5R 11-27)

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 definition rules from ISO 11179-4

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

Rule 7-46: Data definition follows 11179-4 requirements || 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: (N5R 11-28)

- 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 7-47: Data definition follows 11179-4 recommendations || 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: (N5R 11-29)

- 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 7-48: Standard opening phrase for abstract property data definition || The data definition for an abstract property SHOULD begin with the standard opening phrase “A data concept…”. (N5R 11-35)

Rule 7-49: Standard opening phrase for association property data definition || 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)…”. (N5R 11-34)

Rule 7-50: Standard opening phrase for date property data definition || 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)…”. (N5R 11-36)

Rule 7-51: Standard opening phrase for quantity property data definition || The data definition for a property with a quantity representation term SHOULD begin with the standard opening phrase “An (optional adjectives) (count|number)…”. (N5R 11-37)

Rule 7-52: Standard opening phrase for picture property data definition || The data definition for a property with a picture representation term SHOULD begin with the standard opening phrase “An (optional adjectives) (image|picture|photograph)”. (N5R 11-38)

Rule 7-53: Standard opening phrase for indicator property data definition || The data definition for a property with an indicator representation term SHOULD begin with the standard opening phrase “True if …; false (otherwise|if)…”. (N5R 11-39)

Rule 7-54: Standard opening phrase for identification property data definition || The data definition for a property with an identification representation term SHOULD begin with the standard opening phrase “(A|An) (optional adjectives) identification…”. (N5R 11-40)

Rule 7-55: Standard opening phrase for name property data definition || The data definition for a property with a name representation term SHOULD begin with the standard opening phrase “(A|An) (optional adjectives) name…”. (N5R 11-41)

Rule 7-56: Standard opening phrase for property data definition || The data definition for a property SHOULD begin with the standard opening phrase “(A|An)”. (N5R 11-42)

7.2.4.2 Opening phrases for classes

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

Rule 7-57: Standard opening phrase for association class data definition || The data definition for an association class SHOULD begin with the standard opening phrase “A data type for (a relationship|an association)…”. (N5R 11-43)

Rule 7-58: Standard opening phrase for class data definition || The data definition for a class SHOULD begin with the standard opening phrase “A data type…” (N5R 11-46, 11-47)

7.3 Rules for specifications of components

Rule 7-59: Enumerations are unique || A Restriction object MUST NOT contain two Facet objects with a FacetCategoryCode of `enumeration` and the same FacetValue. (NEW)

8. Rules for namespaces

8.1 Rules for properties of namespaces

Rule 8-1: Namespace identifier is absolute URI || The namespace MUST have an identifier, which MUST match the grammar syntax `<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. (N5R 9-83, 9-84)

Rule 8-2: Namespace URI is owned by namespace authority || The namespace identifier must be a URI that is owned by the namespace author, as defined in [webarch] §2.2.2.1 *URI ownership*. (NEW)

For example, namespace authors must not choose a namespace URI beginning with https://docs.oasis-open.org/niemopen/ns/model/, because ownership of that URI has been delegated to the authors of the NIEM model.

Rule 8-3: Namespaces use slash URIs || The namespace SHOULD have an identifier ending in the slash ('/') character. (NEW)

Rule 8-4: Namespace URI includes version || The namespace SHOULD have an identifier ending in the pattern `/version/`, where *version* is a version identifier. (NEW)

Examples:

```
https://docs.oasis-open.org/niemopen/ns/model/niem-core/6.0/
http://example.com/myNS/1.0.1/
http://example.com/yourNS/1.1.1-alpha.7/
```

Rule 8-5: Namespace URI uses semantic versioning || The version identifier in a namespace identifier SHOULD conform to the [SemVer] specification. (NEW)

In semantic versioning, version numbers and the way they change convey meaning about the underlying code and what has been modified from one version to the next.

Rule 8-6: Namespace has a prefix || The namespace MUST have a defined prefix, which MUST match the grammar syntax `<NCName>` as defined by [XML Namespaces]. (NEW)

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 8-7: Namespace has version || 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. (N5R 9-85)

Rule 8-8: Namespace has language || 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. (N5R 10-45, 11-30)

8.2 Rules for reference namespaces

Rule 8-9: Reference namespace asserts conformance || A reference namespace MUST assert the conformance target identifier https://docs.oasis-open.org/niemopen/ns/specification/NDR/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 an effective conformance target identifier of the schema document (*see §6.2*). (N5R 4-1,4-5)

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

Rule 8-10: Reference namespace does not have wildcard || In CMF, a Class object with a Namespace that is a reference namespace MUST NOT have the AnyAttributeIndicator property or the AnyElementIndicator property with a value of “true”. In XSD, the schema document for the reference namespace MUST NOT contain the element `xs:any` or `xs:anyAttribute`. (N5R 9-70, 9-71)

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

Rule 8-11: Object properties in reference namespace are referenceable || 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`. (NEW)

To promote reuse, object properties defined in reference namespaces and extension namespaces are always referenceable. 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 8-12: Reference namespace uses reference namespace components || A component that is used in a reference namespace MUST be defined in a reference namespace. (N5R 11-50)

8.3 Rules for extension namespaces

Rule 8-13: Extension namespace asserts conformance || An extension namespace MUST assert the conformance target identifier https://docs.oasis-open.org/niemopen/ns/specification/NDR/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 an effective conformance target identifier of the schema document *(see §6.2)*. (N5R 4-2,4-6)

Rule 8-14: Object properties in extension namespace are referenceable || 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`. (NEW)

8.4 Rules for subset namespaces

Rule 8-15: Subset namespace asserts conformance || A subset namespace must assert the conformance target identifier https://docs.oasis-open.org/niemopen/ns/specification/NDR/6.0/#SubsetSchemaDocument. In CMF, this is a value of the ConformanceTargetURI property in the Namespace object. In XSD, this is an effective conformance target identifier of the schema document *(see §6.2)*. (N5R 4-5)

Rule 8-16: Subset has corresponding reference or extension namespace || A representation of a reference namespace or extension namespace with the same identifier as the subset namespace MUST exist. (NEW)

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 8-17: Subset does not extend component range || A subset namespace MUST NOT extend the valid range of a component in the corresponding reference namespace or extension namespace. (NEW)

Rule 8-18: Subset does not add components || 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. (NEW)

Rule 8-19: Subset does not alter data definition || 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. (NEW)

The previous three rules together make up the subset rule: Any data that is valid for a subset namespace must also be valid for its reference namespace or extension namespace, and must have the same meaning.

