National Information Exchange Model Naming and Design Rules

Version 5.0

NIEM Technical Architecture Committee (NTAC)

December 18, 2020

# Abstract

This document specifies the data model, XML Schema components, and XML data for use with the National Information Exchange Model (NIEM) version 5.0.

# Authors

Webb Roberts, Georgia Tech Research Institute ([<webb.roberts@gtri.gatech.edu>](mailto:webb.roberts@gtri.gatech.edu)), Lead Author

# Status

This document specifies the data model, XML Schema components, and XML data for use with the National Information Exchange Model (NIEM) version 5.0. It presents a technical architecture that has evolved through the collaborative work of the NIEM Business Architecture Committee (NBAC), the NIEM Technical Architecture Committee (NTAC), and their predecessors.

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

Updates, revisions, and errata for this specification are posted to <https://github.com/NIEM/NIEM-NDR>. Please submit comments on this specification as issues at <https://github.com/NIEM/NIEM-NDR/issues>.

# Contents

* [1. Introduction](#section_1)
* [1.1. Scope](#section_1.1)
* [1.2. Audience](#section_1.2)
* [2. Document conventions and normative content](#section_2)
* [2.1. Document references](#section_2.1)
* [2.2. Clark notation and qualified names](#section_2.2)
* [2.3. Use of namespaces and namespace prefixes](#section_2.3)
* [2.4. Normative and informative content](#section_2.4)
* [2.4.1. Rules](#section_2.4.1)
* [2.4.2. Use of normative Schematron](#section_2.4.2)
* [2.4.3. Normative XPath functions](#section_2.4.3)
* [2.4.4. Normative Schematron namespace declarations](#section_2.4.4)
* [2.5. Additional formatting](#section_2.5)
* [3. Terminology](#section_3)
* [3.1. IETF Best Current Practice 14 terminology](#section_3.1)
* [3.2. XML terminology](#section_3.2)
* [3.3. XML Information Set terminology](#section_3.3)
* [3.4. XML Schema terminology](#section_3.4)
* [3.4.1. Schema components](#section_3.4.1)
* [3.4.2. Schema information set contributions](#section_3.4.2)
* [3.5. XML Namespaces terminology](#section_3.5)
* [3.6. Conformance Targets Attribute Specification terminology](#section_3.6)
* [4. Conformance targets](#section_4)
* [4.1. Conformance targets defined](#section_4.1)
* [4.1.1. Reference schema document](#section_4.1.1)
* [4.1.2. Extension schema document](#section_4.1.2)
* [4.1.3. Schema document set](#section_4.1.3)
* [4.1.4. Instance documents and elements](#section_4.1.4)
* [4.2. Applicability of rules to conformance targets](#section_4.2)
* [4.3. Conformance target identifiers](#section_4.3)
* [5. The NIEM conceptual model](#section_5)
* [5.1. Purpose of the NIEM conceptual model](#section_5.1)
* [5.2. The RDF model](#section_5.2)
* [5.3. NIEM in terms of RDF](#section_5.3)
* [5.4. Unique identification of data objects](#section_5.4)
* [5.5. NIEM data is explicit, not implicit](#section_5.5)
* [5.6. Mapping of NIEM concepts to RDF concepts](#section_5.6)
* [5.6.1. Resource IRIs for XML Schema components and information items](#section_5.6.1)
* [5.6.2. RDF Literals](#section_5.6.2)
* [5.6.3. Instance document mapped to RDF](#section_5.6.3)
* [5.6.3.1. Instance document](#section_5.6.3.1)
* [5.6.3.2. Element as a simple property without relationship metadata](#section_5.6.3.2)
* [5.6.3.3. Element as a simple property with relationship metadata](#section_5.6.3.3)
* [5.6.3.4. Element simple value without relationship metadata](#section_5.6.3.4)
* [5.6.3.5. Element simple value with relationship metadata](#section_5.6.3.5)
* [5.6.3.6. Attribute as a simple property without relationship metadata](#section_5.6.3.6)
* [5.6.3.7. Attribute as a simple property, with relationship metadata](#section_5.6.3.7)
* [5.6.3.8. Elements and attributes via an augmentation type](#section_5.6.3.8)
* [5.6.3.9. Properties applied via structures:metadata](#section_5.6.3.9)
* [5.6.4. Type information for instance documents](#section_5.6.4)
* [5.6.5. NIEM schema component definitions to RDF](#section_5.6.5)
* [5.6.5.1. NIEM complex type definitions to RDF](#section_5.6.5.1)
* [5.6.5.2. NIEM element declaration mappings to RDF](#section_5.6.5.2)
* [5.6.5.3. NIEM attribute declarations to RDF](#section_5.6.5.3)
* [6. Guiding principles](#section_6)
* [6.1. Specification guidelines](#section_6.1)
* [6.1.1. Keep specification to a minimum](#section_6.1.1)
* [6.1.2. Focus on rules for schemas](#section_6.1.2)
* [6.1.3. Use specific, concise rules](#section_6.1.3)
* [6.2. XML Schema design guidelines](#section_6.2)
* [6.2.1. Purpose of XML Schemas](#section_6.2.1)
* [6.2.2. Prohibit XML parsing from constructing values](#section_6.2.2)
* [6.2.3. Use XML validating parsers for content validation](#section_6.2.3)
* [6.2.4. Validate for conformance to schema](#section_6.2.4)
* [6.2.5. Allow multiple schemas for XML constraints](#section_6.2.5)
* [6.2.6. Define one reference schema per namespace](#section_6.2.6)
* [6.2.7. Disallow mixed content](#section_6.2.7)
* [6.2.8. Specify types for all constructs](#section_6.2.8)
* [6.2.9. Avoid wildcards in reference schemas](#section_6.2.9)
* [6.2.10. Schema locations provided in schema documents are hints](#section_6.2.10)
* [6.2.11. Use open standards](#section_6.2.11)
* [6.3. Modeling design guidelines](#section_6.3)
* [6.3.1. Namespaces enhance reuse](#section_6.3.1)
* [6.3.2. Design NIEM for extensibility](#section_6.3.2)
* [6.4. Implementation guidelines](#section_6.4)
* [6.4.1. Avoid displaying raw XML data](#section_6.4.1)
* [6.4.2. Leave implementation decisions to implementers](#section_6.4.2)
* [6.5. Modeling guidelines](#section_6.5)
* [6.5.1. Documentation](#section_6.5.1)
* [6.5.2. Consistent naming](#section_6.5.2)
* [6.5.3. Reflect the real world](#section_6.5.3)
* [6.5.4. Be consistent](#section_6.5.4)
* [6.5.5. Reserve inheritance for specialization](#section_6.5.5)
* [6.5.6. Do not duplicate definitions](#section_6.5.6)
* [6.5.7. Keep it simple](#section_6.5.7)
* [6.5.8. Be aware of scope](#section_6.5.8)
* [6.5.9. Be mindful of namespace cohesion](#section_6.5.9)
* [7. Conformance to standards](#section_7)
* [7.1. Conformance to XML](#section_7.1)
* [7.2. Conformance to XML Namespaces](#section_7.2)
* [7.3. Conformance to XML Schema](#section_7.3)
* [7.4. ISO 11179 Part 4](#section_7.4)
* [7.5. ISO 11179 Part 5](#section_7.5)
* [7.6. IC-ISM and IC-NTK](#section_7.6)
* [8. Strategy for a NIEM profile of XML Schema](#section_8)
* [8.1. Wildcards](#section_8.1)
* [8.2. Components are globally reusable](#section_8.2)
* [8.3. Avoid recursive model groups](#section_8.3)
* [8.4. Ensure XML parsing does not construct values](#section_8.4)
* [8.5. Use namespaces rigorously](#section_8.5)
* [8.6. Documentation is for people; appinfo is for machines](#section_8.6)
* [9. Rules for a NIEM profile of XML Schema](#section_9)
* [9.1. Type definition components](#section_9.1)
* [9.1.1. Type definition hierarchy](#section_9.1.1)
* [9.1.1.1. Types prohibited as base types](#section_9.1.1.1)
* [9.1.2. Simple type definition](#section_9.1.2)
* [9.1.2.1. Simple types prohibited as list item types](#section_9.1.2.1)
* [9.1.2.2. Simple types prohibited as union member types](#section_9.1.2.2)
* [9.1.3. Complex type definition](#section_9.1.3)
* [9.1.3.1. No mixed content](#section_9.1.3.1)
* [9.1.3.2. Complex content](#section_9.1.3.2)
* [9.1.3.2.1. Base type of complex type with complex content has complex content](#section_9.1.3.2.1)
* [9.1.3.3. Simple content](#section_9.1.3.3)
* [9.2. Declaration components](#section_9.2)
* [9.2.1. Element declaration](#section_9.2.1)
* [9.2.1.1. No element value constraints](#section_9.2.1.1)
* [9.2.2. Element substitution group](#section_9.2.2)
* [9.2.3. Attribute declaration](#section_9.2.3)
* [9.2.3.1. Prohibited attribute types](#section_9.2.3.1)
* [9.2.3.2. No attribute value constraints](#section_9.2.3.2)
* [9.2.4. Notation declaration](#section_9.2.4)
* [9.3. Model group components](#section_9.3)
* [9.3.1. Model group](#section_9.3.1)
* [9.3.1.1. Sequence](#section_9.3.1.1)
* [9.3.1.2. Choice](#section_9.3.1.2)
* [9.3.2. Particle](#section_9.3.2)
* [9.3.2.1. Sequence cardinality](#section_9.3.2.1)
* [9.3.2.2. Choice cardinality](#section_9.3.2.2)
* [9.3.3. Attribute use](#section_9.3.3)
* [9.3.4. Wildcard](#section_9.3.4)
* [9.4. Identity-constraint definition components](#section_9.4)
* [9.5. Group definition components](#section_9.5)
* [9.5.1. Model group definition](#section_9.5.1)
* [9.5.2. Attribute group definition](#section_9.5.2)
* [9.6. Annotation components](#section_9.6)
* [9.6.1. Application information annotation](#section_9.6.1)
* [9.7. Schema as a whole](#section_9.7)
* [9.8. Schema assembly](#section_9.8)
* [9.8.1. Namespaces for referenced components are imported](#section_9.8.1)
* [10. Rules for NIEM modeling, by NIEM concept](#section_10)
* [10.1. Categories of NIEM type definitions](#section_10.1)
* [10.2. Object](#section_10.2)
* [10.2.1. General object types](#section_10.2.1)
* [10.2.1.1. Object types with complex content](#section_10.2.1.1)
* [10.2.2. Role types and roles](#section_10.2.2)
* [10.2.3. External adapter types and external components](#section_10.2.3)
* [10.2.3.1. Import of external namespace](#section_10.2.3.1)
* [10.2.3.2. External adapter types](#section_10.2.3.2)
* [10.2.3.3. External attribute use](#section_10.2.3.3)
* [10.2.3.4. External element use](#section_10.2.3.4)
* [10.2.4. Code types](#section_10.2.4)
* [10.2.5. Proxy types](#section_10.2.5)
* [10.3. Associations](#section_10.3)
* [10.3.1. Association types](#section_10.3.1)
* [10.3.2. Association element declarations](#section_10.3.2)
* [10.4. Augmentations](#section_10.4)
* [10.4.1. Augmentable types](#section_10.4.1)
* [10.4.2. Augmentation point element declarations](#section_10.4.2)
* [10.4.3. Augmentation point element use](#section_10.4.3)
* [10.4.4. Augmentation types](#section_10.4.4)
* [10.4.5. Augmentation element declarations](#section_10.4.5)
* [10.5. Metadata](#section_10.5)
* [10.5.1. Metadata types](#section_10.5.1)
* [10.5.2. Metadata element declarations](#section_10.5.2)
* [10.6. Container elements](#section_10.6)
* [10.7. The “Representation” pattern](#section_10.7)
* [10.8. Naming rules](#section_10.8)
* [10.8.1. Character case](#section_10.8.1)
* [10.8.2. Use of acronyms and abbreviations](#section_10.8.2)
* [10.8.2.1. Use of Acronyms, Initialisms, Abbreviations, and Jargon](#section_10.8.2.1)
* [10.8.3. Word forms](#section_10.8.3)
* [10.8.4. Object-class term](#section_10.8.4)
* [10.8.5. Property term](#section_10.8.5)
* [10.8.6. Qualifier terms](#section_10.8.6)
* [10.8.7. Representation terms](#section_10.8.7)
* [10.9. Machine-readable annotations](#section_10.9)
* [10.9.1. The NIEM appinfo namespace](#section_10.9.1)
* [10.9.1.1. Deprecation](#section_10.9.1.1)
* [10.9.1.2. External adapters](#section_10.9.1.2)
* [10.9.1.3. appinfo:appliesToTypes annotation](#section_10.9.1.3)
* [10.9.1.4. appinfo:appliesToElements annotation](#section_10.9.1.4)
* [10.9.2. Local terminology](#section_10.9.2)
* [10.10. NIEM structural facilities](#section_10.10)
* [11. Rules for NIEM modeling, by XML Schema component](#section_11)
* [11.1. Type definition components](#section_11.1)
* [11.1.1. Type definition hierarchy](#section_11.1.1)
* [11.1.2. Simple type definition](#section_11.1.2)
* [11.1.2.1. Derivation by list](#section_11.1.2.1)
* [11.1.2.2. Derivation by union](#section_11.1.2.2)
* [11.1.2.3. Code simple types](#section_11.1.2.3)
* [11.1.3. Complex type definition](#section_11.1.3)
* [11.2. Declaration components](#section_11.2)
* [11.2.1. Element declaration](#section_11.2.1)
* [11.2.1.1. Object element declarations](#section_11.2.1.1)
* [11.2.2. Element substitution group](#section_11.2.2)
* [11.2.3. Attribute declaration](#section_11.2.3)
* [11.2.4. Notation declaration](#section_11.2.4)
* [11.3. Model group components](#section_11.3)
* [11.3.1. Model group](#section_11.3.1)
* [11.3.2. Particle](#section_11.3.2)
* [11.3.2.1. Element use](#section_11.3.2.1)
* [11.3.3. Attribute use](#section_11.3.3)
* [11.3.3.1. Attribute group use](#section_11.3.3.1)
* [11.3.4. Wildcard](#section_11.3.4)
* [11.4. Identity-constraint definition components](#section_11.4)
* [11.5. Group definition components](#section_11.5)
* [11.5.1. Model group definition](#section_11.5.1)
* [11.5.2. Attribute group definition](#section_11.5.2)
* [11.6. Annotation components](#section_11.6)
* [11.6.1. Human-readable documentation](#section_11.6.1)
* [11.6.1.1. Data definition opening phrases](#section_11.6.1.1)
* [11.6.1.1.1. Element and attribute opening phrases](#section_11.6.1.1.1)
* [11.6.1.1.2. Complex type opening phrases](#section_11.6.1.1.2)
* [11.7. Schema as a whole](#section_11.7)
* [11.7.1. xs:schema document element restrictions](#section_11.7.1)
* [11.8. Schema assembly](#section_11.8)
* [11.8.1. Supporting namespaces are imported as conformant](#section_11.8.1)
* [12. XML instance document rules](#section_12)
* [12.1. The meaning of NIEM data](#section_12.1)
* [12.2. Identifiers and references](#section_12.2)
* [12.2.1. Local identifiers and references](#section_12.2.1)
* [12.2.2. Uniform resource identifiers in NIEM data](#section_12.2.2)
* [12.2.3. Differentiating reference-to-identifier links and use of URIs](#section_12.2.3)
* [12.2.4. Reference and content elements have same meaning](#section_12.2.4)
* [12.3. Property order and sequence identifiers](#section_12.3)
* [12.4. Instance metadata](#section_12.4)
* [Appendix A. References](#appendix_A)
* [Appendix B. Structures namespace](#appendix_B)
* [Appendix C. Appinfo namespace](#appendix_C)
* [Appendix D. Index of definitions](#appendix_D)
* [Appendix E. Index of rules](#appendix_E)
* [Appendix F. General index](#appendix_F)

# Table of Figures

* [Figure 2-1: Normative Schematron namespace declarations](#figure_2-1)
* [Figure 2-2: Example of an XML fragment](#figure_2-2)
* [Figure 6-1: Example of the use of a namespace](#figure_6-1)
* [Figure 7-1: Example of data definition of element nc:Activity](#figure_7-1)
* [Figure 9-1: Example of complex type with simple content that claims to have complex content](#figure_9-1)
* [Figure 10-1: An element declaration that constitutes a role without the use of a role type](#figure_10-1)
* [Figure 10-2: Element j:CrashDriver, modeling the role of a driver in a crash](#figure_10-2)
* [Figure 10-3: Role type j:CrashDriverType, modeling a driver involved in a crash](#figure_10-3)
* [Figure 10-4: Declaration of RoleOf element nc:RoleOfPerson](#figure_10-4)
* [Figure 10-5: An XML instance of a role type](#figure_10-5)
* [Figure 10-6: Use of external components to create a NIEM-conformant type](#figure_10-6)
* [Figure 10-7: An association in an instance](#figure_10-7)
* [Figure 10-8: A definition of an association type](#figure_10-8)
* [Figure 10-9: Definition of augmentable type nc:PersonType](#figure_10-9)
* [Figure 10-10: Declaration of augmentation point element nc:PersonAugmentationPoint](#figure_10-10)
* [Figure 10-11: Declaration of augmentation type and element](#figure_10-11)
* [Figure 10-12: An augmentation that is not an augmentation type](#figure_10-12)
* [Figure 10-13: A type used for an augmentation](#figure_10-13)
* [Figure 10-14: Sample use of appinfo:appliesToTypes](#figure_10-14)
* [Figure 10-15: A type definition that uses the “Representation” pattern](#figure_10-15)
* [Figure 10-16: Element declarations that implement representations](#figure_10-16)
* [Figure 11-1: Example of complex type with simple content derived from a simple type](#figure_11-1)
* [Figure 12-1: Example of content elements](#figure_12-1)
* [Figure 12-2: Diagram showing the meaning of XML data](#figure_12-2)
* [Figure 12-3: Example of structures:id and structures:ref](#figure_12-3)
* [Figure 12-4: Example of structures:uri holding an absolute URI](#figure_12-4)
* [Figure 12-5: Example of structures:uri holding a relative URI, with an xml:base](#figure_12-5)
* [Figure 12-6: Example of structures:uri holding a fragment](#figure_12-6)
* [Figure 12-7: Example of structures:uri, structures:id, and structures:ref identifying the same object.](#figure_12-7)
* [Figure 12-8: Example with no reference](#figure_12-8)
* [Figure 12-9: Example with a backward reference](#figure_12-9)
* [Figure 12-10: Example with a forward reference](#figure_12-10)
* [Figure 12-11: An instance of a name type](#figure_12-11)
* [Figure 12-12: An instance of a name type that uses structures:sequenceID](#figure_12-12)
* [Figure 12-13: A simple example of object metadata](#figure_12-13)
* [Figure 12-14: A simple example of relationship metadata](#figure_12-14)

# Table of Tables

* [Table 4-1: Codes representing conformance targets](#table_4-1)
* [Table 10-1: Abbreviations used in schema component names](#table_10-1)
* [Table 10-2: Property representation terms](#table_10-2)
* [Table 12-1: Meaning of NIEM XML](#table_12-1)

# 1. Introduction

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

NIEM is a product of a combined government and industry effort to improve information interoperability and exchange within the United States at federal, state, tribal, and local levels of government. NIEM may have started in the U.S., but its reach does not stop there. International governments and private sector organizations can benefit from the value of NIEM, as well. Communities in Europe, North America, and Australia already use NIEM for their information exchange efforts. NIEM 5.0 represents an initial step toward evolving NIEM to support a more-global exchange environment

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

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

This document specifies principles and enforceable rules for NIEM data components and schemas. Schemas and components that obey the rules set forth here are considered to be conformant to specific conformance targets. These targets are defined in order that they may be leveraged for comprehensive definitions of NIEM conformance. Such definitions 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. Scope

This document was developed to specify NIEM 5.0. Later releases of NIEM may be specified by later versions of this document. The document covers the following issues in depth:

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

This document does NOT address the following:

* A *formal* definition of the NIEM data model. Such a definition would focus on the Resource Description Framework (RDF) and concepts not strictly required for interoperability. This document instead focuses on definition of schemas that work with the data model, to ensure translatability and interoperability.
* A detailed discussion of NIEM architecture and schema versioning.
* Aggregate artifacts that define information exchanges and models, such as message specifications, information exchange package descriptions (IEPDs) and other forms of model package descriptions (MPDs), information exchange specifications (IESs), and enterprise information exchange models (EIEMs).
* Normative guidance for the location of [·schema documents·](#definition_schema_document) or for schema assembly.