9. 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 9-1: Schema is CTAS-conformant || The schema document MUST be a conformant document as defined by [CTAS-v3.0]. (N5R 4-3)

Rule 9-2: Document element has attribute `ct:conformanceTargets` || 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. (N5R 4-4)

9.1 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 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 9.4: 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 9-3: Document is a valid schema document || The XSD representation of a namespace MUST be a schema document, as defined by [XML Schema Structures]. (N5R 7-1,7-2,7-3)

Rule 9-4: Document element is `xs:schema` || The document element of the XSD representation of a namespace MUST be `xs:schema`. (N5R 7-4)

Rule 9-5: Prohibited schema components || A schema document MUST NOT contain any of the following elements: (N5R 9-59,9-61,9-72,9-73,9-74,9-75,9-76,9-88,9-89)

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

Rule 9-6: Prohibited base types || A schema component MUST NOT have an attribute `{}base` with a value of any of these types: (N5R 9-1,9-2,9-3,9-4,9-5,9-6,9-7,9-8,9-9)

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

Rule 9-7: Prohibited list item types || A schema component MUST NOT have an attribute `{}itemType` with any of the following values: (N5R 9-15,9-16,9-17,9-18)

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

Rule 9-8: Prohibited union item types || A schema component MUST NOT have an attribute `{}memberTypes` with any of the following values: (N5R 9-19,9-20,9-21,9-22,9-23,9-24)

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

Rule 9-9: Prohibited attribute and element types || A schema component MUST NOT have an attribute `{}type` with any of the following types: (N5R 9-51,9-52,9-53,9-54,9-55,9-56)

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

Rule 9-88 also forbids the type `xs:IDREFS` for all schema components other thanreference attribute properties.

Rule 9-10: No mixed content on complex type or complex content || A complex type definition MUST NOT have mixed content. (N5R 9-27,9-28)

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 9-11: Complex type content is explicitly simple or complex || A complex type definition MUST have a `xs:complexContent` or a `xs:simpleContent` child element (N5R 9-29)

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 9-12: Base type of complex type with complex content must have complex content || The base type of a complex type with complex content MUST have complex content. (N5R 9-31,9-32)

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 9-13: Untyped element is abstract || An untyped element or an element of type `xs:anySimpleType` MUST be abstract. (N5R 9-38,9-39)

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 9-14: Element type not in the XML or XML Schema namespace || An element type MUST NOT be in the XML Schema namespace or the XML namespace. (N5R 9-40,9-41)

Rule 9-15: Element type is not simple type || An element type that is not `xs:anySimpleType` MUST NOT be a simple type. (N5R 9-42)

Rule 9-16: Attribute declaration has type || An attribute declaration MUST have a type. (N5R 9-50)

Rule 9-17: No default or fixed value || An element declaration MUST NOT have an attribute `{}default` or `{}fixed`. (N5R 9-45,9-46,9-57,9-58)

Rule 9-18: Sequence has minimum and maximum cardinality 1 || An element `xs:sequence` MUST have a `minOccurs` and `maxOccurs` of 1. (N5R 9-66,9-67)

Rule 9-19: `xs:choice` must be child of `xs:sequence` || An element `xs:choice` MUST be a child of `xs:sequence`. (N5R 9-65)

Rule 9-20: Choice has minimum and maximum cardinality 1 || An element `xs:choice` MUST have a `minOccurs` and `maxOccurs` of 1. (N5R 9-68,9-69)

Rule 9-21: Comment is not recommended || An XML comment SHOULD NOT appear in the schema. (N5R 9-77)

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 9-22: Documentation element has no element children || A child of element `xs:documentation` MUST be text or an XML comment. (N5R 9-78)

Rule 9-23: Import has namespace || An element `xs:import` MUST have an attribute `{}namespace`. (N5R 9-90)

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 9-24: Import specifies local resource || An element `xs:import` MUST specify a schema document, which MUST be a local resource. (NEW)

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.

9.2 Rules for XSD types

This section provides rules for *type definitions* in the XSD representation of a model. A type definition in XML Schema can create a complex data type - a type for elements with child elements - with `xs:complexType`). It can also create a simple data type, a type for elements with a literal value, with `xs:simpleType`.

Rule 9-25: Name of type definitions || A type definition that does not define a *proxy type* MUST have a name ending in "Type"; all other XSD components MUST NOT. (N5R 11-2)

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* §9.5).

Rule 9-26: Name of simple type definitions || A simple type definition MUST have a name ending in "SimpleType"; all other XSD components MUST NOT. (N5R 11-4)

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 9-27: Name of complex type definition || A complex type definition MUST be a Class component, a Datatype component, or a proxy type. (NEW)

Rule 9-28: `xs:sequence` must be child of `xs:extension` || An element `xs:sequence` MUST be a child of `xs:extension`. (N5R 9-62)

Rule 9-29: `xs:sequence` must be child of `xs:extension` or `xs:restriction` || An element `xs:sequence` MUST be a child of `xs:extension` or `xs:restriction`. (N5R 9-63)

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

Rule 9-30: Type definition is top-level || A type definition MUST be top-level. (N5R 9-10,9-25)

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 9-31: Complex type has a category || A complex type definition MUST be an object type, an association type, an adapter type, or an augmentation type. (N5R 10-1)

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 9-32: Object type with complex content is derived from `structures:ObjectType` || 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. (N5R 10-2)

Rule 9-33: Adapter type derived from `structures:AdapterType` || A type definition that represents an adapter class MUST be derived from `structures:AdapterType`. (NEW)

Rule 9-34: Association type derived from `structures:AssociationType` || A type definition that represents an association class MUST be derived from `structures:AssociationType` or from another association class. (N5R 10-21)

Rule 9-35: Augmentation type derived from `structures:AugmentationType` || A type definition that is an augmentation type MUST be derived from `structures:AugmentationType`. (N5R 10-35)

Rule 9-36: Complex type with simple content has `structures:SimpleObjectAttributeGroup` || A complex type definition with simple content MUST include `structures:SimpleObjectAttributeGroup`. (N5R 11-11)

Rule 9-37: Base type definition defined by conformant schema || 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. (N5R 11-3)

Rule 9-38: Component reference defined by conformant schema || An attribute or element reference MUST have the target namespace or a namespace that is imported as conformant. (N5R 11-21,11-22)

Rule 9-39: Schema uses only known attribute groups || An attribute group reference MUST be `structures:SimpleObjectAttributeGroup`. (N5R 11-23)

The use of attribute groups is restricted in a conforming schema document. 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 9-40: Augmentation elements are not used directly || A complex type definition MUST NOT have an element use of an augmentation element declaration, or an element declaration that is in the substitution group of an augmentation point element declaration. (N5R 10-37)

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

Rule 9-41: List item type defined by conformant schemas || 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. (N5R 11-6)

Rule 9-42: Union member types defined by conformant schemas || 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. (N5R 11-7)

Rule 9-43: No complex wildcards || A complex type definition MUST not contain the element `xs:any` or `xs:anyAttribute` that has a `namespace` or `processContents` attribute. (NEW)

Restrictions on attribute and element wildcards, if desired, must be enforced through some mechanism other than XML Schema validation.

9.3 Rules for attribute and element declarations

Rule 9-44: No literal properties in XSD || The name of an element declaration or attribute declaration MUST NOT end in "Literal". (NEW)

Literal properties appear only in the CMF representation of a literal class.

Rule 9-45: Declarations are top-level || An attribute declaration or element declaration MUST be top-level. (N5R 9-36,9-48)

Rule 9-46: Element type is not simple type || An element declaration MUST NOT have a simple type. (N5R 9-42,11-12)

Rule 9-47: Attribute and element type is from conformant namespace || 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. (N5R 11-13,11-18)

Rule 9-48: Element substitution group defined by conformant schema || An element substitution group MUST have either the target namespace or a namespace that is imported as conformant. (N5R 11-17)

Rule 9-49: Attribute and element type not from structures namespace || An attribute declaration or element declaration MUST NOT have a type from the structures namespace. (NEW)

Rule 9-50: Only reference attributes have type `xs:IDREFS` || The attribute declaration of a reference attribute property MUST have type `xs:IDREFS`; all other attribute and element declarations MUST NOT. (NEW)

Reference attribute properties are a special form of object reference; see §5.3.6.

9.4 Rules for adapters and external components

Rule 9-51: Import of external schema document is labeled || An `xs:import` element importing an external schema document MUST own the attribute `appinfo:externalImportIndicator` with a value of `true`. (NEW)

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 adapter type, and which is defined by an extension schema document or subset schema document, may incorporate an externally-defined attribute.

Rule 9-52: Import of external namespace has data definition || An `xs:import` element importing an external schema document MUST be a documented component. (N5R 10-7)

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 9-53: Name of adapter type || An adapter type MUST have a name ending in “AdapterType”; all other type definitions MUST NOT. (N5R 10-8)

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

An adapter type should contain the information from an external standard to express a complete concept. This expression should be composed of content entirely from an external schema 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 adapter type may use those components.

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

Rule 9-54: Structure of external adapter type definition follows pattern || An 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. (N5R 10-9)

Rule 9-55: Element use from external adapter type defined by external schema documents || An element reference that appears within an adapter type MUST have a target namespace that is imported as external. (N5R 10-10)

Rule 9-56: External adapter type not a base type || An adapter type definition MUST NOT be a base type definition. (N5R 10-11, 10-12)

Rule 9-57: External attribute use has data definition || An external attribute use MUST be a documented component with a non-empty data definition. (N5R 10-14)

Rule 9-58: External attribute use not an ID || An attribute use schema component MUST NOT have an attribute declaration with an ID type. (N5R 10-15)

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 9-59: External element use has data definition || An external attribute use MUST be a documented component with a non-empty data definition. (N5R 10-16)

9.5 Rules for proxy types

Rule 9-60: Proxy types || The XSD declaration of a proxy type MUST have the same name as the simple type it extends. (N5R 10-20)

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 wrapper 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 9-61: Proxy type has designated structure || 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`. (N5R 10-20)

9.6 Rules for augmentations

Rule 9-62: Name of augmentation types || The XSD definition of an augmentation type MUST have a name ending in “AugmentationType”; all other XSD components MUST NOT. (N5R 10-33,10-34)

Rule 9-63: Name of augmentation elements || The XSD declaration of an augmentation element MUST have a name ending in “Augmentation”; all other XSD components MUST NOT. (N5R 10-36)

Rule 9-64: Name of augmentation point elements || The XSD declaration of an augmentation point element MUST have a name ending in “AugmentationPoint”; all other XSD components MUST NOT. (NEW)

Rule 9-65: Standard opening phrase for augmentation point element data definition || The data definition for an augmentation point element SHOULD begin with standard opening phrase “An augmentation point…”. (N5R 11-31)

Rule 9-66: Standard opening phrase for augmentation element data definition || The data definition for an augmentation element SHOULD begin with the standard opening phrase “Supplements…” or “Additional information about…”. (N5R 11-32)

Rule 9-67: Standard opening phrase for augmentation type data definition || The data definition for an augmentation type SHOULD begin with the standard opening phrase “A data type (that supplements|for additional information about)…”. (N5R 11-44)

Rule 9-68: Augmentation point element corresponds to its base type || A schema document containing an element declaration for an augmentation point element MUST also contain a type definition for its augmented base type. (N5R 10-25)

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

Rule 9-69: An augmentation point element has no type || An augmentation point element MUST have no type. (N5R 10-26)

Rule 9-70: An augmentation point element has no substitution group || An augmentation point element MUST have no substitution group. (N5R 10-27)

Rule 9-71: Augmentation point element is only referenced by its base type || An augmentation point element MUST only be referenced by its base type. (N5R 10-28)

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

Rule 9-72: Augmentation point element use is optional and unbounded || An augmentation point element particle MUST have attribute `minOccurs` equal to 0 and attribute `maxOccurs` set to unbounded. (N5R 10-29,10-30)

Rule 9-73: Augmentation point element use must be last element in its base type || An augmentation point element particle MUST be the last element occurrence in the content model of its augmentable type. (N5R 10-31)

9.7 Rules for machine-readable annotations

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 9-74: Appinfo attribute annotates schema component || An attribute in the appinfo namespace MUST be owned by an element with a namespace name http://www.w3.org/2001/XMLSchema. (N5R 10-69)

Rule 9-75: `xs:appinfo` children are comments, elements, or whitespace || A child of element `xs:appinfo` MUST be an element, a comment, or whitespace text. (N5R 9-79)

Rule 9-76: Appinfo child elements have namespaces || An element that is a child of `xs:appinfo` MUST have a namespace name. (N5R 9-80)

Rule 9-77: Appinfo descendants are not XML Schema elements || An element that is a descendent of `xs:appinfo` MUST NOT have the XML Schema namespace. (N5R 9-81)

Rule 9-78: Component marked as deprecated is deprecated component || A schema component that has an attribute `appinfo:deprecated` with a value of true MUST be a deprecated component. (N5R 10-68)

Rule 9-79: LocalTerm appinfo applies to schema || When the element `appinfo:LocalTerm` appears in a schema document, it MUST be application information on an element `xs:schema`. (N5R 10-76)

9.8 Rules for reference schema documents

Rule 9-80: No simple type disallowed derivation || A reference schema document MUST NOT have an attribute `{}final`. (N5R 9-11)

Rule 9-81: No use of “fixed” on simple type facets || A simple type constraining facet in a reference schema document MUST NOT have an attribute `{}fixed`. (N5R 9-13)