This document is intended as a technical specification. It is not intended to be a tutorial or a user guide.

## 1.2. Audience

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

# 2. Document conventions and normative content

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

## 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]**](#RFC3986). All reference documents are recorded in [Appendix A, *References*, below](#appendix_A).

## 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]**](#XMLNamespaces) [Section 4, *Qualified Names*](http://www.w3.org/TR/2009/REC-xml-names-20091208/#NT-QName). 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](#section_2.3).

This document sometimes uses Clark notation to represent qualified names in normative text. Clark notation is described by [**[ClarkNS]**](#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]**](#XMLSchema-1) and [**[XML Schema Datatypes]**](#XMLSchema-2), “http://www.w3.org/2001/XMLSchema”.
* xsi: The XML Schema instance namespace, defined by [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.6, *Schema-Related Markup in Documents Being Validated*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#Instance_Document_Constructions), for use in XML documents, “http://www.w3.org/2001/XMLSchema-instance”.
* sch: The Schematron namespace, as defined by [**[Schematron]**](#Schematron), “http://purl.oclc.org/dsdl/schematron”.
* nf: The namespace defined by this specification for XPath functions, “http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#NDRFunctions”.
* ct: The namespace defined by [**[CTAS]**](#CTAS) for the conformanceTargets attribute, “http://release.niem.gov/niem/conformanceTargets/3.0/”.
* appinfo: The namespace for the [·appinfo namespace·](#definition_appinfo_namespace), “http://release.niem.gov/niem/appinfo/5.0/”.
* structures: The namespace for the [·structures namespace·](#definition_structures_namespace), “http://release.niem.gov/niem/structures/5.0/”.

In addition to the prefixes above, the prefix example is used for XML examples, denoting a user-defined namespace, such as might be defined by an information exchange specification.

## 2.4. Normative and informative content

This document includes a variety of content. Some content of this document is [·normative·](#definition_normative), while other content is [·informative·](#definition_informative). In general, the informative material appears as supporting text, description, and rationales for the normative material.

[Definition: normative]

The term “normative” is as defined by [**[ConfReq]**](#ConfReq) Section 7.2, *Conformance by key words*, which states:

NORMATIVE — statements provided for the prescriptive parts of the specification, providing that which is necessary in order to be able to claim conformance to the specification.

[Definition: informative]

The term “informative” is as defined by [**[ConfReq]**](#ConfReq) Section 7.2, *Conformance by key words*, which states:

INFORMATIVE (NON-NORMATIVE) — statements provided for informational purposes, intended to assist the understanding or use of the specification and shall not contain provisions that are required for conformance.

Conventions used within this document include:

[Definition: <term>]

A formal definition of a term associated with NIEM.

Definitions are [·normative·](#definition_normative). Uses of these terms are given special formatting, using raised dots to identify the term, for example this use of the term [·conformance target·](#definition_conformance_target).

[Principle ]

A guiding principle for NIEM.

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

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

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

### 2.4.1. Rules

A rule states a specific requirement on an artifact or on the interpretation of an artifact. The classes of artifacts are identified by [·conformance targets·](#definition_conformance_target) that are enumerated by this document in [Section 4.1, *Conformance targets defined*, below](#section_4.1). The rules are normative. Human-readable text in rules uses [**[BCP 14]**](#BCP14) terminology as described in [Section 3.1, *IETF Best Current Practice 14 terminology*, below,](#section_3.1) for normative requirements and recommendations.

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

An enforceable rule for NIEM.

Each rule has a *classification*, which is either “Constraint” or “Interpretation”. If the classification is “Constraint”, then the rule is a [·constraint rule·](#definition_constraint_rule). If the classification is “Interpretation”, then the rule is an [·interpretation rule·](#definition_interpretation_rule).

[Definition: constraint rule]

A **constraint rule** is a rule that sets a requirement on an artifact with respect to its conformance to a [·conformance target·](#definition_conformance_target).

[Definition: interpretation rule]

An **interpretation rule** is a rule that sets the methodology, pattern, or procedure for understanding some aspect of an instance of a conformance target.

Each rule identifies its *applicability*. This identifies the conformance targets to which the rule applies. Each entry in the list is a code from [Table 4-1, *Codes representing conformance targets*, below](#table_4-1). If a code appears in the applicability list for a rule, then the rule applies to the corresponding conformance target. The conformance targets are defined in [Section 4, *Conformance targets*, below](#section_4). For example, a rule with applicability “(REF, EXT)” would be applicable to a [·reference schema document·](#definition_reference_schema_document), as well as to an [·extension schema document·](#definition_extension_schema_document).

Rules are stated with the help of XML Infoset terminology (e.g., elements and attributes), as described by [Section 3.3, *XML Information Set terminology*, below](#section_3.3), and XML Schema terminology (e.g., schema components), as described by [Section 3.4, *XML Schema terminology*, below](#section_3.4). The choice of terminology is driven by which terms best express a concept. Certain concepts are more clearly expressed using XML Infoset terms items, others using XML Schema terms; still others are best expressed using a combination of terminology drawn from each standard.

Rules are numbered according to the section in which they appear and the order in which they appear within that section. For example, the first rule in Section 7 is Rule 7-1.

### 2.4.2. Use of normative Schematron

This document defines many normative rules using Schematron rule-based validation syntax, as defined by [**[Schematron]**](#Schematron). For example, see [Rule 9-29, *Complex type content is explicitly simple or complex*, below](#rule_9-29). Effort has been made to make the rules precise and unambiguous. Very detailed text descriptions of rules can introduce ambiguity, and they are not directly executable by users. Providing NDR rules that are expressed as Schematron rules ensures that the rules are precise, and that they are directly executable through commercially-available and free tools.

Many rules herein do not have executable Schematron supporting them. Some are not fit for automatic validation, and others may be difficult or cumbersome to express in Schematron. In neither case are such rules any less normative. A rule that has no Schematron is just as normative as a rule that does have Schematron. The level of requirements and recommendations within a rule is expressed using terminology from [**[BCP 14]**](#BCP14) as described in [Section 3.1, *IETF Best Current Practice 14 terminology*, below](#section_3.1).

The Schematron rules are written using XPath2 as defined by [**[XPath 2]**](#XPath2). These executable rules are normative.

An execution of a Schematron pattern that issues a failed assertion (represented via sch:assert) represents a validation error, and signifies that the assessed artifact violates a requirement (i.e., a “MUST”) of a conformance rule. An example of a constraint rule that uses Schematron is [Rule 9-10, *Simple type definition is top-level*, below](#rule_9-10).

An execution of a Schematron pattern that issues a report (represented via sch:report) indicates a warning. This may be an indication that an automated rule has found that the assessed artifact violates a recommendation of the specification (i.e., a “SHOULD”, rather than a “MUST”), and that attention should be paid to ensure that the artifact maintains the spirit of the specification. For example, see [Rule 10-42, *Name of element that ends in “Representation” is abstract*, below](#rule_10-42).

In addition to the exclusive use of sch:report for warnings, this document annotates sch:report elements with attribute role="warning", which is interpreted by some Schematron platforms to indicate that a failure indicates a warning, rather than an error.

In either case, a diagnostic report generated by testing an XML document against the Schematron rules may identify specific locations (e.g., line numbers) within the document that need further attention.

### 2.4.3. Normative XPath functions

The Schematron within this document is supported by functions, to make the rules more comprehensible, and to abstract away process-specific operations. Each function has a normative XPath interface and a normative text definition. Any implementation provided for these functions should be considered informative, not normative, but may be useful for certain implementations of the rules.

The following XPath functions are defined normatively when used within Schematron by this specification:

* nf:get-document-element($context as element()) as element()

Yields the document element of the XML document in which $context occurs.

This function provides the ability for a validator to consolidate multiple XML Schema documents and XML instance documents into a single XML document, which may simplify validation, and allow for preprocessing of xs:include elements.

* nf:get-target-namespace($element as element()) as xs:anyURI?

Yields the target namespace of the XML Schema document in which *$element* appears. If it is a [·schema document·](#definition_schema_document) with no target namespace defined, then it yields the zero-length xs:anyURI value (xs:anyURI('')). If the [·XML document·](#definition_XML_document) in which *$element* appears is not a [·schema document·](#definition_schema_document), then the function yields the empty sequence (()).

* nf:resolve-namespace($context as element(), $namespace-uri as xs:anyURI) as element(xs:schema)?

Yields the document element of the first available [·schema document·](#definition_schema_document) that has the target namespace *$namespace-uri*. If there is no such [·schema document·](#definition_schema_document) available, it yields the empty sequence (()).

* nf:resolve-type($context as element(), $qname as xs:QName) as element()?

Yields the first occurrence of an element xs:simpleType or xs:complexType that defines a type with a {target namespace} and {name} matching *$qname*, that is a [child] of the element yielded by nf:resolve-namespace(), above. If there is no such occurrence, it yields the empty sequence (()).

* nf:resolve-element($context as element(), $qname as xs:QName) as element(xs:element)?

Yields the first occurrence of an element xs:element that declares an element with a {target namespace} and {name} matching *$qname*, that is a [child] of the element yielded by nf:resolve-namespace(), above. If there is no occurrence available, it yields the empty sequence. (())

* nf:has-effective-conformance-target-identifier($context as element(), $match as xs:anyURI) as xs:boolean

Yields true if and only if an [·effective conformance target identifier·](#X6d0b4369562c0703761463ba00cd0e2b474de22) of the XML document containing *$context* is *$match*.

### 2.4.4. Normative Schematron namespace declarations

The following Schematron namespace declarations are normative for the Schematron rules and supporting Schematron code within this specification:

Figure 2-1: Normative Schematron namespace declarations

<sch:ns prefix="xs" uri="http://www.w3.org/2001/XMLSchema"/>  
<sch:ns prefix="xsl" uri="http://www.w3.org/1999/XSL/Transform"/>  
<sch:ns prefix="nf" uri="http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#NDRFunctions"/>  
<sch:ns prefix="ct" uri="http://release.niem.gov/niem/conformanceTargets/3.0/"/>  
<sch:ns prefix="xsi" uri="http://www.w3.org/2001/XMLSchema-instance"/>  
<sch:ns prefix="appinfo" uri="http://release.niem.gov/niem/appinfo/5.0/"/>  
<sch:ns prefix="structures" uri="http://release.niem.gov/niem/structures/5.0/"/>

Note that the binding of the prefix xml to the XML namespace (“http://www.w3.org/XML/1998/namespace”) is implicit.

## 2.5. Additional formatting

In addition to the special formatting above, this document uses additional formatting conventions.

[square brackets]: Terms in plain [square brackets] are properties of XML information set information items, as defined by [**[XML Infoset]**](#XMLInfoset). The information items and many of the information items’ properties are defined in that document. [**[XML Schema Structures]**](#XMLSchema-1) defines additional information item properties that are contributed by validation of an [·XML document·](#definition_XML_document) against an [·XML Schema·](#definition_XML_Schema).

{curly brackets}: Terms in plain {curly brackets} are properties of [·schema components·](#definition_schema_component), as defined by [**[XML Schema Structures]**](#XMLSchema-1).

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

*Italics*: A phrase appearing in *italics* is one of:

* a title of a section of document or a rule,
* a locally-defined term, often one that is not normatively defined, or
* is emphasized for importance or prominence.

**Bold**: A phrase appearing in **bold** is one of:

* a term being defined within a definition,
* a term that was bold in the original source text for a quote
* a heading, such as for a section, a figure, a principle, definition, or rule, or
* a cross-reference within the document, to a reference to an outside document, or to a normative heading.

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

Figure 2-2: Example of an XML fragment

<xs:complexType name="PersonType">  
  ...  
</xs:complexType>

# 3. Terminology

This document uses standard terminology from other standards to explain the principles and rules that describe NIEM. In addition, it defines terms related to these other standards. This section enumerates this externally-dependent terminology.

## 3.1. IETF Best Current Practice 14 terminology

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]**](#BCP14) [**[RFC 2119]**](#RFC2119) [**[RFC 8174]**](#RFC8174) when, and only when, they appear in all capitals, as shown here.

## 3.2. XML terminology

[Definition: XML document]

The term “XML document” is as defined by [**[XML]**](#XML) [Section 2, *Documents*](http://www.w3.org/TR/2008/REC-xml-20081126/#dt-xml-doc), which states:

A data object is an XML document if it is well-formed, as defined in this specification. In addition, the XML document is valid if it meets certain further constraints.

## 3.3. XML Information Set terminology

When discussing XML documents, this document uses terminology and language as defined by [**[XML Infoset]**](#XMLInfoset).

[**[XML Infoset]**](#XMLInfoset) uses the term “information item” to describe pieces of XML documents. Documents, elements, and attributes are types of information items. The use of the term “element information item”, for example, refers to the term as defined by [**[XML Infoset]**](#XMLInfoset). Shorthand terms may also be used to refer to information items, such as [·element·](#definition_element), as defined below. The information items are identified and defined by [**[XML Infoset]**](#XMLInfoset) [Section 2, *Information Items*](http://www.w3.org/TR/2004/REC-xml-infoset-20040204/#infoitem).

[Definition: element]

An *element* is an *element information item*, as defined by [**[XML Infoset]**](#XMLInfoset) [Section 2.2, *Element Information Items*](http://www.w3.org/TR/2004/REC-xml-infoset-20040204/#infoitem.element)

[Definition: attribute]

An *attribute* is an *attribute information item*, as defined by [**[XML Infoset]**](#XMLInfoset) [Section 2.3, *Attribute Information Items*](http://www.w3.org/TR/2004/REC-xml-infoset-20040204/#infoitem.element)

[**[XML Infoset]**](#XMLInfoset) also describes properties of information items. Each class of information item carries a set of properties. Each property has a name, and the property is identified by putting the name into square brackets. For example, the element that contains an attribute is described as the [owner element] of an attribute information item, as defined in [**[XML Infoset]**](#XMLInfoset) [Section 2.3, *Attribute Information Items*](http://www.w3.org/TR/2004/REC-xml-infoset-20040204/#infoitem.attribute).

Shorthand terms for properties of information items include:

* parent (of an element): the value of the [parent] property of an element information item
* child (of an element): a member of the list of information items that is the value of the [children] property of an element information item
* owner (of an attribute): the value of the [owner element] property of an attribute information item
* **document element**: the value of the [document element] property of a document information item; preferred over the term “root element”.

## 3.4. XML Schema terminology

This document uses many terms from [**[XML Schema Structures]**](#XMLSchema-1) and [**[XML Schema Datatypes]**](#XMLSchema-2) in a normative way.

[Definition: schema component]

The term “schema component” is as defined by [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2, *XML Schema Abstract Data Model*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#key-component), which states:

**Schema component** is the generic term for the building blocks that comprise the abstract data model of the schema.

Note that this defines an abstract concept. This is not a direct reference to elements that are defined by the [·XML Schema definition language·](#X7e4888920c30d9bbda28173a5b8b96fa0c14a07); this is an abstract concept that might be realized within a tool as an in-memory model of data objects.

[Definition: XML Schema]

The term “XML Schema” is as defined by [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2, *XML Schema Abstract Data Model*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#key-schema), which states:

An **XML Schema** is a set of schema components.

Note, again, that this is an abstract concept: the set of abstract [·schema components·](#definition_schema_component) that are put together to define a schema against which an XML document might be validated.

[Definition: XML Schema definition language]

The term “XML Schema definition language” is as defined by [**[XML Schema Structures]**](#XMLSchema-1) [subsection *Abstract*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#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. The schema language, which is itself represented in XML 1.0 and uses namespaces, substantially reconstructs and considerably extends the capabilities found in XML 1.0 document type definitions (DTDs).

This describes the XML syntax (and related semantics) defined by the XML Schema specifications. It is through the [·XML Schema definition language·](#X7e4888920c30d9bbda28173a5b8b96fa0c14a07) that a complex type definition schema component is created using the xs:complexType element.

[Definition: schema document]

The term “schema document” is as defined by [**[XML Schema Structures]**](#XMLSchema-1) [Section 3.1.2, *XML Representations of Components*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#key-schemaDoc), which states:

A document in this form (i.e. a <schema> element information item) is a **schema document**.

This definition describes an [·XML document·](#definition_XML_document) that follows the syntax of the [·XML Schema definition language·](#X7e4888920c30d9bbda28173a5b8b96fa0c14a07).

[Definition: valid]

The term “valid” is as defined by [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.1, *Overview of XML Schema*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#key-vn), which states:

[Definition:] the word **valid** and its derivatives are used to refer to clause 1 above, the determination of local schema-validity.

The referenced clause 1 is a part of a description of schema-validity:

Schema-validity assessment has two aspects:

1. Determining local schema-validity, that is whether an element or attribute information item satisfies the constraints embodied in the relevant components of an XML Schema;
2. Synthesizing an overall validation outcome for the item, combining local schema-validity with the results of schema-validity assessments of its descendants, if any, and adding appropriate augmentations to the infoset to record this outcome.

In addition, this specification locally defines terms relevant to XML Schema concepts:

[Definition: instance document]

An *instance document* (of an [·XML Schema·](#definition_XML_Schema)) is an [·XML document·](#definition_XML_document) that is [·valid·](#definition_valid) against the [·XML Schema·](#definition_XML_Schema).

The term “instance document” is used with [**[XML Schema Structures]**](#XMLSchema-1), but is not defined therein.

[Definition: XML Schema document set]

An *XML Schema document set* is a set of [·schema documents·](#definition_schema_document) that together define an [·XML Schema·](#definition_XML_Schema) suitable for assessing the [·validity·](#definition_valid) of an [·XML document·](#definition_XML_document).

Schema assembly is a tricky topic that is not resolved by this document. Other specifications may express specifics about the process of turning a set of [·schema documents·](#definition_schema_document) into an [·XML Schema·](#definition_XML_Schema). Methods used may include use of tool-specific schema caches and mappings, use of XML catalogs and entity resolvers, use of schemaLocation attributes on xs:import elements, and xsi:schemaLocation attributes in XML documents, among others. The topic of schema assembly is discussed in [Section 6.2.10, *Schema locations provided in schema documents are hints*, below](#section_6.2.10). This specification abstracts away details of schema assembly through the use of XPath functions described by [Section 2.4.3, *Normative XPath functions*, above](#section_2.4.3).

### 3.4.1. Schema components

In this document, the name of a referenced schema component may appear without the suffix “schema component” to enhance readability of the text. For example, the term “complex type definition” may be used instead of “complex type definition schema component”.

[Definition: base type definition]

The term “base type definition” is as defined by [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2.1.1, *Type Definition Hierarchy*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#key-baseTypeDefinition), which states:

A type definition used as the basis for an extension or restriction is known as the base type definition of that definition.

[Definition: simple type definition]

The term “simple type definition” is as defined by [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2.1.2, *Simple Type Definition*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#Simple_Type_Definition).

[Definition: complex type definition]

The term “complex type definition” is as defined by [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2.1.3, *Complex Type Definition*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#Complex_Type_Definition).

[Definition: element declaration]

The term “element declaration” is as defined by [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2.2.1, *Element Declaration*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#Element_Declaration).