Rule 9-82: No disallowed substitutions || A reference schema document MUST NOT contain the attribute `{}block` or `{}blockDefault`. (N5R 9-34,9-43,9-86)

Rule 9-83: No disallowed derivation || A reference schema document MUST NOT contain the attribute `{}final` or `{}finalDefault`. (N5R 9-35,9-44,9-87)

Rule 9-84: Element declaration is nillable || An element declaration in a reference schema document MUST have the `{nillable}` property with a value of true. (N5R 9-47)

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

Rule 9-85: No `xs:choice` || A reference schema document MUST NOT contain the element `xs:choice`. (N5R 9-64)

Rule 9-86: External attribute use only in adapter type || An external attribute use within a reference schema document MUST be in an adapter type. (N5R 10-13)

9.9 Rules for extension schema documents

Rule 9-87: Element declaration is nillable || An element declaration in an extension schema document MUST have the `{nillable}` property with a value of true. (N5R 9-47)

9.10 Rules for subset schema documents

Rule 9-88: Attribute augmentations are documented || Within a message model, an attribute reference that does not appear in the corresponding reference schema document or extension schema document MUST have the attribute `appinfo:augmentingNamespace` containing the namespace prefix or URI of the augmenting namespace. (NEW)

Augmented XSD type definitions in a message model must include attribute augmentations so that the schema will validate all conforming messages. *(See section 4.15.6, Attribute augmentations in message models).*

---

10. Rules for models

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

Rule 10-1: Namespaces are conforming or external || Every namespace in a model MUST be one of the following: (NEW)

- 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.

The appinfo namespace is not part of a NIEM model. It provides schema components for use in the XSD representation of a NIEM model.

Rule 10-2: Unique namespace prefixes || A model MUST NOT contain two namespaces with the same prefix. (NEW)

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

10.1 Rules for model files

Rule 10-3: Unique namespace identifiers || A model MUST NOT contain two namespaces with the same identifier. (NEW)

This is impossible in an XSD representation of a model.

10.2 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 10-4: Composition of schema document set || The schema documents in a schema document set MUST be exactly those determined by the following procedure: (NEW)

- 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 10-5: Consistent import schema document || 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. (N5R 11-54)

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

and diagnose.

Rule 10-6: Consistent import labels || The members of a schema document set MUST NOT contain two `xs:import` elements with the same namespace but different values for `appinfo:externalImportIndicator`. (N5R 11-55)

Rule 10-7: Consistent import documentation || The members of a schema document set MUST NOT contain two `xs:import` elements with non-empty data definitions that are different. (NEW)

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 10-8: Namespace prefix is unique || There MUST be a one-to-one match between namespace prefix and namespace URI among all the members of a schema document set. (NEW)

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 10-9: Schema document set must be complete || 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. (N5R 9-91)

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 satisfy this rule.

Rule 10-10: Use structures namespace consistent with specification || A schema document set MUST include the structures namespace as it is defined in Appendix B of this document. (N5R 10-78)

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.

---

11. Rules for message types and message formats

Rule 11-1: Message type declares initial property || A message type MUST declare the initial property of conforming messages. (NEW)

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

Rule 11-2: Message format schema matches message type || The schema for a message format MUST validate exactly those messages that conform to the format's message type. (NEW)

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.

---

12. Rules for XML messages

Rule 12-1: Message begins with initial property || An XML message MUST be an XML document that contains one instance of the element for the initial property specified by its message type, and all of the message content MUST be a descendent of that element. (NEW)

The element for the initial property is often the document element, but this is not necessarily so. An XML message may be embedded within an XML document; for example, as a payload within a SOAP response.

Rule 12-2: Message is schema-valid || An XML message MUST be schema-valid as assessed against the schema document set that represents the message model of a message type. (N5R 12-1)

This rule should not be construed to mean that XML validation must be performed on all XML instances as they are served or consumed; only that the XML instances validate if XML validation is performed. The XML Schema component definitions specify XML documents and element information items, and the instances should follow the rules given by the schemas, even when validation is not performed.

Rule 12-3: No attributes from wildcards in structures || Every attribute in an XML message MUST be valid by virtue of an `xs:attribute` element in a conforming schema document. An XML message MUST NOT contain an attribute that is schema-valid only by virtue of an `xs:anyAttribute` element in the structures namespace. (NEW)

The schema document for the structures namespace contains `xs:anyAttribute` elements for the purpose of attribute augmentation. This permits a message designer to augment his subset of a reference schema document or extension schema document with one or more attribute properties, while still following the subset rule. This does *not* permit any element in a conforming message to contain any attribute defined in the *message model*.

Rule 12-4: No forbidden references || An element in an XML message MUST NOT have the attribute `structures:id` if its element declaration or type definition has the attribute `appinfo:referenceCode` with a value of `NONE`. (NEW)

Rule 12-5: No forbidden references || An element in an XML message MUST NOT have the attribute `structures:ref` if its element declaration or type definition has the attribute `appinfo:referenceCode` with a value of `NONE` or `URI`. (NEW)

Rule 12-6: No forbidden references || An element in an XML message MUST NOT have the attribute `structures:uri` if its element declaration or type definition has the attribute `appinfo:referenceCode` with a value of `NONE` or `REF`. (NEW)

Rule 12-7: Element has only one resource identifying attribute || An element in an XML message MUST NOT have more than one attribute that is `structures:id`, `structures:ref`, or `structures:uri`. (N5R 12-3)

Model designers may use `appinfo:referenceCode` in the XSD representation of a model to constrain the permissible kinds of reference to objects of a specified class or property. For example:

```
<xs:complexType name="nc:PersonType" appinfo:referenceCode="NONE">
```

declares that objects of that class may not be the target of an object reference, and must instead appear inline. Conforming messages must follow those constraints.

Rule 12-8: Attribute `structures:ref` must reference `structures:id` || The value of an attribute `structures:ref` MUST match the value of an attribute `structures:id` of some element in the XML message. (N5R 12-4)

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 12-9: Linked elements have same validation root || Every element that has an attribute `structures:ref` MUST have a referencing element validation root that is equal to the referenced element validation root. (N5R 12-5)

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 12-10: Attribute `structures:ref` references element of correct type || An element that is the target of a `structures:ref` object reference MUST have a type that is validly derived from the type of the referencing element. (N5R 12-6)

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.

Rule 12-11: Reference attribute property refers to correct class || An element that is the target of a reference attribute property MUST have a type with a name that is the QName of the property, with the local part capitalized, and the trailing "Ref" replaced with "Type", or a derived type. (NEW)

For example, an element that is the target of `nc:metadataRef` must have the type `nc:MetadataType`, or a derived type.

Rule 12-12: `xs:anyURI` value must be valid URI || The value of an attribute with or derived from `xs:anyURI` MUST satisfy the grammar syntax `<URI-reference>` as defined by [RFC 3986]. (NEW)

XML Schema validation does not always check the validity of URI values. Examples of valid and invalid URI attributes:

```
structures:uri="http://example.com/Person/223/"
<-- valid
structures:uri="#boogala"                    <-- valid
structures:uri="boogala"                     <-- invalid
```

Rule 12-13: No duplicate augmentation elements || An element MUST NOT contain two instances of the same augmentation element. (NEW)

For example, a message must not contain

```
<nc:Person>
  <j:PersonAugmentation>...
  <j:PersonAugmentation>...
</nc:Person>
```

even though this is schema-valid. Instead, all augmentation properties should be consolidated into a single `j:PersonAugmentation` element.

Rule 12-14: Nilled element must be an object reference || An element with `xsi:nil="true"` MUST have the attribute `structures:ref` or `structures:uri`. (NEW)

The attribute `xsi:nil` can only be used to create an object reference. It cannot be used to omit mandatory content.

13. Rules for JSON messages

Rule 13-1: Message is a JSON object || A JSON message MUST be valid according to the grammar syntax `<object>` as defined by [RFC 8259]. (NEW)

According to the JSON specification, a valid JSON text can be an object, array, number, string, or literal name. Only the first of these is allowed as a NIEM JSON message.

Rule 13-2: Message is a JSON-LD document || A JSON message MUST conform to the JSON-LD specification in [JSON-LD]. (NEW)

Rule 13-3: Message conforms to message format || A JSON message MUST be valid when assessed against the schema of its message format. (NEW)

The schema for a JSON message format is expressed in JSON Schema, and validates exactly those messages that conform to the message type. *(see rule 11-2.)*

Rule 13-4: Message has context map for model namespaces || A JSON message MUST have an embedded context, remote context, or context via HTTP header. The context MUST map each namespace prefix in the message model to its corresponding namespace URI. The URL for a remote context MUST be an absolute URI. (NEW)

Embedded context, remote context, and context via HTTP header are defined in[JSON-LD] §3.1: The Context.

For example, the JSON message in example 3-2 has a context that maps the prefixes `nc` and `msg` to their corresponding URIs.

Rule 13-5: Object keys are defined || The name in a name-value mapping within a JSON object MUST be a JSON-LD keyword, or a term that expands to the URI of a property in the message model. (NEW)

For example:

```
"@context": {
  "nc": "https://docs.oasis-open.org/niemopen/ns/model/niem-core/6.0/",
  "pname": "nc:PersonName"},
"nc:Person": {
valid, expands to https://docs.oasis-open.org/niemopen/ns/model/niem-core/6.0/Person
  "@id": "#JD",  valid, JSON-LD keyword
  "pname": {     valid, expands to https://docs.oasis-open.org/niemopen/ns/model/niem-core/6.0/PersonName
    "foo:FullName": "John Doe"    invalid, no mapping for "foo" prefix
  }
}
```

Rule 13-6: @id keyword is object reference || Two JSON objects with the same value for the `@id` key MUST represent the value of the same message object. (NEW)

Rule 13-7: No forbidden references || A JSON object representing the value of a model Property object with an effective ReferenceCode of `NONE` MUST NOT contain the `@id` key. (NEW)

Rule 13-8: Linked objects have compatible class || Two JSON objects with the same value for the `@id` key MUST represent message objects having the same class or common class ancestor. (NEW)

For example, the following NIEM JSON is valid, because `nc:Item` and `nc:Equipment` have the same class `nc:ItemType`.

```
"nc:Item": {
  "@id": "#ITEM7",
  "nc:ItemQuantity": 7
},
"nc:Equipment": {
  "@id": "#ITEM7",
  "nc:EquipmentName": "Pump"
}
```

14. RDF interpretation of NIEM models and messages

TODO

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 fromhttp://www.jclark.com/xml/xmlns.htm.

[CMF]

Common Model Format Specification, NIEM Technical Architecture Committee. Available from https://github.com/niemopen/common-model-format.

[Code Lists]

Roberts, W. “NIEM Code Lists Specification”. NIEM Technical Architecture Committee (NTAC), November 7, 2017. Available from https://reference.niem.gov/niem/specification/code-lists/4.0/niem-code-lists-4.0.html.

[CTAS-v3.0]

Conformance Targets Attribute Specification (CTAS) Version 3.0. Edited by Tom Carlson. 22 February 2023. OASIS Project Specification 01. https://docs.oasis-open.org/niemopen/ctas/v3.0/ps01/ctas-v3.0-ps01.html. Latest stage: https://docs.oasis-open.org/niemopen/ctas/v3.0/ctas-v3.0.html.

[ISO 11179-4]

“ISO/IEC 11179-4 Information Technology — Metadata Registries (MDR) — Part 4: Formulation of Data Definitions Second Edition”, 15 July 2004.

[ISO 11179-5]

“ISO/IEC 11179-5:2005, Information technology — Metadata registries (MDR) — Part 5: Naming and identification principles”.

[JSON-LD]

Sporny, M., et al. “JSON-LD 1.1: A JSON-based Serialization for Linked Data”. W3C Recommendation, 16 July 2020. https://www.w3.org/TR/json-ld11/.

[OED]

Oxford English Dictionary, Third Edition, Oxford University Press, November 2010.http://dictionary.oed.com/.

[RFC 2119]

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.

[RFC 3986]

Berners-Lee, T., et al., “Uniform Resource Identifier (URI): Generic Syntax”, Request for Comments 3986, January 2005. http://tools.ietf.org/html/rfc3986.

[RFC 8174]

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.

[RFC 8259]

Bray, T. “The JavaScript Object Notation (JSON) Data Interchange Format”, Request for Comments 8259, December 2017. https://www.rfc-editor.org/rfc/rfc8259.

[SemVer]

“Semantic Versioning 2.0.0”. https://semver.org/.

[XML]

“Extensible Markup Language (XML) 1.0 (Fourth Edition)”, W3C Recommendation, 16 August 2006. Available from http://www.w3.org/TR/2008/REC-xml-20081126/.

[XML Infoset]

Cowan, John, and Richard Tobin. “XML Information Set (Second Edition)”, 4 February 2004. http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.

[XML Namespaces]

“Namespaces in XML 1.0 (Third Edition)”, W3C Recommendation, 8 December 2009. Available from http://www.w3.org/TR/2009/REC-xml-names-20091208/.

[XML Schema Structures]

“XML Schema Part 1: Structures Second Edition”, W3C Recommendation, 28 October 2004. Available from http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

[webarch]

Jacobs, I. “Architecture of the World Wide Web, Volume One”. W3C Recommendation 15 December 2004. https://www.w3.org/TR/webarch/.