### 3.4.2. Schema information set contributions

As described in [Section 3.3, *XML Information Set terminology*, above](#section_3.3), the XML Information Set specification defined properties of the content of XML documents. The XML Schema specification also provides properties of the content of XML documents. These properties are called Schema information set contribution, as described by [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.3, *Constraints and Validation Rules*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#gloss-sic), which defines them as:

Augmentations to post-schema-validation infosets expressed by schema components, which follow as a consequence of validation and/or assessment.

This document uses these property terms within definitions and other text. Terms used include:

* [[type definition] (of an element)](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#e-type_definition): The type of the element as determined at run-time. This will reflect the use of the attribute xsi:type in an XML document.

## 3.5. XML Namespaces terminology

This document uses XML Namespaces as defined by [**[XML Namespaces]**](#XMLNamespaces).

## 3.6. Conformance Targets Attribute Specification terminology

[**[CTAS]**](#CTAS) defines several terms used normatively within this specification.

[Definition: conformance target]

The term “conformance target” is as defined by [**[CTAS]**](#CTAS), which states:

A *conformance target* is a class of artifact, such as an interface, protocol, document, platform, process or service, that is the subject of conformance clauses and normative statements. There may be several conformance targets defined within a specification, and these targets may be diverse so as to reflect different aspects of a specification. For example, a protocol message and a protocol engine may be different conformance targets.

[Definition: conformance target identifier]

The term “conformance target identifier” is as defined by [**[CTAS]**](#CTAS), which states:

A *conformance target identifier* is an internationalized resource identifier that uniquely identifies a conformance target.

[Definition: effective conformance target identifier]

The term “effective conformance target identifier” is as defined by [**[CTAS]**](#CTAS) [Section 4, *Semantics and Use*](http://reference.niem.gov/niem/specification/conformance-targets-attribute/3.0/NIEM-CTAS-3.0-2014-07-31.html#definition_effective_conformance_target_identifier), which states:

An *effective conformance target identifier* of a conformant document is an internationalized resource identifier reference that occurs in the document’s effective conformance targets attribute.

# 4. Conformance targets

## 4.1. Conformance targets defined

This section defines and describes conformance targets of this specification. Each conformance target has a formal definition, along with a notional description of the characteristics and intent of each. These include:

* [Section 4.1.1, *Reference schema document*](#section_4.1.1)
* [Section 4.1.2, *Extension schema document*](#section_4.1.2)
* [Section 4.1.3, *Schema document set*](#section_4.1.3)
* [Section 4.1.4, *Instance documents and elements*](#section_4.1.4)

### 4.1.1. Reference schema document

[Definition: reference schema document]

A **reference schema document** is a [·schema document·](#definition_schema_document) that is intended to provide the authoritative definitions of broadly reusable [·schema components·](#definition_schema_component). It is a [·conformance target·](#definition_conformance_target) of this specification. A reference schema document MUST conform to all rules of this specification that apply to this conformance target. An [·XML document·](#definition_XML_document) with a [·conformance target identifier·](#definition_conformance_target_identifier) of http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ReferenceSchemaDocument MUST be a conformant reference schema document.

A [·reference schema document·](#definition_reference_schema_document) is a [·schema document·](#definition_schema_document) that is intended to be the authoritative definition schema for a namespace. Examples include NIEM Core and NIEM domains.

Some characteristics of a [·reference schema document·](#definition_reference_schema_document):

* It is explicitly designated as a reference schema via the conformance targets attribute, per [Rule 4-5, *Schema claims reference schema conformance target*, below](#rule_4-5).
* It provides the broadest, most fundamental definitions of components in its namespace.
* It provides the authoritative definition of business semantics for components in its namespace.
* It is intended to serve as the basis for components in information exchanges and extension schema documents.
* It satisfies all rules specified in the Naming and Design Rules for reference schemas.

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

The rules for reference schema documents are more stringent than are the rules for other classes of NIEM-conformant schemas. Reference schema documents are intended to support the broadest reuse. They are very uniform in their structure. As they are the primary definitions for schema components, they do not need to restrict other data definitions, and they are not allowed to use XML Schema’s restriction mechanism (e.g., [Rule 9-30, *Complex content uses extension*, below](#rule_9-30)). Reference schema documents are intended to be as regular and simple as possible.

Many reference schemas are **optional and over-inclusive**. Data definitions in namespaces defined by reference schemas are designed with parts that are intended to be omitted or refined as needed for a particular exchange. Many reference schemas define more complex types than any individual exchange will need and define complex types that have more properties, with broader cardinality, than an individual exchange will need. Data definitions within reference schemas are designed to be a basis that is refined and specialized for a particular exchange. Developers of information exchanges are expected to subset, profile, and extend reference schemas to construct precise data definitions to match their information exchange requirements.

### 4.1.2. Extension schema document

[Definition: extension schema document]

An **extension schema document** is a [·schema document·](#definition_schema_document) that is intended to provide definitions of [·schema components·](#definition_schema_component) that are intended for reuse within a more narrow scope than those defined by a [·reference schema document·](#definition_reference_schema_document). It is a [·conformance target·](#definition_conformance_target) of this specification. An extension schema document MUST conform to all rules of this specification that apply to this conformance target. An XML document with a [·conformance target identifier·](#definition_conformance_target_identifier) of http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ExtensionSchemaDocument MUST be an extension schema document.

Characteristics of an [·extension schema document·](#definition_extension_schema_document) include:

* It is explicitly designated as an [·extension schema document·](#definition_extension_schema_document) via the conformance targets attribute.
* It provides the authoritative definition of business semantics for components in its namespace.
* It contains components that, when appropriate, use or are derived from the components in [·reference schema documents·](#definition_reference_schema_document).
* It is intended to express the additional vocabulary required for an information exchange, above and beyond the vocabulary available from reference schemas, and to also support additional XML Schema validation requirements for an exchange.
* It satisfies all rules specified in this document for [·extension schema documents·](#definition_extension_schema_document).

An extension schema in an information exchange specification serves several functions. First, it defines new content within a new namespace, which may be an exchange-specific namespace or a namespace shared by several exchanges. This content is NIEM-conformant but has fewer restrictions on it than do [·reference schema documents·](#definition_reference_schema_document). Second, the [·extension schema document·](#definition_extension_schema_document) bases its content on content from [·reference schema documents·](#definition_reference_schema_document), where appropriate. Methods of deriving content include using (by reference) existing [·schema components·](#definition_schema_component), as well as creating extensions and restrictions of existing components.

For example, an information exchange specification may define a type for an exchange-specific phone number and base that type on a type defined by the NIEM Core reference schema document. This exchange-specific phone number type may restrict the NIEM Core type to limit those possibilities that are permitted of the base type. Exchange extensions and restrictions must include annotations and documentation to be conformant, but they are allowed to use restriction, choice, and some other constructs that are not allowed in [·reference schema documents·](#definition_reference_schema_document).

Note that exchange specifications may define schemas that meet the criteria of reference schemas for those components that its developers wish to nominate for later inclusion in NIEM Core or in domains.

### 4.1.3. Schema document set

A [·conformant schema document set·](#Xeff662d8d67f5b4e72a2955d729e0ac83cad2d6) is a set of schema documents that are capable of validating XML documents.

[Definition: conformant schema document set]

A **conformant schema document set** is a collection of [·schema documents·](#definition_schema_document) that together are capable of [·validating·](#definition_valid) a [·conformant instance XML document·](#X6a4ab7051f7203e0b47588881449d56367bed27). It is a [·conformance target·](#definition_conformance_target) of this specification. A conformant schema document set MUST conform to all rules of this specification that apply to this conformance target.

A [·conformant schema document set·](#Xeff662d8d67f5b4e72a2955d729e0ac83cad2d6) has strong dependencies on [·reference schema documents·](#definition_reference_schema_document) and [·extension schema documents·](#definition_extension_schema_document). Without the guarantees provided by those conformance targets, the rules for a [·conformant schema document set·](#Xeff662d8d67f5b4e72a2955d729e0ac83cad2d6) would not be helpful. Those schemas in a schema set that are marked as reference or extension schemas are required to conform to the appropriate conformance targets.

Rule 4-1. Schema marked as reference schema document must conform

[Rule 4-1] (SET) (Constraint)

Any [·schema document·](#definition_schema_document) with an [·effective conformance target identifier·](#X6d0b4369562c0703761463ba00cd0e2b474de22) of http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ReferenceSchemaDocument MUST be a [·reference schema document·](#definition_reference_schema_document).

Rule 4-2. Schema marked as extension schema document must conform

[Rule 4-2] (SET) (Constraint)

Any [·schema document·](#definition_schema_document) with an [·effective conformance target identifier·](#X6d0b4369562c0703761463ba00cd0e2b474de22) of http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ExtensionSchemaDocument MUST be an [·extension schema document·](#definition_extension_schema_document).

### 4.1.4. Instance documents and elements

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

[Definition: conformant instance XML document]

A **conformant instance XML document** is an [·XML document·](#definition_XML_document) that is an [·instance document·](#definition_instance_document) [·valid·](#definition_valid) to a [·conformant schema document set·](#Xeff662d8d67f5b4e72a2955d729e0ac83cad2d6). It is a [·conformance target·](#definition_conformance_target) of this specification. A conformant instance XML document MUST conform to all rules of this specification that apply to this conformance target.

Characteristics of a [·conformant instance XML document·](#X6a4ab7051f7203e0b47588881449d56367bed27) include:

* The [·document element·](#definition_document_element) is locally schema-valid.
* Each element information item within the [·XML document·](#definition_XML_document) that has property [namespace name] matching the target namespace of a [·reference schema document·](#definition_reference_schema_document) or [·extension schema document·](#definition_extension_schema_document) is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7).

Schema-validity may be assessed against a single set of schemas or against multiple sets of schemas.

Assessment against schemas may be directed by a message specification, such as an information exchange package description (IEPD), model package description (MPD), or information exchange specification, or by other instructions or tools.

Note that this specification does not require the [·document element·](#definition_document_element) of a [·conformant instance XML document·](#X6a4ab7051f7203e0b47588881449d56367bed27) to be a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7). Other specifications may add additional conformance constraints for elements within an exchange.

[Definition: conformant element information item]

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

* Its [namespace name] and [local name] matches an element declared by a [·reference schema document·](#definition_reference_schema_document) or [·extension schema document·](#definition_extension_schema_document).
* It occurs within a [·conformant instance XML document·](#X6a4ab7051f7203e0b47588881449d56367bed27).
* It is locally schema-valid.
* It satisfies all rules specified in this document applicable to an element in a [·conformant instance XML document·](#X6a4ab7051f7203e0b47588881449d56367bed27).

Because each NIEM-conformant element information item must be locally schema-valid, each element must validate against the schema definition of the element, even if the element information item is allowed within the document because of a wildcard that the {process contents} with a value of “skip”. As described by [**[XML Schema Structures]**](#XMLSchema-1) [Section 3.10.1, *The Wildcard Schema Component*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#process_contents), the content of an element introduced by a wildcard with {process contents} set to “skip” does not have any schema validity constraint; it is only required to be well-formed XML. Within a NIEM-conformant XML document, each element that is from a NIEM namespace conforms to its schema specification.

## 4.2. Applicability of rules to conformance targets

Each rule within this document is applicable to one or more of the conformance targets identified by [Section 4.1, *Conformance targets defined*, above](#section_4.1). Each rule identifies its applicability as described in [Section 2.4.1, *Rules*, above](#section_2.4.1). The applicability field of each rule will contain one or more code values from [Table 4-1, *Codes representing conformance targets*, below](#table_4-1). A rule is applicable to the identified conformance targets.

Table 4-1: Codes representing conformance targets

|  |  |
| --- | --- |
| Code | Conformance target |
| REF | [·reference schema document·](#definition_reference_schema_document) |
| EXT | [·extension schema document·](#definition_extension_schema_document) |
| SET | [·conformant schema document set·](#Xeff662d8d67f5b4e72a2955d729e0ac83cad2d6) |
| INS | [·conformant instance XML document·](#X6a4ab7051f7203e0b47588881449d56367bed27) |

## 4.3. Conformance target identifiers

A conformant schema document claims to be conformant thorough the use of a set of [·conformance target identifiers·](#definition_conformance_target_identifier).

Rule 4-3. Schema is CTAS-conformant

[Rule 4-3] (REF, EXT) (Constraint)

The schema document MUST be a conformant document as defined by the NIEM Conformance Targets Attribute Specification.

The term “conformant document” is defined by [**[CTAS]**](#CTAS) [Section 3.2, *Conformance to this Specification*](http://reference.niem.gov/niem/specification/conformance-targets-attribute/3.0/NIEM-CTAS-3.0-2014-07-31.html#section_3.2).

Rule 4-4. Document element has attribute ct:conformanceTargets

[Rule 4-4] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[. is nf:get-document-element(.)  
                       or exists(@ct:conformanceTargets)]">  
    <sch:assert test="(. is nf:get-document-element(.)) = exists(@ct:conformanceTargets)"  
      >The [document element] of the XML document, and only the [document element], MUST own an attribute {http://release.niem.gov/niem/conformanceTargets/3.0/}conformanceTargets.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 4-5. Schema claims reference schema conformance target

[Rule 4-5] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[. is nf:get-document-element(.)]">  
    <sch:assert test="nf:has-effective-conformance-target-identifier(., xs:anyURI('http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ReferenceSchemaDocument'))"  
      >The document MUST have an effective conformance target identifier of http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ReferenceSchemaDocument.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the term [·effective conformance target identifier·](#X6d0b4369562c0703761463ba00cd0e2b474de22).

Rule 4-6. Schema claims extension conformance target

[Rule 4-6] (EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[. is nf:get-document-element(.)]">  
    <sch:assert test="nf:has-effective-conformance-target-identifier(., xs:anyURI('http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ExtensionSchemaDocument'))"  
      >The document MUST have an effective conformance target identifier of http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ExtensionSchemaDocument.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the term [·effective conformance target identifier·](#X6d0b4369562c0703761463ba00cd0e2b474de22).

# 5. The NIEM conceptual model

This section describes aspects of the RDF model, and provides a mapping between NIEM concepts and the RDF model.

* [Section 5.1, *Purpose of the NIEM conceptual model*](#section_5.1)
* [Section 5.2, *The RDF model*](#section_5.2)
* [Section 5.3, *NIEM in terms of RDF*](#section_5.3)
* [Section 5.4, *Unique identification of data objects*](#section_5.4)
* [Section 5.5, *NIEM data is explicit, not implicit*](#section_5.5)
* [Section 5.6, *Mapping of NIEM concepts to RDF concepts*](#section_5.6)

## 5.1. Purpose of the NIEM conceptual model

Each release of NIEM provides a concrete data model, in the form of a set of [·schema documents·](#definition_schema_document). These schema documents may be used to build messages and information exchanges. The schema documents spell out what kinds of objects exist and how those objects may be related. A set of XML data that follows the rules of NIEM implies specific meaning. The varieties of [·schema components·](#definition_schema_component) used within conformant schema documents are selected to clarify the meaning of XML data. That is, schema components that do not have a clear meaning have been avoided. NIEM provides a framework within which XML data has a specific meaning.

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

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

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

* NIEM’s conceptual model is defined by a recognized standard.
* NIEM’s conceptual model is very well defined.
* NIEM’s conceptual model provides a consistent basis for relating attributes, elements, types, and other XML Schema components.
* NIEM’s use of the RDF model defines what a set of NIEM data means. The RDF specification provides a detailed description of what a statement means. This meaning is leveraged by NIEM.
* NIEM’s use of the RDF model provides a basis for inferring and reasoning about XML data that uses NIEM. That is, using the rules defined for the RDF model, programs can determine implications of relationships between NIEM-defined objects.

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

This section defines the meaning of NIEM-conformant XML data and schemas through the definition of mappings from XML data and schema to RDF data and schema.

Rule 5-1. structures:uri denotes resource identifier

[Rule 5-1] (INS) (Interpretation)

The interpretation of a [·conformant instance XML document·](#X6a4ab7051f7203e0b47588881449d56367bed27) MUST be consistent with an RDFS interpretation of the RDF graph composed of the RDF entailed by the XML document and the RDF entailed by the schema.

The interpretation of NIEM-conformant data and schemas are in place to ensure that a precise meaning can be derived from data. That is, the data makes specific assertions, which are well understood since they are derived from the data, the schema, and the NIEM rules.

## 5.2. The RDF model

This section identifies features of RDF and RDFS, in order to establish a mapping between RDF semantics and NIEM. A reader should read the referenced source documents to obtain a full understanding of the concepts mentioned in this section.

RDF establishes a graph-based data model, as described by [**[RDF Concepts]**](#RDFConcepts) [Section 1.1, *Graph-based Data Model*](http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#data-model), which states:

The core structure of the abstract syntax is a set of triples, each consisting of a subject, a predicate and an object. A set of such triples is called an RDF graph.

[**[RDF Concepts]**](#RDFConcepts) also states:

There can be three kinds of nodes in an RDF graph: IRIs, literals, and blank nodes.

[**[RDF Concepts]**](#RDFConcepts) [Section 1.2, *Resources and Statements*](http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#resources-and-statements) describes resources:

Any IRI or literal **denotes** something in the world (the “universe of discourse”). These things are called **resources**. Anything can be a resource, including physical things, documents, abstract concepts, numbers and strings; the term is synonymous with “entity” as it is used in the RDF Semantics specification. The resource denoted by an IRI is called its referent, and the resource denoted by a literal is called its literal value.

[**[RDF Concepts]**](#RDFConcepts) also describes relationships and blank nodes.

Asserting an RDF triple says that some relationship, indicated by the predicate, holds between the resources denoted by the subject and object. This statement corresponding to an RDF triple is known as an RDF statement. The predicate itself is an IRI and denotes a property, that is, a resource that can be thought of as a binary relation. (Relations that involve more than two entities can only be indirectly expressed in RDF.)

Unlike IRIs and literals, blank nodes do not identify specific resources. Statements involving blank nodes say that something with the given relationships exists, without explicitly naming it.

[**[RDF Concepts]**](#RDFConcepts) [Section 1.7, *Equivalence, Entailment and Inconsistency*](http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#dfn-logical-expression) describes the meaning of an RDF triple:

An RDF triple encodes a statement—a simple logical expression, or claim about the world. An RDF graph is the conjunction (logical AND) of its triples.

[**[RDF Concepts]**](#RDFConcepts) [Section 3.1, *Triples*](http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#dfn-rdf-triple) defines an RDF triple:

An RDF triple consists of three components:

* the subject, which is an IRI or a blank node
* the predicate, which is an IRI
* the object, which is an IRI, a literal or a blank node

An RDF triple is conventionally written in the order subject, predicate, object.

[**[RDF Concepts]**](#RDFConcepts) [Section 4, *RDF Datasets*](http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#dfn-rdf-dataset) defines an RDF dataset:

An RDF dataset is a collection of RDF graphs, and comprises:

* Exactly one **default graph**, being an RDF graph. The default graph does not have a name and MAY be empty.
* Zero or more **named graphs**. Each named graph is a pair consisting of an IRI or a blank node (the **graph name**), and an RDF graph. Graph names are unique within an RDF dataset.

## 5.3. NIEM in terms of RDF

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

This NDR discusses NIEM data in XML terminology, complex types and elements, rather than using RDF terms, resources and properties. NIEM objects and associations coincide with RDF resources; both objects and associations correspond to RDF resources with additional constraints:

NIEM associations are defined as n-ary properties, as described in [**[N-ary]**](#N-ary), [“Use Case 3: N-ary relation with no distinguished participant”](http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/#useCase3). NIEM associations are defined in [Section 10.3, *Associations*, below](#section_10.3). Assertions are made via NIEM-conformant XML data, described by [Section 12, *XML instance document rules*, below](#section_12).

The XML Schema types that define NIEM objects and associations are related to each other via elements and attributes. That is, a type contains elements and attributes, and an element or attribute has a value that is an instance of an XML Schema type. These elements and attributes are XML Schema representations, which correspond to RDF properties. NIEM-conformant XML Schemas describe things and their properties. NIEM-conformant data contains elements and attributes. These correspond to RDF resources and their properties, which describe their characteristics and relationships.

Although within an XML document, the XML element children of a parent element have a specific order, that element order is not reflected in the NIEM conceptual model. NIEM provides for the order of properties relative to each other to be expressed using the attribute structures:sequenceID, as defined by [Section 12.3, *Property order and sequence identifiers*, below](#section_12.3). Without the use of structures:sequenceID, the NIEM conceptual model does not define relative order of properties of an object, which means that processors may present properties in whatever order is most convenient, natural, or appropriate. Should a particular order of elements be desired, it should be expressed using structures:sequenceID. Should a data set include structures:sequenceID, it must be respected. This specification does not define a mapping for this mechanism to RDF.

This document describes most RDF data as triples, omitting a graph name. Users of NIEM data may assign these triples to a named graph, as needed. NIEM data only explicitly assigns triples into specific named graphs to support the use of structures:relationshipMetadata, which attributes metadata to triples using named graphs identified by metadata objects.

## 5.4. Unique identification of data objects

A NIEM data exchange may be ephemeral and ad hoc. That is, a message may be transmitted without any expectation of persistence. Such a message exists only to exchange data and may not have any universal meaning beyond that specific exchange. As such, a message may or may not have a URI as an identifier. NIEM was designed with the assumption that a given exchange need not have any unique identifier; NIEM does not require a unique identifier. NIEM also does not require any object carried by a message to be identified by a URI.

A NIEM-conformant instance XML document may carry any of these attributes to identify objects within messages:

* Attribute **xml:base** (of type xs:anyURI) is defined by [**[XML Base]**](#XMLBase) [Section 3, *xml:base Attribute*](http://www.w3.org/TR/2009/REC-xmlbase-20090128/#syntax), which states:

The attribute xml:base may be inserted in XML documents to specify a base URI other than the base URI of the document or external entity.

An XML document has an implicit base URI, the identifier of the document itself. This attribute allows the base URI to be made explicit within a NIEM XML document.

* Attribute **structures:uri** (of type xs:anyURI) may appear within an XML element to define a URI for that element. This may be an absolute URI (e.g., http://example.org/incident182#person12), or may be a relative URI (e.g., #person12). Relative URIs are resolved against a URI determined by the xml:base attributes in scope, falling back to the base URI of the containing document.
* Attribute **structures:id** (of type xs:ID) provides a document-relative identifier for an element. Semantically, “structures:id="abe92"” is equivalent to “structures:uri="#abe92"”.
* Attribute **structures:ref** (of type xs:IDREF) provides a reference to another element within a document, by providing a value of a structures:id attribute within the document. Semantically, “structures:ref="abe92"” is equivalent to “structures:uri="#abe92"”

The values of URIs and IDs within NIEM XML documents are not presumed to have any particular significance. XML requires every ID to be unique within its XML document, and for every IDREF to refer to an ID within the same document. The mapping of IDs and IDREFs to URIs does not mean that the identifiers are persistent or significant.

These attributes provide the identifiers of objects. The properties of an object may be spread across several XML elements that have the same identifier. These properties must be merged together to provide all the properties of a single object, as described by [**[JSON LD]**](#JSONLD) [Section 8.2, *Node Objects*](https://www.w3.org/TR/2014/REC-json-ld-20140116/#node-objects):

The properties of a node in a graph may be spread among different node objects within a document. When that happens, the keys of the different node objects need to be merged to create the properties of the resulting node.

Mapping of NIEM data to RDF frequently involves the use of blank nodes, instead of universally-meaningful resource IRIs.

The identifier of an object is constructed using the above attributes; if the above attributes do not appear on an object, then an object may be treated as a blank node, and assigned a blank node identifier.

## 5.5. NIEM data is explicit, not implicit

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

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

## 5.6. Mapping of NIEM concepts to RDF concepts

This section provides RDF implementations for many aspects of NIEM-conformant schemas and instance documents.

[**[N-Quads]**](#N-Quads) [Section 2, *N-Quads Language*](https://www.w3.org/TR/2014/REC-n-quads-20140225/#n-quads-language) defines a plain text format for encoding an RDF dataset:

The simplest statement is a sequence of (subject, predicate, object) terms forming an RDF triple and an optional blank node label or IRI labeling what graph in a dataset the triple belongs to, all are separated by whitespace and terminated by “.” after each statement.

<http://example.org/#spiderman> <http://www.perceive.net/schemas/relationship/enemyOf> <http://example.org/#green-goblin> <http://example.org/graphs/spiderman> .

The graph label IRI can be omitted, in which case the triples are considered part of the default graph of the RDF dataset.

RDF examples and templates within this document are provided using a modified N-Quads format, where qualified names (e.g., nc:PersonType) and variables (e.g., *$object* may be substituted for full URIs and their surrounding angle brackets. Within this section, the following substitutions apply:

* Prefix rdf denotes http://www.w3.org/1999/02/22-rdf-syntax-ns#.
* rdf:type denotes IRI http://www.w3.org/1999/02/22-rdf-syntax-ns#type.
* rdf:value denotes IRI http://www.w3.org/1999/02/22-rdf-syntax-ns#value.
* Prefix rdfs denotes http://www.w3.org/2000/01/rdf-schema#.
* rdfs:subClassOf denotes IRI http://www.w3.org/2000/01/rdf-schema#subClassOf.
* rdfs:subPropertyOf denotes IRI http://www.w3.org/2000/01/rdf-schema#subPropertyOf.
* rdfs:range denotes IRI http://www.w3.org/2000/01/rdf-schema#range.
* Prefix structures denotes http://release.niem.gov/niem/structures/5.0/.
* structures:metadata denotes IRI http://release.niem.gov/niem/structures/5.0/metadata.

### 5.6.1. Resource IRIs for XML Schema components and information items

The term “qualified name” is defined by [**[XML Namespaces]**](#XMLNamespaces) [Section 2.1, *Basic Concepts*](http://www.w3.org/TR/2009/REC-xml-names-20091208/#dt-qualname), which states:

A **qualified name** is a name subject to namespace interpretation.

A QName is used to represent a qualified name, as described by [**[XML Schema Datatypes]**](#XMLSchema-2) [Section 3.2.18, *QName*](http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/#QName), which states:

The value space of QName is the set of tuples {namespace name, local part}, where namespace name is an anyURI and local part is an NCName.

Certain components defined by NIEM schemas and instances have corresponding resource IRIs. Each IRI is taken from a qualified name, as follows:

* If namespace name ends with “#”: concatenate(namespace name, local part)
* Otherwise: concatenate(namespace name, “#”, local part)

Note that this is only meaningful when the namespace name is not empty and is an absolute URI.

The corresponding RDF resource IRIs for information items and schema components are:

* element information item or attribute information item: the resource IRI is built from the qualified name constructed from its [namespace name] and [local name] properties.
* schema component: the resource IRI is built from the qualified name constructed from its {target namespace} and {name} properties.

### 5.6.2. RDF Literals

A simple value may be incorporated into a [·conformant instance XML document·](#X6a4ab7051f7203e0b47588881449d56367bed27) as the value of an attribute information item, or as a child of an element information item. This section describes how a simple value is mapped to an RDF literal. Note that there is no mapping for the simple content of an element that is not a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7), nor for attributes defined by non-conformant schema documents, so there is no accommodation of mixed content, untyped values, or other cases outside of what conformant schema documents define.

For each occurrence of a simple value, the following may be relevant:

* The value of the literal, which is a normalized value of an attribute or element information item processed in accordance with [**[XML]**](#XML) [Section 2.10, *White Space Handling*](http://www.w3.org/TR/2008/REC-xml-20081126/#sec-white-space) and [**[XML Schema Structures]**](#XMLSchema-1) [Section 3.1.4, *White Space Normalization during Validation*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#d0e1654).
* The occurrence of an attribute xml:lang applicable to the value (described by [**[XML]**](#XML) [Section 2.12, *Language Identification*](http://www.w3.org/TR/2008/REC-xml-20081126/#sec-lang-tag)), which may entail a language tag on the literal (described by [**[RDF Concepts]**](#RDFConcepts) [Section 3.3, *Literal*](http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#dfn-language-tag)).
* The XML Schema-defined base type of the simple value, which may be an attribute’s {type definition}, or a simple type base type of an element’s {type definition}.

The literal for a simple value *$value* is:

* If *$value* has a base type definition that is derived from type xs:string (and not an XML Schema-defined type derived from xs:string), and a non-empty language specification is applied to *$value* using xml:lang, as described by [**[XML]**](#XML) [Section 2.12, *Language Identification*](http://www.w3.org/TR/2008/REC-xml-20081126/#sec-lang-tag), then the literal is a language-tagged string, as described by [**[RDF Concepts]**](#RDFConcepts) [Section 3.3, *Literals*](http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#dfn-language-tagged-string):

"$lexical-form"@$language-tag

Where:

* *$lexical-form* is a Unicode string for *$value*.
* *$language-tag* is the in-scope value of attribute xml:lang.
* Otherwise, if *$value* has a base type definition *$base-type* that is listed as an RDF-compatible XSD type in [**[RDF Concepts]**](#RDFConcepts) [Section 5.1, *The XML Schema Built-in Datatypes*](http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/#xsd-datatypes), and *$base-type* is not xs:string, then the literal is:

"$lexical-form"^^$datatype-IRI

Where:

* *$lexical-form* is a Unicode string for *$value*.
* *$datatype-IRI* is the QName of *$base-type*.
* Otherwise, the literal is a simple literal, which is:

"$lexical-form"

Where: *$lexical-form* is a Unicode string for *$value*.

### 5.6.3. Instance document mapped to RDF

This section has the following subsections:

* [Section 5.6.3.1, *Instance document*](#section_5.6.3.1)
* [Section 5.6.3.2, *Element as a simple property without relationship metadata*](#section_5.6.3.2)
* [Section 5.6.3.3, *Element as a simple property with relationship metadata*](#section_5.6.3.3)
* [Section 5.6.3.4, *Element simple value without relationship metadata*](#section_5.6.3.4)
* [Section 5.6.3.5, *Element simple value with relationship metadata*](#section_5.6.3.5)
* [Section 5.6.3.6, *Attribute as a simple property without relationship metadata*](#section_5.6.3.6)
* [Section 5.6.3.7, *Attribute as a simple property, with relationship metadata*](#section_5.6.3.7)
* [Section 5.6.3.8, *Elements and attributes via an augmentation type*](#section_5.6.3.8)
* [Section 5.6.3.9, *Properties applied via structures:metadata*](#section_5.6.3.9)

#### 5.6.3.1. Instance document

A [·conformant instance XML document·](#X6a4ab7051f7203e0b47588881449d56367bed27) entails a dataset, described by mappings of its content to RDF triples and quads.

#### 5.6.3.2. Element as a simple property without relationship metadata

Given:

* *$predicate-element* is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7), and is an instance of an [·object type·](#definition_object_type) or [·association type·](#definition_association_type).
* *$context-element* is parent of *$predicate-element*, is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7), and is an instance of an [·object type·](#definition_object_type), [·association type·](#definition_association_type), or [·metadata type·](#definition_metadata_type).
* *$predicate-element* either owns has no attribute structures:relationshipMetadata or it owns an attribute structures:relationshipMetadata that contains no items.

The following RDF is entailed:

$context-node-id $predicate-id $predicate-node-id .

Where:

* *$context-node-id* is a node identifier for the object held by *$context-element*.
* *$predicate-id* is the IRI for *$predicate-element*.
* *$predicate-node-id* is a node identifier for the object held by *$predicate-element*.

#### 5.6.3.3. Element as a simple property with relationship metadata

Given:

* *$predicate-element* is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7), and is an instance of an [·object type·](#definition_object_type) or [·association type·](#definition_association_type).
* *$context-element* is parent of *$predicate-element*, is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7), and is an instance of an [·object type·](#definition_object_type), [·association type·](#definition_association_type), or [·metadata type·](#definition_metadata_type).
* *$predicate-element* owns attribute *$relationship-metadata-attribute* that has name structures:relationshipMetadata, and that contains one or more references to metadata objects.

For each item *$metadata-reference* in the list held by *$relationship-metadata-attribute*, the following RDF quad is entailed:

$context-node-id $predicate-id $predicate-node-id $metadata-node-id .

Where:

* *$context-node-id* is a node identifier for the object held by *$context-element*.
* *$predicate-id* is the IRI for *$predicate-element*.
* *$predicate-node-id* is a node identifier for the object held by *$predicate-element*.
* *$metadata-node-id* is a node identifier for the object referenced by *$metadata-reference*.

#### 5.6.3.4. Element simple value without relationship metadata

Given:

* *$context-element* is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7) that is an instance of an [·object type·](#definition_object_type).
* *$context-element* either owns has no attribute structures:relationshipMetadata or it owns an attribute structures:relationshipMetadata that contains no items.
* *$context-element* has a non-empty simple value.

The following RDF is entailed:

$context-node-id rdf:value $literal .

Where:

* *$context-node-id* is a node identifier for the object held by *$context-element*.
* rdf:value denotes IRI http://www.w3.org/1999/02/22-rdf-syntax-ns#value.
* *$literal* is the literal value for *$context-element*, as described in [Section 5.6.2, *RDF Literals*, above](#section_5.6.2).

#### 5.6.3.5. Element simple value with relationship metadata

Given:

* *$context-element* is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7) that is an instance of an [·object type·](#definition_object_type).
* *$context-element* owns attribute *$relationship-metadata-attribute* that has name structures:relationshipMetadata, and that contains one or more references to metadata objects.
* *$context-element* has a non-empty simple value.

For each item *$metadata-reference* in the list held by *$relationship-metadata-attribute*, the following RDF quad is entailed:

$context-node-id rdf:value $literal $metadata-node-id .

Where:

* *$context-node-id* is a node identifier for the object held by *$context-element*.
* rdf:value denotes IRI http://www.w3.org/1999/02/22-rdf-syntax-ns#value.
* *$literal* is the literal value for *$context-element*, as described in [Section 5.6.2, *RDF Literals*, above](#section_5.6.2).
* *$metadata-node-id* is a node identifier for the object referenced by *$metadata-reference*.

#### 5.6.3.6. Attribute as a simple property without relationship metadata

Given:

* *$context-element* is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7) and is an instance of an [·object type·](#definition_object_type), [·association type·](#definition_association_type), or [·metadata type·](#definition_metadata_type).
* *$predicate-attribute* is a attribute that has owner *$context-element*, and has property [attribute declaration] that is defined by a [·reference schema document·](#definition_reference_schema_document) or an [·extension schema document·](#definition_extension_schema_document).
* *$context-element* either owns has no attribute structures:relationshipMetadata or it owns an attribute structures:relationshipMetadata that contains no items.

The following RDF is entailed:

$context-node-id $predicate-id $literal .

Where:

* *$context-node-id* is a node identifier for the object held by *$context-element*.
* *$predicate-id* is the resource IRI for *$predicate-attribute*.
* *$literal* is the literal value for *$attribute*, as described in [Section 5.6.2, *RDF Literals*, above](#section_5.6.2).

#### 5.6.3.7. Attribute as a simple property, with relationship metadata

Given:

* *$context-element* is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7) and is an instance of an [·object type·](#definition_object_type), [·association type·](#definition_association_type), or [·metadata type·](#definition_metadata_type).
* *$predicate-attribute* is a attribute that has owner *$context-element*, and has property [attribute declaration] that is defined by a [·reference schema document·](#definition_reference_schema_document) or an [·extension schema document·](#definition_extension_schema_document).
* *$context-element* owns attribute *$relationship-metadata-attribute* that has name structures:relationshipMetadata, and that contains one or more references to metadata objects.

For each item *$metadata-reference* in *$metadata-list*, the following RDF quad is entailed:

$context-node-id $predicate-id $literal $metadata-node-id .

Where:

* *$context-node-id* is a node identifier for the object held by *$context-element*.
* *$predicate-id* is the resource IRI for *$predicate-attribute*.
* *$literal* is the literal value for *$attribute*, as described in [Section 5.6.2, *RDF Literals*, above](#section_5.6.2).
* *$metadata-node-id* is a node identifier for the object referenced by *$metadata-reference*.

#### 5.6.3.8. Elements and attributes via an augmentation type

Given:

* *$base* is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7) and is an instance of an [·object type·](#definition_object_type) or [·association type·](#definition_association_type).
* *$augmentation-element* is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7), is a child of *$base*, and is an instance of an [·augmentation type·](#definition_augmentation_type).
* *$augmentation-identifier* is a node identifier for the object help by *$augmentation-element*.

For each *$resolved-element* that is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7) holding an object with node identifier *$augmentation-identifier*:

Each element child of *$resolved-element* entails RDF as if it were an element child of *$base*.

Each attribute that has property [attribute owner] equal to *$resolved-element* entails RDF as if it were an attribute with property [attribute owner] equal to *$base*.

#### 5.6.3.9. Properties applied via structures:metadata

Given a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7) *$context* that has attribute structures:metadata with a value that is a list of references, each item *$item* in the list entails the RDF:

$subject structures:metadata $object .

Where:

* *$subject* is a node identifier for the object held by *$context*.
* *$object* is a node identifier for the object referenced by *$item*.

### 5.6.4. Type information for instance documents

A [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7) *$element* that is an instance of an [·object type·](#definition_object_type) or [·association type·](#definition_association_type) entails the following RDF:

$element-object rdf:type $element-type .

Where:

* *$element-object* is a node identifier for the object held by *$element*.
* *$element-type* is resource IRI for the value of the [type definition] property of *$element*.

### 5.6.5. NIEM schema component definitions to RDF

The definition of schema components within NIEM-conformant schemas a parallel representation in RDF. This section describes the mapping of selected XML Schema constructs to RDF.

This section has the following subsections:

* [Section 5.6.5.1, *NIEM complex type definitions to RDF*](#section_5.6.5.1)
* [Section 5.6.5.2, *NIEM element declaration mappings to RDF*](#section_5.6.5.2)
* [Section 5.6.5.3, *NIEM attribute declarations to RDF*](#section_5.6.5.3)

#### 5.6.5.1. NIEM complex type definitions to RDF

The following RDF mappings apply to the content of a [·reference schema document·](#definition_reference_schema_document) or [·extension schema document·](#definition_extension_schema_document).

An [·object type·](#definition_object_type) or [·association type·](#definition_association_type) $type entails the following RDF:

$resource rdf:type rdf:Class .

Where *$resource* is the resource IRI for *$type*.

An [·object type·](#definition_object_type) or [·association type·](#definition_association_type) *$type* that has property {base type definition} *$base* also entails the RDF:

$resource rdfs:subClassOf $base-resource .

Where *$base-resource* is the resource IRI for *$base*.

#### 5.6.5.2. NIEM element declaration mappings to RDF

The following RDF mappings apply to the content of a [·reference schema document·](#definition_reference_schema_document) or [·extension schema document·](#definition_extension_schema_document).

A top-level element declaration schema component *$element-declaration* that has property {type definition} that is

* the ur-type, or
* is, or is derived from, structures:ObjectType, or
* is, or is derived from, structures:AssociationType

entails the RDF:

$element-resource rdf:type rdf:Property .

Where *$element-resource* is the resource IRI for *$element-declaration*.

If *$element-declaration* has property {substitution group affiliation} with a value of element declaration *$base*, then it entails the RDF:

$element-resource rdfs:subPropertyOf $base-resource .

Where:

* *$element-resource* is the resource IRI for *$element-declaration*.
* *$base-resource* is the resource IRI for *$base*.

If *$element-declaration* has property {type definition} with a value *$type* that is an [·object type·](#definition_object_type) or [·association type·](#definition_association_type), then it entails the RDF:

$element-resource rdfs:range $type-resource .

Where:

* *$element-resource* is the resource IRI for *$element-declaration*.
* *$type-resource* is the resource IRI for *$type*.