A.2 Informative References

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
  </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 fo
    </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 relat
    </xs:annotation>
    <xs:sequence>
      <xs:element ref="structures:AssociationAugmentationPoint" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="structures:appliesToParent"/>
```



```
    <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: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:
    </xs:annotation>
  </xs:attribute>
</xs:schema>
```

Appendix C. Index of rules

- Rule 8-7: Namespace has version.
- Rule 8-8: Namespace has language.
- Rule 8-9: Reference namespace asserts conformance.
- Rule 8-10: Reference namespace does not have wildcard.
- Rule 8-11: Object properties in reference namespace are referenceable
- Rule 8-12: Reference namespace uses reference namespace components.
- Rule 8-13: Extension namespace asserts conformance.
- Rule 8-14: Object properties in extension namespace are referenceable
- Rule 8-15: Subset namespace asserts conformance.
- Rule 8-16: Subset has corresponding reference or extension namespace
- Rule 8-17: Subset does not extend component range
- Rule 8-18: Subset does not add components.
- Rule 8-19: Subset does not alter data definition
- Rule 9-1: Schema is CTAS-conformant.
- Rule 9-2: Document element has attribute `ct:conformanceTargets`.
- Rule 9-3: Document is a valid schema document
- Rule 9-4: Document element is `xs:schema`.
- Rule 9-5: Prohibited schema components.
- Rule 9-6: Prohibited base types.
- Rule 9-7: Prohibited list item types.
- Rule 9-8: Prohibited union item types
- Rule 9-9: Prohibited attribute and element types.
- Rule 9-10: No mixed content on complex type or complex content
- Rule 9-11: Complex type content is explicitly simple or complex
- Rule 9-12: Base type of complex type with complex content must have complex content
- Rule 9-13: Untyped element is abstract.
- Rule 9-14: Element type not in the XML or XML Schema namespace
- Rule 9-15: Element type is not simple type
- Rule 9-16: Attribute declaration has type.
- Rule 9-17: No default or fixed value
- Rule 9-18: Sequence has minimum and maximum cardinality 1
- Rule 9-19: `xs:choice` must be child of `xs:sequence`.
- Rule 9-20: Choice has minimum and maximum cardinality 1.
- Rule 9-21: Comment is not recommended
- Rule 9-22: Documentation element has no element children.
- Rule 9-23: Import has namespace.
- Rule 9-24: Import specifies local resource
- Rule 9-25: Name of type definitions.
- Rule 9-26: Name of simple type definitions
- Rule 9-27: Name of complex type definition
- Rule 9-28: `xs:sequence` must be child of `xs:extension`.
- Rule 9-29: `xs:sequence` must be child of `xs:extension` or `xs:restriction`.
- Rule 9-30: Type definition is top-level.
- Rule 9-31: Complex type has a category.
- Rule 9-32: Object type with complex content is derived from `structures:ObjectType`.
- Rule 9-33: Adapter type derived from `structures:AdapterType`.
- Rule 9-34: Association type derived from `structures:AssociationType`.
- Rule 9-35: Augmentation type derived from `structures:AugmentationType`.
- Rule 9-36: Complex type with simple content has `structures:SimpleObjectAttributeGroup`.
- Rule 9-37: Base type definition defined by conformant schema
- Rule 9-38: Component reference defined by conformant schema
- Rule 9-39: Schema uses only known attribute groups
- Rule 9-40: Augmentation elements are not used directly.
- Rule 9-41: List item type defined by conformant schemas
- Rule 9-42: Union member types defined by conformant schemas
- Rule 9-43: No complex wildcards.
- Rule 9-44: No literal properties in XSD
- Rule 9-45: Declarations are top-level.

- Rule 12-4: No forbidden references.
- Rule 12-5: No forbidden references.
- Rule 12-6: No forbidden references.
- Rule 12-7: Element has only one resource identifying attribute.
- Rule 12-8: Attribute `structures:ref` must reference `structures:id`.
- Rule 12-9: Linked elements have same validation root.
- Rule 12-10: Attribute `structures:ref` references element of correct type.
- Rule 12-11: Reference attribute property refers to correct class.
- Rule 12-12: `xs:anyURI` value must be valid URI.
- Rule 12-13: No duplicate augmentation elements.
- Rule 12-14: Nilled element must be an object reference.
- Rule 13-1: Message is a JSON object.
- Rule 13-2: Message is a JSON-LD document.
- Rule 13-3: Message conforms to message format.
- Rule 13-4: Message has context map for model namespaces.
- Rule 13-5: Object keys are defined.
- Rule 13-6: @id keyword is object reference.
- Rule 13-7: No forbidden references.
- Rule 13-8: Linked objects have compatible class.

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	rule8-9
Rule 4-2, Schema marked as extension schema document must conform	rule8-13
Rule 4-3, Schema is CTAS-conformant	rule9-1
Rule 4-4, Document element has attribute ct:conformanceTargets	rule9-2
Rule 4-5, Schema claims reference schema conformance target	rule8-9, rule8-15
Rule 4-6, Schema claims extension conformance target	rule8-13
Rule 5-1, structures:uri denotes resource identifier	*no matching NIEM6 rule*
Rule 7-1, Document is an XML document	rule9-3
Rule 7-2, Document uses XML namespaces properly	rule9-3
Rule 7-3, Document is a schema document	rule9-3
Rule 7-4, Document element is xs:schema	rule9-4
Rule 7-5, Component name follows ISO 11179 Part 5 Annex A	rule7-24
Rule 9-1, No base type in the XML namespace	rule9-6
Rule 9-2, No base type of xs:ID	rule9-6
Rule 9-3, No base type of xs:IDREF	rule9-6
Rule 9-4, No base type of xs:IDREFS	rule9-6
Rule 9-5, No base type of xs:anyType	rule9-6

NIEM 5 Rule	NIEM 6 Rules
Rule 9-6, No base type of xs:anySimpleType	rule9-6
Rule 9-7, No base type of xs:NOTATION	rule9-6
Rule 9-8, No base type of xs:ENTITY	rule9-6
Rule 9-9, No base type of xs:ENTITIES	rule9-6
Rule 9-10, Simple type definition is top-level	rule9-30
Rule 9-11, No simple type disallowed derivation	rule9-80
Rule 9-12, Simple type has data definition	rule7-38
Rule 9-13, No use of fixed on simple type facets	rule9-81
Rule 9-14, Enumeration has data definition	rule7-39
Rule 9-15, No list item type of xs:ID	rule9-7
Rule 9-16, No list item type of xs:IDREF	rule9-7
Rule 9-17, No list item type of xs:anySimpleType	rule9-7
Rule 9-18, No list item type of xs:ENTITY	rule9-7
Rule 9-19, No union member types of xs:ID	rule9-8
Rule 9-20, No union member types of xs:IDREF	rule9-8
Rule 9-21, No union member types of xs:IDREFS	rule9-8
Rule 9-22, No union member types of xs:anySimpleType	rule9-8