#### 5.6.5.3. NIEM attribute declarations to RDF

The following RDF mappings apply to the content of a [·reference schema document·](#definition_reference_schema_document) or [·extension schema document·](#definition_extension_schema_document).

A top-level attribute declaration schema component *$attribute-declaration* that has property {type definition} that is a simple type definition defined within a [·reference schema document·](#definition_reference_schema_document) or an [·extension schema document·](#definition_extension_schema_document), then it entails the RDF:

$attribute-resource rdf:type rdf:Property .

Where *$attribute-resource* is the resource IRI for *$attribute-declaration*.

# 6. Guiding principles

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

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

The principles discussed in this section are organized as follows:

* [Section 6.1, *Specification guidelines*](#section_6.1)
* [Section 6.2, *XML Schema design guidelines*](#section_6.2)
* [Section 6.3, *Modeling design guidelines*](#section_6.3)
* [Section 6.4, *Implementation guidelines*](#section_6.4)
* [Section 6.5, *Modeling guidelines*](#section_6.5)

## 6.1. Specification guidelines

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

### 6.1.1. Keep specification to a minimum

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

[Principle 1]

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

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

### 6.1.2. Focus on rules for schemas

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

[Principle 2]

This specification SHOULD focus on specifying characteristics of schema documents, their instance documents, and their interpretation.

### 6.1.3. Use specific, concise rules

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

[Principle 3]

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

## 6.2. XML Schema design guidelines

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

### 6.2.1. Purpose of XML Schemas

[Principle 4]

The specification SHOULD focus on rules for XML Schemas in order to support:

1. the expression of interface specifications for exchange of information between data producers and consumers,
2. validation of XML documents against interface specifications,
3. comprehension and understanding of XML documents, including the use of post-schema-validation infoset as defined by [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.1, *Overview of XML Schema*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#key-psvi), and
4. development of implementations supported by tools that use XML Schemas.

### 6.2.2. Prohibit XML parsing from constructing values

XML Schema has features that can make the data provided by an XML Schema validating parser differ from that provided by a non-validating XML parser. For example, if an XML Schema attribute declaration has a default value, and if an XML document omits the attribute where it might appear, an XML Schema validating parser will *construct* the attribute with the default value in the infoset that it provides to its caller. Without schema validation, there would be no attribute value, but after processing, the attribute value exists in the parsed data provided to the caller. [Section 8.4, *Ensure XML parsing does not construct values*, below,](#section_8.4) provides more detail.

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

[Principle 5]

The data of a NIEM-conformant XML document provided by an XML parser SHOULD NOT be modified by the process of validating the data against an XML Schema.

### 6.2.3. Use XML validating parsers for content validation

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

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

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

[Principle 6]

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

### 6.2.4. Validate for conformance to schema

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

[Principle 7]

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

### 6.2.5. Allow multiple schemas for XML constraints

NIEM does not attempt to create a one-size-fits-all schema to perform all validation. Instead, it creates a set of reference schemas, on which additional constraints may be placed. Although NIEM is principally expressed as a set of reference schemas for a NIEM release, additional validation may be conducted through other mechanisms, which may include XML Schemas that express additional constraints, and business rules and structure-specifying languages other than the [·XML Schema definition language·](#X7e4888920c30d9bbda28173a5b8b96fa0c14a07).

[Principle 8]

Constraints on XML instances MAY be validated by multiple validation passes, using multiple schemas and specifications for different aspects of each namespace.

### 6.2.6. Define one reference schema per namespace

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

[Principle 9]

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

### 6.2.7. Disallow mixed content

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

[Principle 10]

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

### 6.2.8. Specify types for all constructs

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

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

[Principle 11]

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

### 6.2.9. Avoid wildcards in reference schemas

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

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

[Principle 12]

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

### 6.2.10. Schema locations provided in schema documents are hints

[**[XML Schema Structures]**](#XMLSchema-1) provides several mechanisms for acquiring components of an [·XML Schema·](#definition_XML_Schema) for the purpose of assessing validity of an instance. [**[XML Schema Structures]**](#XMLSchema-1) [Section 4.3.2, *How schema definitions are located on the Web*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#schema_reference) includes:

* Use schema definitions already known to the processor.
* Use a location URI or namespace name to identify a schema document from a network location or local schema repository.
* Attempt to resolve a location URI or namespace name to locate a schema document.

In addition, there are several ways for a processor to determine schema locations:

* Use schema locations identified by user direction.
* Use locations provided via xsi:schemaLocation or xsi:noNamespaceSchemaLocation attributes in an [·XML document·](#definition_XML_document) under assessment.
* Use schema locations provided by xs:import elements.

[**[XML Schema Structures]**](#XMLSchema-1) characterizes several of these methods as *hints* of where to acquire a schema document for assessment. [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.6.3, *xsi:schemaLocation, xsi:noNamespaceSchemaLocation*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#xsi_schemaLocation) states:

The xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes can be used in a document to provide hints as to the physical location of schema documents which may be used for assessment.

[**[XML Schema Structures]**](#XMLSchema-1) [Section 4.2.3, *References to schema components across namespaces*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#composition-schemaImport) states:

The actual value of the schemaLocation, if present, gives a hint as to where a serialization of a schema document with declarations and definitions for that namespace (or none) may be found.

The specification explicitly maintains that schema location provided in schemas or instances may be overridden by applications or by user direction.

[Principle 13]

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

In accordance with [Section 1.1, *Scope*, above](#section_1.1), this document does not provide normative guidance for location of [·schema documents·](#definition_schema_document) or for schema assembly.

### 6.2.11. Use open standards

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

[Principle 14]

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

## 6.3. Modeling design guidelines

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

### 6.3.1. Namespaces enhance reuse

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

[Principle 15]

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

NIEM relies heavily on XML namespaces to prevent naming conflicts and clashes. Reuse of any component is always by reference to both its namespace and its local name. All NIEM component names have global scope. An instance document or schema should use or reuse a NIEM component by referencing its NIEM-defined namespace and local name. An application that validates an instance document that contains a component in a NIEM namespace should conduct that validation using NIEM reference schema documents or profiles of NIEM reference schema documents, to ensure consistency across exchanges and implementations.

Example:

Figure 6-1: Example of the use of a namespace

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

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

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

[Principle 16]

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

### 6.3.2. Design NIEM for extensibility

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

[Principle 17]

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

## 6.4. Implementation guidelines

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

### 6.4.1. Avoid displaying raw XML data

XML data should be made human-understandable when possible, but it is not targeted at human consumers. HTML is intended for browsers. Browsers and similar technology provide human interfaces to XML and other structured content. Structured XML content does not belong in places targeting humans. Human-targeted information should be of a form suitable for presentation.

[Principle 18]

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

### 6.4.2. Leave implementation decisions to implementers

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

[Principle 19]

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

[Principle 20]

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

## 6.5. Modeling guidelines

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

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

### 6.5.1. Documentation

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

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

[Principle 21]

The [·data definition·](#definition_data_definition) of a component SHOULD be drafted before the component’s name is composed.

### 6.5.2. Consistent naming

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

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

[Principle 22]

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

### 6.5.3. Reflect the real world

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

[Principle 23]

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

Different users often have their own local practices for arranging components together, such as grouping components into sets and segments, or building in extra layers to help categorize components and make them easier to find when drilling down through an object. Since the broader community may not share the same practices, and grouping structures add additional complexity to both schemas and instances, NIEM components should model the real world as simply and closely as possible.

### 6.5.4. Be consistent

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

[Principle 24]

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

### 6.5.5. Reserve inheritance for specialization

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

[Principle 25]

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

Note that the use of [·augmentations·](#definition_augmentation) allows developers to avoid type specialization in many cases; see [Section 10.4, *Augmentations*, below](#section_10.4).

### 6.5.6. Do not duplicate definitions

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

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

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

[Principle 26]

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

### 6.5.7. Keep it simple

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

[Principle 27]

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

### 6.5.8. Be aware of scope

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

[Principle 28]

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

### 6.5.9. Be mindful of namespace cohesion

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

[Principle 29]

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

# 7. Conformance to standards

There are numerous XML standards to which the instance and schema documents that constitute information exchanges must conform. This section applies XML specifications to the conformance targets of this document.

## 7.1. Conformance to XML

The XML specification [**[XML]**](#XML) defines the term [·XML document·](#definition_XML_document). NIEM XML documents, including instance documents and schema documents, must conform to the definition of this term.

Rule 7-1. Document is an XML document

[Rule 7-1] (REF, EXT, INS) (Constraint)

The document MUST be an XML document.

This document establishes the term [·XML document·](#definition_XML_document), by reference to [**[XML]**](#XML).

## 7.2. Conformance to XML Namespaces

The XML namespaces specification defines correct use of XML namespaces; NIEM-conformant XML artifacts (instance documents and schema documents) must use namespaces properly.

Rule 7-2. Document uses XML namespaces properly

[Rule 7-2] (REF, EXT, INS) (Constraint)

The document MUST be namespace-well-formed and namespace-valid.

The terms *namespace-well-formed* and *namespace-valid* are normatively defined by [**[XML Namespaces]**](#XMLNamespaces).

## 7.3. Conformance to XML Schema

The XML Schema specification part 1 [**[XML Schema Structures]**](#XMLSchema-1) defines the syntax of the [·XML Schema definition language·](#X7e4888920c30d9bbda28173a5b8b96fa0c14a07), and identifies an [·XML document·](#definition_XML_document) that follows that syntax as a [·schema document·](#definition_schema_document). This section requires that NIEM reference and extension schema documents be [·schema documents·](#definition_schema_document).

Rule 7-3. Document is a schema document

[Rule 7-3] (REF, EXT) (Constraint)

The [XML document] MUST be a [schema document].

This document establishes the term [·schema document·](#definition_schema_document), by reference to [**[XML Schema Structures]**](#XMLSchema-1).

Rule 7-4. Document element is xs:schema

[Rule 7-4] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[. is nf:get-document-element(.)]">  
    <sch:assert test="self::xs:schema"  
      >The [document element] of the [XML document] MUST have the name xs:schema.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document establishes the terms [·document element·](#definition_document_element) by reference to [**[XML Infoset]**](#XMLInfoset), and [·XML document·](#definition_XML_document) by reference to [**[XML]**](#XML).

## 7.4. ISO 11179 Part 4

Good data definitions are fundamental to data interoperability. You cannot effectively exchange what you cannot understand. NIEM employs the guidance of [**[ISO 11179-4]**](#ISO11179-4) as a baseline for its data component definitions.

To advance the goal of creating semantically rich NIEM-conformant schemas, it is necessary that data definitions be descriptive, meaningful, and precise. [**[ISO 11179-4]**](#ISO11179-4) provides standard structure and rules for defining data definitions. NIEM uses this standard for component definitions.

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

For convenience and reference, the summary requirements and recommendations in [**[ISO 11179-4]**](#ISO11179-4) are reproduced here:

**ISO 11179 Requirements**

A data definition SHALL:

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

**ISO 11179 Recommendations**

A data definition SHOULD:

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

In addition to the requirements and recommendations of [**[ISO 11179-4]**](#ISO11179-4), NIEM applies additional rules to data definitions. These rules are detailed in [Section 11.6.1, *Human-readable documentation*, below](#section_11.6.1).

These definitions leverage the term “definition” as defined by [**[ISO 11179-4]**](#ISO11179-4):

representation of a concept by a descriptive statement which serves to differentiate it from related concepts

[Definition: data definition]

The **data definition** of a [·schema component·](#definition_schema_component) is 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 the element that defines the component.

[Definition: documented component]

In a NIEM-conformant schema, a **documented component** is [·schema component·](#definition_schema_component) that has an associated [·data definition·](#definition_data_definition). Each documented component has a textual definition, so that the component may be well-understood.

An example of a data definition is provided in [Figure 7-1, *Example of data definition of element nc:Activity*, below](#figure_7-1).

Figure 7-1: Example of data definition of element nc:Activity

<xs:element name="Activity" type="nc:ActivityType" nillable="true">  
  <xs:annotation>  
    <xs:documentation>A single or set of related actions, events, or process steps.</xs:documentation>  
  </xs:annotation>  
</xs:element>

See [Rule 11-28, *Data definition follows 11179-4 requirements*, below,](#rule_11-28) and [Rule 11-29, *Data definition follows 11179-4 recommendations*, below,](#rule_11-29) for application of [**[ISO 11179-4]**](#ISO11179-4) to constrain NIEM [·data definitions·](#definition_data_definition).

## 7.5. ISO 11179 Part 5

Names are a simple but incomplete means of providing semantics to data components. Data definitions, structure, and context help to fill the gap left by the limitations of naming. The goals for data component names should be syntactic consistency, semantic precision, and simplicity. In many cases, these goals conflict and it is sometimes necessary to compromise or to allow exceptions to ensure clarity and understanding. To the extent possible, NIEM applies [**[ISO 11179-5]**](#ISO11179-5) to construct NIEM data component names.

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

Rule 7-5. Component name follows ISO 11179 Part 5 Annex A

[Rule 7-5] (REF, EXT) (Constraint)

A NIEM component name SHOULD be formed by applying the informative guidelines and examples detailed in Annex A of [**[ISO 11179-5]**](#ISO11179-5), with exceptions as specified in this document.

The guidelines and examples of [**[ISO 11179-5]**](#ISO11179-5) provide a simple, consistent syntax for data names that captures context and thereby imparts a reasonable degree of semantic precision.

NIEM uses the guidelines and examples of [**[ISO 11179-5]**](#ISO11179-5) as a baseline for normative naming rules. However, some NIEM components require bending of these rules. Special naming rules for these classes of components are presented and discussed in the relevant sections. In spite of these exceptions, most NIEM component names can be disassembled into their [**[ISO 11179-5]**](#ISO11179-5) constituent words or terms.

Example:

The NIEM component name AircraftFuselageColorCode disassembles as follows:

* Object class term = “Aircraft”
* Qualifier term = “Fuselage”
* Property term = “Color”
* Representation term = “Code”

[Section 10.8, *Naming rules*, below,](#section_10.8) details the specific rules for each kind of term and how to construct NIEM component names from it.

## 7.6. IC-ISM and IC-NTK

The Office of the Director of National Intelligence manages and maintains a set of specifications that support the requirements of the Intelligence Community (IC) to share and manage data across the IC enterprise. These specifications are available at <http://purl.org/ic/standards/public>. The design of NIEM supports the use of NIEM-conformant data across the IC.

The following features have been provided to support the use of NIEM-conformant data definitions across the IC, by supporting the use of IC-ISM and IC-NTK within NIEM-defined data:

* NIEM base types defined by the [·structures namespace·](#definition_structures_namespace) incorporate xs:anyAttribute declarations, which allow the use of attributes from the ISM and NTK namespaces. See [Appendix B, *Structures namespace*, below,](#appendix_B) for the reference schema.
* [Rule 11-23, *Schema uses only known attribute groups*, below,](#rule_11-23) allows conformant data types to reference attribute groups defined in the ISM and NTK namespaces.
* Complex types may be constructed that use external attributes, including attributes from the ISM and NTK namespaces. Each such attribute use must be a [·documented component·](#definition_documented_component), as specified by [Rule 10-14, *External attribute use has data definition*, below](#rule_10-14).

These features ensure that payloads that correctly use IC-ISM AND IC-NTK are supported by NIEM-conformant schema definitions.

# 8. Strategy for a NIEM profile of XML Schema

## 8.1. Wildcards

There are many constructs within XML Schema that act as wildcards. That is, they introduce buckets that may carry arbitrary or otherwise not-validated content. Such constructs violate [Principle 12, above](#principle_12), and as such provide implicit workarounds for the difficult task of agreeing on the content of data models. Such workarounds should be made explicitly, outside the core data model.

The following restrictions help to ban wildcards and arbitrary data:

* Use of the type xs:anyType is prohibited.
* Use of the type xs:anySimpleType is prohibited in most cases.
* Use of the element xs:any is prohibited in reference schemas.
* Use of the element xs:anyAttribute is prohibited in reference schemas.

## 8.2. Components are globally reusable

Each component defined by a NIEM-conformant schema may be reused from outside the schema document. Every type definition, element declaration, or attribute declaration schema component that is defined by a NIEM-conformant schema is given an explicit name. These schema components are not defined as local or anonymous. These components are defined at the top level, as children of element xs:schema.

This is supported by the rules:

* [Rule 9-10, *Simple type definition is top-level*, below](#rule_9-10)
* [Rule 9-25, *Complex type definition is top-level*, below](#rule_9-25)
* [Rule 9-36, *Element declaration is top-level*, below](#rule_9-36)
* [Rule 9-48, *Attribute declaration is top-level*, below](#rule_9-48)

Additional restrictions ensure that NIEM components are also defined such that new components may be derived from them and substituted for them. Reference schemas are defined to maximize reuse, while extension schemas are defined to enable a developer to customize schema definitions to match her exchange needs. In reference schemas, the following restrictions help enforce reusability through extension and substitution:

* A [·reference schema document·](#definition_reference_schema_document) or [·extension schema document·](#definition_extension_schema_document) may not use blockDefault (per [Rule 9-86, *No disallowed substitutions*, below](#rule_9-86)) or finalDefault (per [Rule 9-87, *No disallowed derivations*, below](#rule_9-87)).
* Element declarations and type definitions in a [·reference schema document·](#definition_reference_schema_document) may not use block or final, per the rules:
* [Rule 9-34, *No complex type disallowed substitutions*, below](#rule_9-34)
* [Rule 9-35, *No complex type disallowed derivation*, below](#rule_9-35)
* [Rule 9-43, *No element disallowed substitutions* , below](#rule_9-43)
* [Rule 9-44, *No element disallowed derivation*, below](#rule_9-44)

## 8.3. Avoid recursive model groups

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

## 8.4. Ensure XML parsing does not construct values

An XML document expresses an infoset (see [**[XML Infoset]**](#XMLInfoset)); the infoset is the data carried by the XML document, and is expressed as a set of information items (e.g., element information items, attribute information items, etc.). [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.1, *Overview of XML Schema*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#d0e504) describes the process followed by an XML Schema validating parser. Beyond the actions of a plain XML parser, which provides the data from an XML document to its caller in a structured way, an XML Schema validating parser does the following:

1. Determining local schema-validity, that is whether an element or attribute information item satisfies the constraints embodied in the relevant components of an XML Schema;
2. Synthesizing an overall validation outcome for the item, combining local schema-validity with the results of schema-validity assessments of its descendants, if any, and adding appropriate augmentations to the infoset to record this outcome.

In short, not only does an XML Schema validating parser yield data from an XML document to its caller, it determines whether the XML document is valid against an XML Schema, and also provides an **augmented infoset** to the caller, constructed to reflect information implied by the schema, which may not appear in the original XML document.

XML Schema allows element and attribute declarations to provide default values. When an XML document does not contain a value for a component that has a default, the XML Schema validating parser will *construct* a value for the component. This is done through the use of the attributes default and fixed, both of which provide default values to attributes and element content. An XML Schema validating parser that validates an XML document against a schema that uses default or fixed may yield an infoset that is augmented, constructing values in the provided XML infoset where none existed in the original XML document.

NIEM schemas should not produce constructed values in the infoset. The process of XML Schema validation against NIEM schemas should provide for marking data as valid or invalid, but should not modify original infoset data with constructed values. The XML infoset yielded by a non-validating XML parser should be the same as that yielded by an XML Schema validating parser. Turning on schema validation should not alter the data received by the caller of the parser.

The prohibition of constructed values is supported by [Section 6.2.2, *Prohibit XML parsing from constructing values*, above](#section_6.2.2). It is also supported through a prohibition on all uses of default, and most uses of fixed on attributes and elements defined by NIEM-conformant schema documents.

## 8.5. Use namespaces rigorously

Every component defined by or used in a NIEM schema has a target namespace.

XML Schema requires that namespaces used in external references be imported using the xs:import element. The xs:import element appears as an immediate child of the xs:schema element. A schema must import any namespace which is not the local namespace, and which is referenced from the schema.