Rule 9-23, No union member types of xs:ENTITY	rule9-8
Rule 9-24, No union member types of xs:ENTITIES	rule9-8
Rule 9-25, Complex type definition is top-level	rule9-30
Rule 9-26, Complex type has data definition	rule7-38
Rule 9-27, No mixed content on complex type	rule9-10
Rule 9-28, No mixed content on complex content	rule9-10
Rule 9-29, Complex type content is explicitly simple or complex	rule9-11
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	rule9-12
Rule 9-32, Base type of complex type with complex content must have complex content	rule9-12
Rule 9-33, Simple content uses extension	*no matching NIEM6 rule*

NIEM 5 Rule	NIEM 6 Rules
Rule 9-34, No complex type disallowed substitutions	rule9-82
Rule 9-35, No complex type disallowed derivation	rule9-83
Rule 9-36, Element declaration is top-level	rule9-45
Rule 9-37, Element declaration has data definition	rule7-38
Rule 9-38, Untyped element is abstract	rule9-13
Rule 9-39, Element of type xs:anySimpleType is abstract	rule9-13
Rule 9-40, Element type not in the XML Schema namespace	rule9-14
Rule 9-41, Element type not in the XML namespace	rule9-14
Rule 9-42, Element type is not simple type	rule9-15, rule9-46
Rule 9-43, No element disallowed substitutions	rule9-82
Rule 9-44, No element disallowed derivation	rule9-83
Rule 9-45, No element default value	rule9-17
Rule 9-46, No element fixed value	rule9-17
Rule 9-47, Element declaration is nillable	rule9-84, rule9-87
Rule 9-48, Attribute declaration is top-level	rule9-45
Rule 9-49, Attribute declaration has data definition	rule7-38
Rule 9-50, Attribute declaration has type	rule9-16
Rule 9-51, No attribute type of xs:ID	rule9-9
Rule 9-52, No attribute type of xs:IDREF	rule9-9
Rule 9-53, No attribute type of xs:IDREFS	rule9-9
Rule 9-54, No attribute type of xs:ENTITY	rule9-9
Rule 9-55, No attribute type of xs:ENTITIES	rule9-9
Rule 9-56, No attribute type of xs:anySimpleType	rule9-9
Rule 9-57, No attribute default values	rule9-17
Rule 9-58, No fixed values for optional attributes	rule9-17
Rule 9-59, No use of element xs:notation	rule9-5
Rule 9-60, Model group does not affect meaning	*no matching NIEM6 rule*
Rule 9-61, No xs:all	rule9-5

NIEM 5 Rule	NIEM 6 Rules
Rule 9-62, xs:sequence must be child of xs:extension	rule9-28
Rule 9-63, xs:sequence must be child of xs:extension or xs:restriction	rule9-29
Rule 9-64, No xs:choice	rule9-85
Rule 9-65, xs:choice must be child of xs:sequence	rule9-19
Rule 9-66, Sequence has minimum cardinality 1	rule9-18
Rule 9-67, Sequence has maximum cardinality 1	rule9-18
Rule 9-68, Choice has minimum cardinality 1	rule9-20
Rule 9-69, Choice has maximum cardinality 1	rule9-20
Rule 9-70, No use of xs:any	rule8-10
Rule 9-71, No use of xs:anyAttribute	rule8-10
Rule 9-72, No use of xs:unique	rule9-5
Rule 9-73, No use of xs:key	rule9-5
Rule 9-74, No use of xs:keyref	rule9-5
Rule 9-75, No use of xs:group	rule9-5
Rule 9-76, No definition of attribute groups	rule9-5
Rule 9-77, Comment is not recommended	rule9-21
Rule 9-78, Documentation element has no element children	rule9-22
Rule 9-79, xs:appinfo children are comments, elements, or whitespace	rule9-75
Rule 9-80, Appinfo child elements have namespaces	rule9-76
Rule 9-81, Appinfo descendants are not XML Schema elements	rule9-77
Rule 9-82, Schema has data definition	rule7-37
Rule 9-83, Schema document defines target namespace	rule8-1
Rule 9-84, Target namespace is absolute URI	rule8-1
Rule 9-85, Schema has version	rule8-7
Rule 9-86, No disallowed substitutions	rule9-82
Rule 9-87, No disallowed derivations	rule9-83
Rule 9-88, No use of xs:redefine	rule9-5
Rule 9-89, No use of xs:include	rule9-5
Rule 9-90, xs:import must have namespace	rule9-23

NIEM 5 Rule	NIEM 6 Rules
Rule 9-91, XML Schema document set must be complete	rule10-9
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	rule9-31
Rule 10-2, Object type with complex content is derived from structures:ObjectType	rule9-32
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	rule9-52
Rule 10-8, External adapter type has indicator	rule9-53
Rule 10-9, Structure of external adapter type definition follows pattern	rule9-54
Rule 10-10, Element use from external adapter type defined by external schema documents	rule9-55
Rule 10-11, External adapter type not a base type	rule9-56
Rule 10-12, External adapter type not a base type	rule9-56
Rule 10-13, External attribute use only in external adapter type	rule9-86
Rule 10-14, External attribute use has data definition	rule9-57
Rule 10-15, External attribute use not an ID	rule9-58
Rule 10-16, External element use has data definition	rule9-59
Rule 10-17, Name of code type ends in CodeType	rule7-6, rule7-9

NIEM 5 Rule	NIEM 6 Rules
Rule 10-18, Code type corresponds to a code list	rule7-6, rule7-9
Rule 10-19, Element of code type has code representation term	rule7-12
Rule 10-20, Proxy type has designated structure	rule9-60, rule9-61
Rule 10-21, Association type derived from structures:AssociationType	rule7-5, rule9-34
Rule 10-22, Association element type is an association type	rule7-11
Rule 10-23, Augmentable type has augmentation point element	*no matching NIEM6 rule*
Rule 10-24, Augmentable type has at most one augmentation point element	*no matching NIEM6 rule*
Rule 10-25, Augmentation point element corresponds to its base type	rule9-68
Rule 10-26, An augmentation point element has no type	rule9-69
Rule 10-27, An augmentation point element has no substitution group	rule9-70
Rule 10-28, Augmentation point element is only referenced by its base type	rule9-71
Rule 10-29, Augmentation point element use is optional	rule9-72
Rule 10-30, Augmentation point element use is unbounded	rule9-72
Rule 10-31, Augmentation point element use must be last element in its base type	rule9-73
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	rule9-62
Rule 10-34, Schema component with name ending in AugmentationType is an augmentation type	rule9-62
Rule 10-35, Type derived from structures:AugmentationType is an augmentation type	rule9-35
Rule 10-36, Augmentation element type is an augmentation type	rule9-63
Rule 10-37, Augmentation elements are not used directly	rule9-40
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	rule7-10

NIEM 5 Rule	NIEM 6 Rules
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	rule7-15
Rule 10-45, Schema component name has xml:lang	rule8-8
Rule 10-46, Schema component names have only specific characters	rule7-16
Rule 10-47, Punctuation in component name is a separator	rule7-20
Rule 10-48, Names use camel case	rule7-17
Rule 10-49, Attribute name begins with lower case letter	rule7-18