The behavior of import statements is not necessarily intuitive. In short, the import introduces a namespace into the schema document in which the import appears; it has no transitive effect. If the namespace introduced by an import statement are not referenced from that schema document, then the import statement has no effect.

Certain tools have been seen introducing transitive behavior on imports, which is not portable across XML Schema validating parsers. If namespace 1 imports namespace 2 which imports namespace 3, a reference from namespace 1 to namespace 3 is not legal; namespace 1 must explicitly import namespace 3. A tool that imports transitively may allow schema 1 to reference 3 without a direct import of namespace 3. This is prohibited by rules which require imports of namespaces of referenced components.

## 8.6. Documentation is for people; appinfo is for machines

The XML Schema specification defines two types of annotations: *user information* and *application information*. It defines that user information is for human consumption, while application information is for automatic processing.

[**[XML Schema Structures]**](#XMLSchema-1) [Section 3.13, *Annotations*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#cAnnotations) states:

Annotations provide for human- and machine-targeted annotations of schema components.

[**[XML Schema Structures]**](#XMLSchema-1) [Section 3.13.1, *The Annotation Schema Component*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#Annotation_details) states:

{user information} is intended for human consumption, {application information} for automatic processing.

The two types: human-targeted and machine-targeted, are kept separate by the use of two separate container elements defined by XML Schema: xs:documentation and xs:appinfo.

xs:documentation elements express “user information”. This information is targeted for reading by humans. The XML Schema specification does not say what form human-targeted information should take. Since it is intended for human consumption, xs:documentation elements in conformant schemas do not contain structured XML data. As such, any XML content appearing within a documentation element should be human-targeted examples, and should be escaped (e.g., using &lt; and &gt;, or using CDATA blocks). Documentation within conformant schemas should be plain text; whitespace formatting may not be preserved across processing.

xs:appinfo elements express “application information”. This is generally information supporting automatic processing of schemas. Application information is addressed in [Section 10.9, *Machine-readable annotations*, below](#section_10.9).

XML comments are not XML Schema constructs and are not specifically associated with any schema-based component; they are not considered semantically meaningful by NIEM and may not be retained through processing of NIEM schemas.

# 9. Rules for a NIEM profile of XML Schema

NIEM-conformant schemas use a profile of XML Schema. 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.

This section establishes a profile of XML Schema for NIEM-conformant schemas. Because the XML Schema specifications are flexible, comprehensive rules are needed to achieve a balance between establishing uniform schema design and providing developers flexibility to solve novel data modeling problems.

Note that external schema documents (i.e., non-NIEM-conformant schema documents) do not need to obey the rules set forth in this section. So long as schema components from external schema documents are adapted for use with NIEM according to the modeling rules in [Section 10.2.3, *External adapter types and external components*, below](#section_10.2.3), they may be used as they appear in the external standard, even if the schema components themselves violate the rules for NIEM-conformant schemas.

The following sections are broken down in the order provided by [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2, *XML Schema Abstract Data Model*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#concepts-data-model), followed by a section on a schema document as a whole and a section on schema namespaces and assembly:

* [Section 9.1, *Type definition components*](#section_9.1)
* [Section 9.2, *Declaration components*](#section_9.2)
* [Section 9.3, *Model group components*](#section_9.3)
* [Section 9.4, *Identity-constraint definition components*](#section_9.4)
* [Section 9.5, *Group definition components*](#section_9.5)
* [Section 9.6, *Annotation components*](#section_9.6)
* [Section 9.7, *Schema as a whole*](#section_9.7)
* [Section 9.8, *Schema assembly*](#section_9.8)

## 9.1. Type definition components

### 9.1.1. Type definition hierarchy

#### 9.1.1.1. Types prohibited as base types

Rule 9-1. No base type in the XML namespace

Although the XML namespace is to be imported as if it is conformant, types from that namespace may not be the [·base type definition·](#definition_base_type_definition) of any type.

[Rule 9-1] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@base)]">  
    <sch:assert test="namespace-uri-from-QName(resolve-QName(@base, .)) != xs:anyURI('http://www.w3.org/XML/1998/namespace')"  
                >A schema component must not have a base type definition with a {target namespace} that is the XML namespace.</sch:assert>  
  </sch:rule>  
</sch:pattern>

The term [·base type definition·](#definition_base_type_definition) has a normative definition.

Rule 9-2. No base type of xs:ID

[Rule 9-2] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@base)]">  
    <sch:assert test="resolve-QName(@base, .) != xs:QName('xs:ID')"  
      >A schema component MUST NOT have an attribute {}base with a value of xs:ID.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-3. No base type of xs:IDREF

[Rule 9-3] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@base)]">  
    <sch:assert test="resolve-QName(@base, .) != xs:QName('xs:IDREF')"  
      >A schema component MUST NOT have an attribute {}base with a value of xs:IDREF.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-4. No base type of xs:IDREFS

[Rule 9-4] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@base)]">  
    <sch:assert test="resolve-QName(@base, .) != xs:QName('xs:IDREFS')"  
      >A schema component MUST NOT have an attribute {}base with a value of xs:IDREFS.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-5. No base type of xs:anyType

[Rule 9-5] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@base)]">  
    <sch:assert test="resolve-QName(@base, .) != xs:QName('xs:anyType')"  
      >A schema component MUST NOT have an attribute {}base with a value of xs:anyType.</sch:assert>  
  </sch:rule>  
</sch:pattern>

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

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

Rule 9-6. No base type of xs:anySimpleType

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

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@base)]">  
    <sch:assert test="resolve-QName(@base, .) != xs:QName('xs:anySimpleType')"  
      >A schema component MUST NOT have an attribute {}base with a value of xs:anySimpleType.</sch:assert>  
  </sch:rule>  
</sch:pattern>

XML Schema provides a restriction of the “ur-type” that contains only simple content. This provides a wildcard for arbitrary text. It is realized in XML Schema as xs:anySimpleType.

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

Rule 9-7. No base type of xs:NOTATION

[Rule 9-7] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@base)]">  
    <sch:assert test="resolve-QName(@base, .) != xs:QName('xs:NOTATION')"  
      >A schema component MUST NOT have an attribute {}base with a value of xs:NOTATION.</sch:assert>  
  </sch:rule>  
</sch:pattern>

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

Rule 9-8. No base type of xs:ENTITY

[Rule 9-8] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@base)]">  
    <sch:assert test="resolve-QName(@base, .) != xs:QName('xs:ENTITY')"  
      >A schema component MUST NOT have an attribute {}base with a value of xs:ENTITY.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-9. No base type of xs:ENTITIES

[Rule 9-9] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@base)]">  
    <sch:assert test="resolve-QName(@base, .) != xs:QName('xs:ENTITIES')"  
      >A schema component MUST NOT have an attribute {}base with a value of xs:ENTITIES.</sch:assert>  
  </sch:rule>  
</sch:pattern>

### 9.1.2. Simple type definition

Rule 9-10. Simple type definition is top-level

[Rule 9-10] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:simpleType">  
    <sch:assert test="exists(parent::xs:schema)"  
      >A simple type definition MUST be top-level.</sch:assert>  
  </sch:rule>  
</sch:pattern>

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

Rule 9-11. No simple type disallowed derivation

[Rule 9-11] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:simpleType">  
    <sch:assert test="empty(@final)"  
      >An element xs:simpleType MUST NOT have an attribute {}final.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-12. Simple type has data definition

[Rule 9-12] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:simpleType">  
    <sch:assert test="some $definition in xs:annotation/xs:documentation[1] satisfies  
                        string-length(normalize-space(string($definition))) &gt; 0"  
      >A simple type MUST have a data definition.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the term [·data definition·](#definition_data_definition).

Rule 9-13. No use of “fixed” on simple type facets

An attribute fixed on a constraining facet (e.g., xs:maxInclusive) of a simple type *$base* prevents a simple type derived from *$base* from further restricting that facet. For example, if simpleType nc:LatitudeDegreeSimpleType uses an xs:maxInclusive facet that limits the maximum value to 90, a simple type derived from that type could not further restrict the type to limit the maximum value to 45.

The use of fixed on simple type facets violates [Principle 17, above](#principle_17), since it prevents an extension schema from constraining a base type. As a result, the fixed on simple type facets in reference schemas is prohibited.

[Rule 9-13] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[self::xs:length or self::xs:minLength or self::xs:maxLength or self::xs:whiteSpace  
      or self::xs:maxInclusive or self::xs:maxExclusive or self::xs:minExclusive or self::xs:minInclusive  
      or self::xs:totalDigits or self::xs:fractionDigits]">  
    <sch:assert test="empty(@fixed)"  
      >A simple type constraining facet MUST NOT have an attribute {}fixed.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-14. Enumeration has data definition

[Rule 9-14] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:enumeration">  
    <sch:assert test="some $definition in xs:annotation/xs:documentation[1] satisfies  
                        string-length(normalize-space(string($definition))) &gt; 0"  
      >An enumeration facet MUST have a data definition.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the term [·data definition·](#definition_data_definition).

#### 9.1.2.1. Simple types prohibited as list item types

There is no explicit prohibition on the use of xs:NOTATION as list item type, because it is prohibited by [**[XML Schema Datatypes]**](#XMLSchema-2).

There is no prohibition on xs:anyType as a list item type, because xs:anyType is not a simple type.

Rule 9-15. No list item type of xs:ID

[Rule 9-15] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@itemType)]">  
    <sch:assert test="resolve-QName(@itemType, .) != xs:QName('xs:ID')"  
      >A schema component MUST NOT have an attribute {}itemType with a value of xs:ID.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-16. No list item type of xs:IDREF

[Rule 9-16] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@itemType)]">  
    <sch:assert test="resolve-QName(@itemType, .) != xs:QName('xs:IDREF')"  
      >A schema component MUST NOT have an attribute {}itemType with a value of xs:IDREF.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-17. No list item type of xs:anySimpleType

[Rule 9-17] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@itemType)]">  
    <sch:assert test="resolve-QName(@itemType, .) != xs:QName('xs:anySimpleType')"  
      >A schema component MUST NOT have an attribute {}itemType with a value of xs:anySimpleType.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-18. No list item type of xs:ENTITY

[Rule 9-18] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@itemType)]">  
    <sch:assert test="resolve-QName(@itemType, .) != xs:QName('xs:ENTITY')"  
      >A schema component MUST NOT have an attribute {}itemType with a value of xs:ENTITY.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 9.1.2.2. Simple types prohibited as union member types

There is no explicit prohibition on the use of xs:NOTATION as a union member type, because it is prohibited by [**[XML Schema Datatypes]**](#XMLSchema-2).

There is no prohibition on xs:anyType as a union member type, because xs:anyType is not a simple type.

Rule 9-19. No union member types of xs:ID

[Rule 9-19] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@memberTypes)]">  
    <sch:assert test="every $type-qname  
                      in tokenize(normalize-space(@memberTypes), ' ')  
                      satisfies resolve-QName($type-qname, .) != xs:QName('xs:ID')"  
      >A schema component MUST NOT have an attribute {}memberTypes that includes a value of xs:ID.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-20. No union member types of xs:IDREF

[Rule 9-20] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@memberTypes)]">  
    <sch:assert test="every $type-qname  
                      in tokenize(normalize-space(@memberTypes), ' ')  
                      satisfies resolve-QName($type-qname, .) != xs:QName('xs:IDREF')"  
      >A schema component MUST NOT have an attribute {}memberTypes that includes a value of xs:IDREF.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-21. No union member types of xs:IDREFS

[Rule 9-21] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@memberTypes)]">  
    <sch:assert test="every $type-qname  
                      in tokenize(normalize-space(@memberTypes), ' ')  
                      satisfies resolve-QName($type-qname, .) != xs:QName('xs:IDREFS')"  
      >A schema component MUST NOT have an attribute {}memberTypes that includes a value of xs:IDREFS.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-22. No union member types of xs:anySimpleType

[Rule 9-22] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@memberTypes)]">  
    <sch:assert test="every $type-qname  
                      in tokenize(normalize-space(@memberTypes), ' ')  
                      satisfies resolve-QName($type-qname, .) != xs:QName('xs:anySimpleType')"  
      >A schema component MUST NOT have an attribute {}memberTypes that includes a value of xs:anySimpleType.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-23. No union member types of xs:ENTITY

[Rule 9-23] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@memberTypes)]">  
    <sch:assert test="every $type-qname  
                      in tokenize(normalize-space(@memberTypes), ' ')  
                      satisfies resolve-QName($type-qname, .) != xs:QName('xs:ENTITY')"  
      >A schema component MUST NOT have an attribute {}memberTypes that includes a value of xs:ENTITY.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-24. No union member types of xs:ENTITIES

[Rule 9-24] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@memberTypes)]">  
    <sch:assert test="every $type-qname  
                      in tokenize(normalize-space(@memberTypes), ' ')  
                      satisfies resolve-QName($type-qname, .) != xs:QName('xs:ENTITIES')"  
      >A schema component MUST NOT have an attribute {}memberTypes that includes a value of xs:ENTITIES.</sch:assert>  
  </sch:rule>  
</sch:pattern>

### 9.1.3. Complex type definition

Rule 9-25. Complex type definition is top-level

[Rule 9-25] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType">  
    <sch:assert test="exists(parent::xs:schema)"  
      >A complex type definition MUST be top-level.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Note that this implies that every xs:complexType element has a name attribute.

Rule 9-26. Complex type has data definition

[Rule 9-26] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType">  
    <sch:assert test="some $definition in xs:annotation/xs:documentation[1] satisfies  
                        string-length(normalize-space(string($definition))) &gt; 0"  
      >A complex type MUST have a data definition.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the term [·data definition·](#definition_data_definition).

#### 9.1.3.1. No mixed content

Rule 9-27. No mixed content on complex type

[Rule 9-27] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType[exists(@mixed)]">  
    <sch:assert test="xs:boolean(@mixed) = false()"  
      >A complex type definition MUST NOT have mixed content.</sch:assert>  
  </sch:rule>  
</sch:pattern>

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

Well-defined markup languages exist outside NIEM and may be used with NIEM data. External schema documents may include mixed content and may be used with NIEM. However, mixed content must not be defined by NIEM-conformant schemas in keeping with [Principle 10, above](#principle_10).

Rule 9-28. No mixed content on complex content

[Rule 9-28] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexContent[exists(@mixed)]">  
    <sch:assert test="xs:boolean(@mixed) = false()"  
      >A complex type definition with complex content MUST NOT have mixed content.</sch:assert>  
  </sch:rule>  
</sch:pattern>

See [Rule 9-27, *No mixed content on complex type*, above,](#rule_9-27) for the rationale for this rule.

Rule 9-29. Complex type content is explicitly simple or complex

[Rule 9-29] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType">  
    <sch:assert test="exists(xs:simpleContent) or exists(xs:complexContent)"  
      >An element xs:complexType MUST have a child element xs:simpleContent or xs:complexContent.</sch:assert>  
  </sch:rule>  
</sch:pattern>

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.

#### 9.1.3.2. Complex content

Rule 9-30. Complex content uses extension

[Rule 9-30] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexContent">  
    <sch:assert test="exists(xs:extension)"  
      >An element xs:complexContent MUST have a child xs:extension.</sch:assert>  
  </sch:rule>  
</sch:pattern>

NIEM does not support the use of complex type restriction in reference schemas. The use of restriction in a reference schema would reduce the ability for that schema to be reused. Restriction may be used in extension schemas.

##### 9.1.3.2.1. Base type of complex type with complex content has complex content

These two rules addresses a peculiarity of the [·XML Schema definition language·](#X7e4888920c30d9bbda28173a5b8b96fa0c14a07), 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. An example is included in the following figure. Note that type niem-xs:integer is a complex type with simple content.

Figure 9-1: Example of complex type with simple content that claims to have complex content

<xs:complexType name="TreeCountType">  
  <xs:annotation>  
    <xs:documentation>A data type for a number of trees</xs:documentation>  
  </xs:annotation>  
  <xs:complexContent>  
    <xs:extension base="niem-xs:integer"/>  
  </xs:complexContent>  
</xs:complexType>

The first rule handles cases that can be determined within a single schema document, while the SET version handles cases that require cross-schema resolution.

Rule 9-31. Base type of complex type with complex content must have complex content

[Rule 9-31] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType/xs:complexContent/xs:\*[  
                       (self::xs:extension or self::xs:restriction)  
                       and (some $base-qname in resolve-QName(@base, .) satisfies  
                              namespace-uri-from-QName($base-qname) = nf:get-target-namespace(.))]">  
    <sch:assert test="some $base-type in nf:resolve-type(., resolve-QName(@base, .)) satisfies  
                        empty($base-type/self::xs:complexType/xs:simpleContent)"  
      >The base type of complex type that has complex content MUST be a complex type with complex content.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-32. Base type of complex type with complex content must have complex content

[Rule 9-32] (SET) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType[  
        nf:has-effective-conformance-target-identifier(., xs:anyURI('http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ReferenceSchemaDocument'))  
        or nf:has-effective-conformance-target-identifier(., xs:anyURI('http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ExtensionSchemaDocument'))  
      ]/xs:complexContent">  
    <sch:assert test="some $derivation in xs:\*[self::xs:extension or self::xs:restriction],  
                           $base-qname in resolve-QName($derivation/@base, $derivation),  
                           $base-type in nf:resolve-type($derivation, $base-qname) satisfies  
                         empty($base-type/self::xs:complexType/xs:simpleContent)"  
      >The base type of complex type that has complex content MUST have complex content.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 9.1.3.3. Simple content

Rule 9-33. Simple content uses extension

[Rule 9-33] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:simpleContent">  
    <sch:assert test="exists(xs:extension)"  
      >A complex type definition with simple content schema component MUST have a derivation method of extension.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This rule ensures that the definition of a complex type with simple content will use XML Schema extension. Under this rule, the structure of these types will be more uniform, as alternate formats are prohibited. The above rule allows for use of xs:restriction within xs:simpleContent in extension schemas.

Rule 9-34. No complex type disallowed substitutions

[Rule 9-34] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType">  
    <sch:assert test="empty(@block)"  
      >An element xs:complexType MUST NOT have an attribute {}block.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-35. No complex type disallowed derivation

[Rule 9-35] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType">  
    <sch:assert test="empty(@final)"  
      >An element xs:complexType MUST NOT have an attribute {}final.</sch:assert>  
  </sch:rule>  
</sch:pattern>

## 9.2. Declaration components

### 9.2.1. Element declaration

Every element declaration in a NIEM-conformant schema document is a top-level element; rules prohibit the declaration of local elements.

Within a NIEM-conformant schema document, an element may be declared as abstract. Elements may be defined without a type, but any element declaration that has no type must be declared abstract, as specified by [Rule 9-38, *Untyped element is abstract*, below](#rule_9-38).

Within an element declaration, the attributes fixed, nillable, and substitutionGroup may be used as per the XML Schema specification. The attribute form is irrelevant to NIEM, as NIEM-conformant schemas may not contain local element declarations, as specified by [Rule 9-36, *Element declaration is top-level*, below](#rule_9-36).

Rule 9-36. Element declaration is top-level

[Rule 9-36] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@name)]">  
    <sch:assert test="exists(parent::xs:schema)"  
      >An element declaration MUST be top-level.</sch:assert>  
  </sch:rule>  
</sch:pattern>

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

Rule 9-37. Element declaration has data definition

[Rule 9-37] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@name)]">  
    <sch:assert test="some $definition in xs:annotation/xs:documentation[1] satisfies  
                        string-length(normalize-space(string($definition))) &gt; 0"  
      >An element declaration MUST have a data definition.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the term [·data definition·](#definition_data_definition).

Rule 9-38. Untyped element is abstract

[Rule 9-38] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:schema/xs:element[empty(@type)]">  
    <sch:assert test="exists(@abstract)  
                      and xs:boolean(@abstract) = true()"  
      >A top-level element declaration that does not set the {type definition} property via the attribute "type" MUST have the {abstract} property with a value of "true".</sch:assert>  
  </sch:rule>  
</sch:pattern>

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 9-39. Element of type xs:anySimpleType is abstract

The type xs:anySimpleType does not have any concrete semantics; The use of xs:anySimpleType is limited to the case where an abstract element is of type xs:anySimpleType, to act as a base for concrete implementations of the element.

[Rule 9-39] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@type)  
                                and resolve-QName(@type, .) = xs:QName('xs:anySimpleType')]">  
    <sch:assert test="exists(@abstract)  
                      and xs:boolean(@abstract) = true()"  
                >An element declaration that has a type xs:anySimpleType MUST have the {abstract} property with a value of "true".</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-40. Element type not in the XML Schema namespace

[Rule 9-40] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@type)]">  
    <sch:assert test="for $type-qname in resolve-QName(@type, .) return  
                        $type-qname = xs:QName('xs:anySimpleType')  
                        or namespace-uri-from-QName($type-qname) != xs:anyURI('http://www.w3.org/2001/XMLSchema')"  
      >An element type that is not xs:anySimpleType MUST NOT have a namespace name http://www.w3.org/2001/XMLSchema.</sch:assert>  
  </sch:rule>  
</sch:pattern>

The prohibition of element types having the XML Schema namespace subsumes a prohibition of the type xs:anyType.

Rule 9-41. Element type not in the XML namespace

The XML namespace may be imported into a conformant schema document as if it were conformant. This specification does not enable a reference to any types that may be defined by any implementation of a schema for that namespace.

[Rule 9-41] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@type)]">  
    <sch:assert test="namespace-uri-from-QName(resolve-QName(@type, .)) != 'http://www.w3.org/XML/1998/namespace'"  
      >An element type MUST NOT have a namespace name that is in the XML namespace.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-42. Element type is not simple type

[Rule 9-42] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[@type]">  
    <sch:assert test="every $type-qname in resolve-QName(@type, .),  
                            $type-ns in namespace-uri-from-QName($type-qname),  
                            $type-local-name in local-name-from-QName($type-qname) satisfies (  
                        $type-qname = xs:QName('xs:anySimpleType')  
                        or (($type-ns = nf:get-target-namespace(.)  
                             or exists(nf:get-document-element(.)/xs:import[  
                                         xs:anyURI(@namespace) = $type-ns  
                                         and empty(@appinfo:externalImportIndicator)]))  
                            and not(ends-with($type-local-name, 'SimpleType'))))"  
      >An element type that is not xs:anySimpleType MUST NOT be a simple type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-43. No element disallowed substitutions

[Rule 9-43] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element">  
    <sch:assert test="empty(@block)"  
      >An element xs:element MUST NOT have an attribute {}block.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-44. No element disallowed derivation

[Rule 9-44] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element">  
    <sch:assert test="empty(@final)"  
      >An element xs:element MUST NOT have an attribute {}final.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 9.2.1.1. No element value constraints

Rule 9-45. No element default value

[Rule 9-45] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element">  
    <sch:assert test="empty(@default)"  
      >An element xs:element MUST NOT have an attribute {}default.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-46. No element fixed value

[Rule 9-46] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element">  
    <sch:assert test="empty(@fixed)"  
      >An element xs:element MUST NOT have an attribute {}fixed.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-47. Element declaration is nillable

All elements declared by reference schemas allow a nil value. This enables the ID/IDREF mechanism linking structures:ref and structures:id, as described by [Section 12.2, *Identifiers and references*, below](#section_12.2).

A developer may constrain the use of nil in an instance by setting nillable to false in subset schemas, or by use of non-XML Schema mechanisms, such as Schematron.

[Rule 9-47] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[@name and (empty(@abstract)  
                                           or xs:boolean(@abstract) = false())]">  
    <sch:assert test="exists(@nillable)  
                      and xs:boolean(@nillable) = true()"  
      >An element declaration MUST have the {nillable} property with a value of true.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the term [·element declaration·](#definition_element_declaration).

### 9.2.2. Element substitution group

This section is intentionally blank. It is present in order to ensure that section numbers correspond to [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2, *XML Schema Abstract Data Model*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#concepts-data-model), as described by [Section 9, *Rules for a NIEM profile of XML Schema*, above](#section_9).

### 9.2.3. Attribute declaration

Within an attribute declaration, the attribute form is irrelevant to NIEM, as NIEM-conformant schemas may not contain local attribute declarations.

Rule 9-48. Attribute declaration is top-level

[Rule 9-48] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@name)]">  
    <sch:assert test="exists(parent::xs:schema)"  
      >An attribute declaration MUST be top-level.</sch:assert>  
  </sch:rule>  
</sch:pattern>

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

Rule 9-49. Attribute declaration has data definition

[Rule 9-49] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@name)]">  
    <sch:assert test="some $definition in xs:annotation/xs:documentation[1] satisfies  
                        string-length(normalize-space(string($definition))) &gt; 0"  
      >An attribute declaration MUST have a data definition.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the term [·data definition·](#definition_data_definition).

Rule 9-50. Attribute declaration has type

[Rule 9-50] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@name)]">  
    <sch:assert test="exists(@type)"  
      >A top-level attribute declaration MUST have a type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

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

#### 9.2.3.1. Prohibited attribute types

There is no explicit prohibition on the use of xs:NOTATION as an attribute type, because it is prohibited by [**[XML Schema Datatypes]**](#XMLSchema-2).

These types are only explicitly prohibited from attributes, not elements, because the case is covered by a general prohibition against elements having simple types.

Rule 9-51. No attribute type of xs:ID

[Rule 9-51] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@type)]">  
    <sch:assert test="resolve-QName(@type, .) != xs:QName('xs:ID')"  
      >A schema component MUST NOT have an attribute {}type with a value of xs:ID.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-52. No attribute type of xs:IDREF

[Rule 9-52] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@type)]">  
    <sch:assert test="resolve-QName(@type, .) != xs:QName('xs:IDREF')"  
      >A schema component MUST NOT have an attribute {}type with a value of xs:IDREF.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-53. No attribute type of xs:IDREFS

[Rule 9-53] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@type)]">  
    <sch:assert test="resolve-QName(@type, .) != xs:QName('xs:IDREFS')"  
      >A schema component MUST NOT have an attribute {}type with a value of xs:IDREFS.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-54. No attribute type of xs:ENTITY

[Rule 9-54] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@type)]">  
    <sch:assert test="resolve-QName(@type, .) != xs:QName('xs:ENTITY')"  
      >A schema component MUST NOT have an attribute {}type with a value of xs:ENTITY.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-55. No attribute type of xs:ENTITIES

[Rule 9-55] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@type)]">  
    <sch:assert test="resolve-QName(@type, .) != xs:QName('xs:ENTITIES')"  
      >A schema component MUST NOT have an attribute {}type with a value of xs:ENTITIES.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-56. No attribute type of xs:anySimpleType

[Rule 9-56] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@type)]">  
    <sch:assert test="resolve-QName(@type, .) != xs:QName('xs:anySimpleType')"  
      >A schema component MUST NOT have an attribute {}type with a value of xs:anySimpleType.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 9.2.3.2. No attribute value constraints

Rule 9-57. No attribute default values

[Rule 9-57] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute">  
    <sch:assert test="empty(@default)"  
      >An element xs:attribute MUST NOT have an attribute {}default.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This rule helps to ensure that schema processing does not alter processed data, as described in [Section 8.4, *Ensure XML parsing does not construct values*, above](#section_8.4).

Rule 9-58. No fixed values for optional attributes

The fixed attribute is described by [**[XML Schema Structures]**](#XMLSchema-1) [Section 3.2.1, *The Attribute Declaration Schema Component*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#Attribute_Declaration_details):

*default* specifies that the attribute is to appear unconditionally in the post-schema-validation infoset, with the supplied value used whenever the attribute is not actually present; *fixed* indicates that the attribute value if present must equal the supplied constraint value, and if absent receives the supplied value as for *default*.

[Rule 9-58] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@ref) and @use eq 'required']">  
    <sch:report test="false()" role="warning">This rule does not constrain attribute uses that are required</sch:report>  
  </sch:rule>  
  <sch:rule context="xs:attribute">  
    <sch:assert test="empty(@fixed)"  
      >An element xs:attribute that is not a required attribute use MUST NOT have an attribute {}fixed.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This rule helps to ensure that schema processing does not alter processed data, as described in [Section 8.4, *Ensure XML parsing does not construct values*, above](#section_8.4). The use of the fixed attribute may result in alteration of the post-schema-validation infoset, like the use of default does. This behavior is described by [**[XML Schema Structures]**](#XMLSchema-1) [Section 3.2.1, *The Attribute Declaration Schema Component*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#Attribute_Declaration_details).

### 9.2.4. Notation declaration

Rule 9-59. No use of element xs:notation

[Rule 9-59] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:notation">  
    <sch:assert test="false()"  
      >The schema MUST NOT contain the element xs:notation.</sch:assert>  
  </sch:rule>  
</sch:pattern>

See [Rule 9-7, *No base type of xs:NOTATION*, above,](#rule_9-7) for comments about the use of notations.

## 9.3. Model group components

### 9.3.1. Model group

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

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

Rule 9-60. Model group does not affect meaning

[Rule 9-60] (EXT) (Interpretation)

The use of model groups xs:all, xs:sequence, and xs:choice MUST NOT define the semantics of an instance. The meaning of an element occurrence within an element occurrence MUST be identical, regardless of the model group used to define a [·schema component·](#definition_schema_component).

Rule 9-61. No xs:all

[Rule 9-61] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:all">  
    <sch:assert test="false()"  
      >The schema MUST NOT contain the element xs:all</sch:assert>  
  </sch:rule>  
</sch:pattern>

The element xs:all provides a set of particles (e.g., elements) that may be included in an instance, in no particular order. This compositor does not support a variety of cardinalities, has shown to be confusing in practice, and has functionality that overlaps with NIEM’s use of substitution groups. Use of substitution groups is the preferred method in NIEM schemas for obtaining flexible content models.

#### 9.3.1.1. Sequence

Rule 9-62. xs:sequence must be child of xs:extension

[Rule 9-62] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:sequence">  
    <sch:assert test="exists(parent::xs:extension)"  
      >An element xs:sequence MUST be a child of element xs:extension.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-63. xs:sequence must be child of xs:extension or xs:restriction

[Rule 9-63] (EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:sequence">  
    <sch:assert test="exists(parent::xs:extension) or exists(parent::xs:restriction)"  
      >An element xs:sequence MUST be a child of element xs:extension or xs:restriction.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 9.3.1.2. Choice

Rule 9-64. No xs:choice

[Rule 9-64] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:choice">  
    <sch:assert test="false()"  
      >The schema MUST NOT contain the element xs:choice</sch:assert>  
  </sch:rule>  
</sch:pattern>

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

The element xs:choice may be used in extension schemas, as it presents a simple way for a schema writer to represent a set of optional content.

Rule 9-65. xs:choice must be child of xs:sequence

[Rule 9-65] (EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:choice">  
    <sch:assert test="exists(parent::xs:sequence)"  
      >An element xs:choice MUST be a child of element xs:sequence.</sch:assert>  
  </sch:rule>  
</sch:pattern>

### 9.3.2. Particle

In NIEM schemas, an element use is an element declaration acting as a particle within a content model. Each such element declaration must reference a top-level named element declaration.

Element declarations acting as a particle (particles formed by xs:element) may have any cardinality. NIEM does not provide additional constraints on the values of the XML Schema properties {min occurs} and {max occurs} on element uses. These properties are described by [**[XML Schema Structures]**](#XMLSchema-1) [Section 3.9.1, *The Particle Schema Component*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#Particle_details).

#### 9.3.2.1. Sequence cardinality

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

A schema developer who requires the instance syntax that would be obtained from the use of specific cardinality on sequences should define cardinality on individual element uses.

Rule 9-66. Sequence has minimum cardinality 1

[Rule 9-66] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:sequence">  
    <sch:assert test="empty(@minOccurs) or xs:integer(@minOccurs) = 1"  
      >An element xs:sequence MUST either not have the attribute {}minOccurs, or that attribute MUST have a value of 1.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-67. Sequence has maximum cardinality 1

[Rule 9-67] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:sequence">  
    <sch:assert test="empty(@maxOccurs) or (@maxOccurs instance of xs:integer  
                                            and 1 = xs:integer(@maxOccurs))"  
      >An element xs:sequence MUST either not have the attribute {}maxOccurs, or that attribute MUST have a value of 1.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 9.3.2.2. Choice cardinality

Rule 9-68. Choice has minimum cardinality 1

[Rule 9-68] (EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:choice">  
    <sch:assert test="empty(@minOccurs) or 1 = xs:integer(@minOccurs)"  
      >An element xs:choice MUST either not have the attribute {}minOccurs, or that attribute MUST have a value of 1.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-69. Choice has maximum cardinality 1

[Rule 9-69] (EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:choice">  
    <sch:assert test="empty(@maxOccurs) or (@maxOccurs instance of xs:integer  
                                            and 1 = xs:integer(@maxOccurs))"  
      >An element xs:choice MUST either not have the attribute {}maxOccurs, or that attribute MUST have a value of 1.</sch:assert>  
  </sch:rule>  
</sch:pattern>

### 9.3.3. Attribute use

An attribute use has an {attribute declaration} property that is a top-level, named attribute declaration. NIEM-conformant schemas do not define local named attributes within type definitions. Within an attribute use, the attribute use may be used as per the XML Schema specification.

### 9.3.4. Wildcard

Rule 9-70. No use of xs:any

[Rule 9-70] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:any">  
    <sch:assert test="false()"  
      >The schema MUST NOT contain the element xs:any.</sch:assert>  
  </sch:rule>  
</sch:pattern>

The xs:any particle (see Model Group Restrictions for an informative definition of particle) provides a wildcard that may carry arbitrary content. The particle xs:any may appear within an [·extension schema document·](#definition_extension_schema_document).

Rule 9-71. No use of xs:anyAttribute

[Rule 9-71] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:anyAttribute">  
    <sch:assert test="false()"  
      >The schema MUST NOT contain the element xs:anyAttribute.</sch:assert>  
  </sch:rule>  
</sch:pattern>

The xs:anyAttribute element provides a wildcard, where arbitrary attributes may appear.

## 9.4. Identity-constraint definition components

XML Schema provides NIEM with the ability to apply uniqueness constraints to schema-validated content. These mechanisms, however, establish relationships in a way that is very difficult to understand, extend, and keep consistent through schema reuse.

Note that there is no prohibition on the use of xs:selector and xs:field, since according to the rules of the [·XML Schema definition language·](#X7e4888920c30d9bbda28173a5b8b96fa0c14a07), they can only appear within xs:key, xs:keyref, and xs:unique, which are themselves prohibited.

Rule 9-72. No use of xs:unique

[Rule 9-72] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:unique">  
    <sch:assert test="false()"  
      >The schema MUST NOT contain the element xs:unique.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-73. No use of xs:key

[Rule 9-73] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:key">  
    <sch:assert test="false()"  
      >The schema MUST NOT contain the element xs:key.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-74. No use of xs:keyref

[Rule 9-74] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:keyref">  
    <sch:assert test="false()"  
      >The schema MUST NOT contain the element xs:keyref.</sch:assert>  
  </sch:rule>  
</sch:pattern>

## 9.5. Group definition components

### 9.5.1. Model group definition

Rule 9-75. No use of xs:group

[Rule 9-75] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:group">  
    <sch:assert test="false()"  
      >The schema MUST NOT contain the element xs:group.</sch:assert>  
  </sch:rule>  
</sch:pattern>

NIEM does not allow groups of elements to be named other than as named complex types. A group in XML Schema creates a named entity that may be included in multiple types, and which consists of a sequence of or choice between element particles. NIEM has not developed a semantic model for these components, and they are not integrated into NIEM’s design.

### 9.5.2. Attribute group definition

Rule 9-76. No definition of attribute groups

Per [Rule 11-23, *Schema uses only known attribute groups*, below](#rule_11-23), the only attribute groups used in NIEM-conformant schemas are structures:SimpleObjectAttributeGroup and attribute groups defined by the IC-ISM and IC-NTK schemas. Therefore, NIEM-conformant schemas do not define additional attribute groups.

[Rule 9-76] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attributeGroup[@name]">  
    <sch:assert test="false()"  
      >The schema MUST NOT contain an attribute group definition schema component.</sch:assert>  
  </sch:rule>  
</sch:pattern>

## 9.6. Annotation components

Rule 9-77. Comment is not recommended

Note that this rule is written with a context that is not as precise as it could be. Its context is the parent node of the comment node, in order to avoid error messages when executed with common Schematron implementations.

[Rule 9-77] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="node()[comment()]">  
    <sch:report test="true()" role="warning"  
      >An XML Comment is not an XML Schema annotation component; an XML comment SHOULD NOT appear in the schema.</sch:report>  
  </sch:rule>  
</sch:pattern>

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.

Rule 9-78. Documentation element has no element children

[Rule 9-78] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:documentation/node()">  
    <sch:assert test="self::text() or self::comment()"  
      >A child of element xs:documentation MUST be text or an XML comment.</sch:assert>  
  </sch:rule>  
</sch:pattern>

### 9.6.1. Application information annotation

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

Rule 9-79. xs:appinfo children are comments, elements, or whitespace

[Rule 9-79] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:appinfo/node()">  
    <sch:assert test="self::comment()  
                      or self::element()  
                      or self::text()[string-length(normalize-space(.)) = 0]"  
      >A child of element xs:appinfo MUST be an element, a comment, or whitespace text.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Application information elements are intended for automatic processing; the meaning of an appinfo annotation is provided via elements.

Rule 9-80. Appinfo child elements have namespaces

[Rule 9-80] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:appinfo/\*">  
    <sch:assert test="namespace-uri() != xs:anyURI('')"  
      >An element that is a child of xs:appinfo MUST have a namespace name.</sch:assert>  
  </sch:rule>  
</sch:pattern>

The XML namespaces specification includes the concept of content not in a namespace. Use of elements without namespaces can lead to conflicting data definitions, and makes it difficult to identify relevant data definitions.

Rule 9-81. Appinfo descendants are not XML Schema elements

[Rule 9-81] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:appinfo//xs:\*">  
    <sch:assert test="false()"  
      >An element with a namespace name of xs: MUST NOT have an ancestor element xs:appinfo.</sch:assert>  
  </sch:rule>  
</sch:pattern>

NIEM-conformant schemas are designed to be very easily processed. Although uses of XML Schema elements as content of xs:appinfo elements could be contrived, it is not current practice and complicates the processing of XML elements by their namespaces and names. Forbidding the use of XML Schema elements outside valid uses of schema simplifies such processing.

## 9.7. Schema as a whole

The XML Schema language defines that the document element xs:schema may contain the optional attributes attributeFormDefault and elementFormDefault. The values of these attributes are not material to a conformant schema, as each attribute and element defined by a conformant schema is defined as a top-level component, and so each is qualified by its target namespace.

Rule 9-82. Schema has data definition

[Rule 9-82] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:schema">  
    <sch:assert test="some $definition in (xs:annotation/xs:documentation)[1] satisfies  
                        string-length(normalize-space(string($definition))) &gt; 0"  
      >An element xs:schema MUST have a data definition.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-83. Schema document defines target namespace

[Rule 9-83] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:schema">  
    <sch:assert test="exists(@targetNamespace)"  
      >The schema MUST define a target namespace.</sch:assert>  
  </sch:rule>  
</sch:pattern>

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

Rule 9-84. Target namespace is absolute URI

[Rule 9-84] (REF, EXT) (Constraint)

The value of the attribute targetNamespace MUST match the production <absolute-URI> as defined by RFC 3986.

Absolute URIs are the only universally meaningful URIs. URIs include both URLs and URNs. Finding the target namespace using standard XML Base technology is complicated and not specified by XML Schema. Relative URIs are not universally identifiable, as they are context-specific. \<absolute-URI> is a grammar production defined by [**[RFC 3986]**](#RFC3986) [Section 4.3, *Absolute URI*](http://tools.ietf.org/html/rfc3986#section-4.3).