Rule 10-50, Name of schema component other than attribute and proxy type begins with upper case letter	rule7-19
Rule 10-51, Names use common abbreviations	rule7-34
Rule 10-52, Local term declaration is local to its schema document	rule7-35
Rule 10-53, Local terminology interpretation	*no matching NIEM6 rule*
Rule 10-54, Singular form is preferred in name	rule7-21
Rule 10-55, Present tense is preferred in name	rule7-22
Rule 10-56, Name does not have nonessential words	rule7-23
Rule 10-57, Element or attribute name follows pattern	rule7-24
Rule 10-58, Object-class term identifies concrete category	rule7-25
Rule 10-59, Property term describes characteristic or subpart	rule7-26
Rule 10-60, Name may have multiple qualifier terms	rule7-27
Rule 10-61, Name has minimum necessary number of qualifier terms	rule7-28
Rule 10-62, Order of qualifiers is not significant	rule7-29
Rule 10-63, Redundant term in name is omitted	rule7-30
Rule 10-64, Element with simple content has representation term	rule7-31
Rule 10-65, Element with complex content has representation term when appropriate	rule7-32
Rule 10-66, Element with complex content has representation term only when appropriate	rule7-33
Rule 10-67, Machine-readable annotations are valid	*no matching NIEM6 rule*
Rule 10-68, Component marked as deprecated is deprecated component	rule9-78
Rule 10-69, Deprecated annotates schema component	rule9-74

NIEM 5 Rule	NIEM 6 Rules
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*
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	rule9-79
Rule 10-77, appinfo:LocalTerm has literal or definition	rule7-36
Rule 10-78, Use structures consistent with specification	rule10-10
Rule 11-1, Name of type ends in Type	rule7-2
Rule 11-2, Only types have name ending in Type or SimpleType	rule7-2, rule9-25
Rule 11-3, Base type definition defined by conformant schema	rule9-37
Rule 11-4, Name of simple type ends in SimpleType	rule9-26
Rule 11-5, Use lists only when data is uniform	*no matching NIEM6 rule*
Rule 11-6, List item type defined by conformant schemas	rule9-41
Rule 11-7, Union member types defined by conformant schemas	rule9-42
Rule 11-8, Name of a code simple type ends in CodeSimpleType	rule7-8
Rule 11-9, Code simple type corresponds to a code list	rule7-8
Rule 11-10, Attribute of code simple type has code representation term	rule7-12
Rule 11-11, Complex type with simple content has structures:SimpleObjectAttributeGroup	rule9-36
Rule 11-12, Element type does not have a simple type name	rule9-46
Rule 11-13, Element type is from conformant namespace	rule9-47
Rule 11-14, Name of element that ends in Abstract is abstract	rule7-10
Rule 11-15, Name of element declaration with simple content has representation term	rule7-31
Rule 11-16, Name of element declaration with simple content has representation term	rule7-31

NIEM 5 Rule	NIEM 6 Rules
Rule 11-17, Element substitution group defined by conformant schema	rule9-48
Rule 11-18, Attribute type defined by conformant schema	rule9-47
Rule 11-19, Attribute name uses representation term	rule7-31
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	rule9-38
Rule 11-22, Referenced attribute defined by conformant schemas	rule9-38
Rule 11-23, Schema uses only known attribute groups	rule9-39
Rule 11-24, Data definition does not introduce ambiguity	rule7-42
Rule 11-25, Object class has only one meaning	rule7-43
Rule 11-26, Data definition of a part does not redefine the whole	rule7-44
Rule 11-27, Do not leak representation into data definition	rule7-45
Rule 11-28, Data definition follows 11179-4 requirements	rule7-46
Rule 11-29, Data definition follows 11179-4 recommendations	rule7-47
Rule 11-30, xs:documentation has xml:lang	rule8-8
Rule 11-31, Standard opening phrase for augmentation point element data definition	rule9-65
Rule 11-32, Standard opening phrase for augmentation element data definition	rule9-66
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	rule7-49
Rule 11-35, Standard opening phrase for abstract element data definition	rule7-48
Rule 11-36, Standard opening phrase for date element or attribute data definition	rule7-50
Rule 11-37, Standard opening phrase for quantity element or attribute data definition	rule7-51
Rule 11-38, Standard opening phrase for picture element or attribute data definition	rule7-52
Rule 11-39, Standard opening phrase for indicator element or attribute data definition	rule7-53
Rule 11-40, Standard opening phrase for identification element or attribute data definition	rule7-54
Rule 11-41, Standard opening phrase for name element or attribute data definition	rule7-55
Rule 11-42, Standard opening phrase for element or attribute data definition	rule7-56
Rule 11-43, Standard opening phrase for association type data definition	rule7-57
Rule 11-44, Standard opening phrase for augmentation type data definition	rule9-67

NIEM 5 Rule	NIEM 6 Rules
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	rule7-58
Rule 11-47, Standard opening phrase for simple type data definition	rule7-58
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	rule8-12
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	rule10-5
Rule 11-55, Consistently marked namespace imports	rule10-6
Rule 12-1, Instance must be schema-valid	rule12-2
Rule 12-2, Empty content has no meaning	*no matching NIEM6 rule*
Rule 12-3, Element has only one resource identifying attribute	rule12-7
Rule 12-4, Attribute structures:ref must reference structures:id	rule12-8
Rule 12-5, Linked elements have same validation root	rule12-9
Rule 12-6, Attribute structures:ref references element of correct type	rule12-10
Rule 12-7, structures:uri denotes resource identifier	*no matching NIEM6 rule*
Rule 12-8, structures:id and structures:ref denote resource identifier	*no matching NIEM6 rule*
Rule 12-9, Nested elements and references have the same meaning.	*no matching NIEM6 rule*
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*

NIEM 5 Rule	NIEM 6 Rules
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. Table of examples

Appendix F. Table of figures

Appendix G. Table of tables

Appendix H. Acknowledgments

H.1 Participants

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

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

Appendix I. Notices

(This required section should not be altered, except to modify the license information in the second paragraph if needed.)

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the “OASIS IPR Policy”). The full Policy may be found at the OASIS website.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the NIEMOpen IPR Statement page.