Rule 9-85. Schema has version

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

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

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

An author of an application schema or exchange may use the version attribute for these purposes within their schemas.

[Rule 9-85] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:schema">  
    <sch:assert test="some $version in @version satisfies  
                      string-length(normalize-space(@version)) &gt; 0"  
        >An element xs:schema MUST have an attribute {}version that MUST NOT be empty.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-86. No disallowed substitutions

[Rule 9-86] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:schema">  
    <sch:assert test="empty(@blockDefault)"  
      >An element xs:schema MUST NOT have an attribute {}blockDefault.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-87. No disallowed derivations

[Rule 9-87] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:schema">  
    <sch:assert test="empty(@finalDefault)"  
      >An element xs:schema MUST NOT have an attribute {}finalDefault.</sch:assert>  
  </sch:rule>  
</sch:pattern>

## 9.8. Schema assembly

Rule 9-88. No use of xs:redefine

[Rule 9-88] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:redefine">  
    <sch:assert test="false()"  
      >The schema MUST NOT contain the element xs:redefine.</sch:assert>  
  </sch:rule>  
</sch:pattern>

The xs:redefine element allows an XML Schema document to restrict and extend components from a namespace, in a separate schema document from the one that initially defined that namespace. Such redefinition introduces duplication of definitions, allowing multiple definitions to exist for components from a single namespace. This violates [Principle 9, above](#principle_9), that a single reference schema defines a NIEM-conformant namespace.

Rule 9-89. No use of xs:include

[Rule 9-89] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:include">  
    <sch:assert test="false()"  
      >The schema MUST NOT contain the element xs:include.</sch:assert>  
  </sch:rule>  
</sch:pattern>

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

Inclusion of schemas that do not have namespaces also complicates schema understanding. This inclusion makes it difficult to find the realization of a specific schema artifact and create aliases for schema components that should be reused. Inclusion of schemas also violates [Principle 9, above](#principle_9), as it uses multiple schemas to construct a namespace.

Rule 9-90. xs:import must have namespace

[Rule 9-90] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:import">  
    <sch:assert test="exists(@namespace)"  
      >An element xs:import MUST have an attribute {}namespace.</sch:assert>  
  </sch:rule>  
</sch:pattern>

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 9-91. XML Schema document set must be complete

An [·XML Schema document set·](#definition_XML_Schema_document_set) defines an [·XML Schema·](#definition_XML_Schema) that may be used to validate an [·XML document·](#definition_XML_document). This rule ensures that a schema document set under consideration contains definitions for everything that it references; it has everything necessary to do a complete validation of XML documents, without any unresolved references. Note that some tools may allow validation of documents using partial schemas, when components that are not present are not exercised by the XML document under validation. Such a schema does not meet qualify as a [·conformant schema document set·](#Xeff662d8d67f5b4e72a2955d729e0ac83cad2d6).

[Rule 9-91] (SET) (Constraint)

The schema document set must constitute a complete XML Schema; it must contain the definition of every schema component referenced by any component defined by the schema set.

### 9.8.1. Namespaces for referenced components are imported

The [·XML Schema definition language·](#X7e4888920c30d9bbda28173a5b8b96fa0c14a07) requires that, when a [·schema document·](#definition_schema_document) references a component in some other namespace, it must use xs:import to import the namespace of the referenced component. The use of xs:import is described by [**[XML Schema Structures]**](#XMLSchema-1) [Section 4.2.3, *References to schema components across namespaces*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#nsi-schema_components).

Some tools do not enforce this constraint; one such tool carries imports from a [·schema document·](#definition_schema_document) into [·schema documents·](#definition_schema_document) that it imports. This has the potential to introduce incompatibility into schema documents and schema document sets that exercise this bug. To maintain compatibility across tool sets, this requirement is an explicit rule for NIEM-conformant schemas.

Rule 9-92. Namespace referenced by attribute type is imported

[Rule 9-92] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[@type]">  
    <sch:assert test="every $namespace in namespace-uri-from-QName(resolve-QName(@type, .)) satisfies (  
                        $namespace = nf:get-target-namespace(.)  
                        or $namespace = xs:anyURI('http://www.w3.org/2001/XMLSchema')  
                        or nf:get-document-element(.)/xs:import[xs:anyURI(@namespace) = $namespace])"  
                >The namespace of a type referenced by @type MUST be the target namespace, the XML Schema namespace, or be imported.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-93. Namespace referenced by attribute base is imported

[Rule 9-93] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[@base]">  
    <sch:assert test="every $namespace in namespace-uri-from-QName(resolve-QName(@base, .)) satisfies (  
                        $namespace = nf:get-target-namespace(.)  
                        or $namespace = xs:anyURI('http://www.w3.org/2001/XMLSchema')  
                        or nf:get-document-element(.)/xs:import[xs:anyURI(@namespace) = $namespace])"  
                >The namespace of a type referenced by @base MUST be the target namespace, the XML Schema namespace, or be imported.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-94. Namespace referenced by attribute itemType is imported

[Rule 9-94] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[@itemType]">  
    <sch:assert test="every $namespace in namespace-uri-from-QName(resolve-QName(@itemType, .)) satisfies (  
                        $namespace = nf:get-target-namespace(.)  
                        or $namespace = xs:anyURI('http://www.w3.org/2001/XMLSchema')  
                        or nf:get-document-element(.)/xs:import[xs:anyURI(@namespace) = $namespace])"  
                >The namespace of a type referenced by @itemType MUST be the target namespace, the XML Schema namespace, or be imported.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-95. Namespaces referenced by attribute memberTypes is imported

[Rule 9-95] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[@memberTypes]">  
    <sch:assert test="every $type in tokenize(normalize-space(@memberTypes), ' '),  
                            $namespace in namespace-uri-from-QName(resolve-QName($type, .)) satisfies (  
                        $namespace = nf:get-target-namespace(.)  
                        or $namespace = xs:anyURI('http://www.w3.org/2001/XMLSchema')  
                        or nf:get-document-element(.)/xs:import[xs:anyURI(@namespace) = $namespace])"  
                >The namespace of a type referenced by @memberTypes MUST be the target namespace, the XML Schema namespace, or be imported.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-96. Namespace referenced by attribute ref is imported

[Rule 9-96] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[@ref]">  
    <sch:assert test="every $namespace in namespace-uri-from-QName(resolve-QName(@ref, .)) satisfies  
                        $namespace = nf:get-target-namespace(.)  
                        or nf:get-document-element(.)/xs:import[xs:anyURI(@namespace) = $namespace]"  
                >The namespace of a component referenced by @ref MUST be the target namespace or be imported.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 9-97. Namespace referenced by attribute substitutionGroup is imported

[Rule 9-97] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[@substitutionGroup]">  
    <sch:assert test="every $namespace in namespace-uri-from-QName(resolve-QName(@substitutionGroup, .)) satisfies  
                        $namespace = nf:get-target-namespace(.)  
                        or nf:get-document-element(.)/xs:import[xs:anyURI(@namespace) = $namespace]"  
                >The namespace of a component referenced by @substitutionGroup MUST be the target namespace or be imported.</sch:assert>  
  </sch:rule>  
</sch:pattern>

# 10. Rules for NIEM modeling, by NIEM concept

This section focuses on building NIEM data models using the [·XML Schema definition language·](#X7e4888920c30d9bbda28173a5b8b96fa0c14a07). Whereas [Section 9, *Rules for a NIEM profile of XML Schema*, above,](#section_9) addressed shrinking the XML Schema definition language to a smaller set of features, this section constructs new NIEM-specific features to address modeling and interoperability problems. This includes naming rules, categories of types, and augmentations.

NIEM provides a framework for modeling concepts and relationships as XML artifacts. The data model is implemented via XML Schema. However, XML Schema does not provide sufficient structure and constraint to enable translating from a conceptual model to a schema and then to instances of the concepts. NIEM provides additional support for modeling concepts as schemas and provides rules for creating and connecting data that realizes those concepts.

Underlying the NIEM data model are two namespaces: the [·structures namespace·](#definition_structures_namespace) and the [·appinfo namespace·](#definition_appinfo_namespace). These two namespaces provide schema components that serve two functions:

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

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

This section is organized by concept, rather than component type. This section is integrated with the following sections:

* [Section 11, *Rules for NIEM modeling, by XML Schema component*, below,](#section_11) is organized by component type, and provides a majority of the constraint rules for schemas that define NIEM models.
* [Section 12, *XML instance document rules*, below,](#section_12) provides rules for XML documents that are instances of NIEM models.

Concepts covered by this section include:

* [Section 10.1, *Categories of NIEM type definitions*](#section_10.1)
* [Section 10.2, *Object*](#section_10.2)
* [Section 10.3, *Associations*](#section_10.3)
* [Section 10.4, *Augmentations*](#section_10.4)
* [Section 10.5, *Metadata*](#section_10.5)
* [Section 10.6, *Container elements*](#section_10.6)
* [Section 10.7, *The “Representation” pattern*](#section_10.7)
* [Section 10.8, *Naming rules*](#section_10.8)
* [Section 10.9, *Machine-readable annotations*](#section_10.9)
* [Section 10.10, *NIEM structural facilities*](#section_10.10)

## 10.1. Categories of NIEM type definitions

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 example, for complex types, the names follow a pattern:

* Name ends with “AssociationType” → type is an association type.
* Name ends with “MetadataType” → type is a metadata type.
* Name ends with “AugmentationType” → type is an augmentation type.
* Otherwise → type is an object type.

Rule 10-1. Complex type has a category

[Rule 10-1] (REF, EXT) (Constraint)

Within the schema, a complex type definition MUST be one of the following:

* An object type.
* An association type.
* A metadata type.
* An augmentation type.

This rule establishes the categories of NIEM complex types. It is a limited set, and each category has distinct semantics, as described by following sections.

## 10.2. Object

The categories of objects are covered by the following sections:

* [Section 10.2.1, *General object types*](#section_10.2.1)
* [Section 10.2.2, *Role types and roles*](#section_10.2.2)
* [Section 10.2.3, *External adapter types and external components*](#section_10.2.3)
* [Section 10.2.4, *Code types*](#section_10.2.4)
* [Section 10.2.5, *Proxy types*](#section_10.2.5)

### 10.2.1. General object types

[Definition: object type]

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

#### 10.2.1.1. Object types with complex content

Rule 10-2. Object type with complex content is derived from structures:ObjectType

[Rule 10-2] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType[exists(xs:complexContent)  
                                    and not(ends-with(@name, 'AssociationType')  
                                        or ends-with(@name, 'MetadataType')  
                                        or ends-with(@name, 'AugmentationType'))]">  
    <sch:assert test="  
        every $derivation-method in (xs:complexContent/xs:extension, xs:complexContent/xs:restriction),  
              $base in $derivation-method/@base,  
              $base-qname in resolve-QName($base, $derivation-method),  
              $base-local-name in local-name-from-QName($base-qname) satisfies (  
          $base-qname = xs:QName('structures:ObjectType')  
          or not(ends-with($base-local-name, 'AssociationType')  
                 or ends-with($base-local-name, 'MetadataType')  
                 or ends-with($base-local-name, 'AugmentationType')))"  
      >An object type with complex content MUST be derived from structures:ObjectType or from another object type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

A [·base type definition·](#definition_base_type_definition) with a {target namespace} that is the XML namespace is prohibited by [Rule 9-1, *No base type in the XML namespace*, above](#rule_9-1). A [·base type definition·](#definition_base_type_definition) with a {target namespace} that is not imported as conformant is prohibited by [Rule 11-3, *Base type definition defined by conformant schema*, below](#rule_11-3).

### 10.2.2. Role types and roles

NIEM differentiates between an object and a role of the object. The term “role” is used here to mean a function or part played by some object. The simplest way to represent a role of an object is to use an element. The following element declaration models the role of a person who undergoes an assessment:

Figure 10-1: An element declaration that constitutes a role without the use of a role type

<xs:element name="AssessmentPerson" type="nc:PersonType" nillable="true">  
  <xs:annotation>  
    <xs:documentation>A person who undergoes an assessment.</xs:documentation>  
  </xs:annotation>  
</xs:element>

In many cases, there is a further need to represent characteristics and additional information associated with a role of an object. In such cases, the above element is insufficient. When a role must be modeled with additional information, a [·role type·](#definition_role_type) is called for.

For example, when a person is a driver involved in an automotive crash, that person has a particular role in the crash, which is modeled by the element j:CrashDriver.

Figure 10-2: Element j:CrashDriver, modeling the role of a driver in a crash

<xs:element name="CrashDriver" type="j:CrashDriverType" nillable="true">  
  <xs:annotation>  
    <xs:documentation>A motor vehicle driver involved into a traffic accident.</xs:documentation>  
  </xs:annotation>  
</xs:element>

There is more information associated with this role of the driver than just his identity as a person. Information associated with this role include the drivers license, contributing circumstances, distractions, and other properties. These characteristics are modeled as shown in the following figure. The role is modeled as a [·role type·](#definition_role_type):

Figure 10-3: Role type j:CrashDriverType, modeling a driver involved in a crash

<xs:complexType name="CrashDriverType">  
  <xs:annotation>  
    <xs:documentation>A data type for a motor vehicle driver involved in a traffic accident.</xs:documentation>  
  </xs:annotation>  
  <xs:complexContent>  
    <xs:extension base="structures:ObjectType">  
      <xs:sequence>  
        <xs:element ref="nc:RoleOfPerson" minOccurs="0" maxOccurs="unbounded"/>  
        <xs:element ref="j:DriverLicense" minOccurs="0" maxOccurs="unbounded"/>  
        <xs:element ref="j:CrashDriverContributingCircumstances" minOccurs="0" maxOccurs="unbounded"/>  
        <xs:element ref="j:CrashDriverDistraction" minOccurs="0" maxOccurs="unbounded"/>  
        <xs:element ref="j:CrashDriverViolation" minOccurs="0" maxOccurs="unbounded"/>  
        <xs:element ref="j:CrashDrivingViolation" minOccurs="0" maxOccurs="unbounded"/>  
        <xs:element ref="j:CrashDriverAugmentationPoint" minOccurs="0" maxOccurs="unbounded"/>  
      </xs:sequence>  
    </xs:extension>  
  </xs:complexContent>  
</xs:complexType>

Role types were introduced into NIEM after XML Schema extension proved to be insufficient in certain situations. An object may have multiple functions in the same instance document, each with associated data. For example, a person might be both a j:CrashDriver and a j:ArrestSubject. Without roles, information about the person would be duplicated in extensions, or would be left ambiguously blank in some places.

Role types and RoleOf elements enable more precise semantics. With roles, a single base object (e.g., a person) can have multiple roles or functions within a message (e.g., a victim, a witness, and a subject). A role type can reference a base object, rather than duplicate it. Role types allow function information to be defined once for multiple base objects, such as a single j:VictimType, which represents victim information for persons, organizations, and items (e.g., via nc:RoleOfPerson, nc:RoleOfOrganization, and nc:RoleOfItem).

The term [·role type·](#definition_role_type) has a normative definition:

[Definition: role type]

A **role type** is an [·object type·](#definition_object_type) that represents a particular function, purpose, usage, or role of one or more objects of its base type.

A [·role type·](#definition_role_type) describes a role of a thing. A role is a function or position played by something in a particular situation or context. A [·role type·](#definition_role_type) holds information that is specific to the role, but that is not specific to the context, and is not specific to thing that plays the role.

In the example ([Figure 10-5, *An XML instance of a role type*, below](#figure_10-5)), a person (John Doe) has the role of being a driver in a crash. Associated with this role are descriptions of a law violation related to him and a law violation related to his driving behavior. These are properties of his role as a driver in a crash.

This data is described by a [·role type·](#definition_role_type), j:CrashDriverType (described by [Figure 10-3, *Role type j:CrashDriverType, modeling a driver involved in a crash*, above](#figure_10-3)), which carries elements and attributes that are properties of the role. The role type also carries a [·RoleOf element·](#definition_RoleOf_element), which indicates the base type for the role. The [·role type·](#definition_role_type) describes a role of the base type, which in this case is nc:PersonType, as identified by the type of nc:RoleOfPerson defined by [Figure 10-4, *Declaration of RoleOf element nc:RoleOfPerson*, below](#figure_10-4).

Developers of NIEM-conformant schemas and exchanges should create and use role types whenever they have information specific to a base object’s function. Such information is a characteristic of the role, and not of the base object. Information that is a characteristic of a base object probably does not belong in a role type.

[Definition: RoleOf element]

A **RoleOf element** is an [·element declaration·](#definition_element_declaration) that

* is defined by a [·reference schema document·](#definition_reference_schema_document) or an [·extension schema document·](#definition_extension_schema_document), and
* has a {name} that begins with “RoleOf”.

A RoleOf element represents a base type for a [·role type·](#definition_role_type).

We saw the use of [·RoleOf element·](#definition_RoleOf_element) nc:RoleOfPerson in [Figure 10-3, *Role type j:CrashDriverType, modeling a driver involved in a crash*, above](#figure_10-3). Its definition is the following figure:

Figure 10-4: Declaration of RoleOf element nc:RoleOfPerson

<xs:element name="RoleOfPerson" type="nc:PersonType" substitutionGroup="nc:RoleOf" nillable="true">  
  <xs:annotation>  
    <xs:documentation>A person of whom the role object is a function.</xs:documentation>  
  </xs:annotation>  
</xs:element>

Here is an example of the j:CrashDriver role type used in an instance:

Figure 10-5: An XML instance of a role type

<j:CrashDriver>  
  <nc:RoleOfPerson structures:ref="BRAVO" xsi:nil="true"/>  
  <j:CrashDriverViolationCode>A10</j:CrashDriverViolationCode>  
  <j:CrashDrivingViolationCode>S16</j:CrashDrivingViolationCode>  
</j:CrashDriver>  
<nc:Person structures:id="BRAVO">  
  <nc:PersonBirthDate>  
    <nc:Date>1966-06-06</nc:Date>  
  </nc:PersonBirthDate>  
  <nc:PersonName>  
    <nc:PersonFullName>John Doe</nc:PersonFullName>  
  </nc:PersonName>  
</nc:Person>

Note that the value of the j:CrashPerson element was indicated above using a reference; it could have been shown as a content element, instead.

Rule 10-3. RoleOf element type is an object type

[Rule 10-3] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[@name[starts-with(., 'RoleOf')]]">  
    <sch:assert test="every $type in @type,  
                            $type-local-name in local-name-from-QName(resolve-QName($type, .)) satisfies  
                        not(ends-with($type-local-name, 'AssociationType')  
                            or ends-with($type-local-name, 'MetadataType')  
                            or ends-with($type-local-name, 'AugmentationType'))"  
      >The type definition of a RoleOf element MUST be an object type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Note that by [Rule 11-13, *Element type is from conformant namespace*, below](#rule_11-13), the element’s type must be from a conformant namespace.

Rule 10-4. Only object type has RoleOf element

[Rule 10-4] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType[  
      empty(@appinfo:externalAdapterTypeIndicator)  
      and exists(descendant::xs:element[  
            exists(@ref[  
              starts-with(local-name-from-QName(resolve-QName(., ..)), 'RoleOf')])])]">  
    <sch:assert test="not(ends-with(@name, 'AssociationType')  
                          or ends-with(@name, 'MetadataType')  
                          or ends-with(@name, 'AugmentationType'))"  
      >A complex type that includes a RoleOf element in its content model MUST be an object type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Note that [·RoleOf element·](#definition_RoleOf_element) and [·object type·](#definition_object_type) are defined terms.

Rule 10-5. RoleOf elements indicate the base types of a role type

[Rule 10-5] (REF, EXT, INS) (Interpretation)

An [·element declaration·](#definition_element_declaration) that has a local name that begins with the string “RoleOf” MUST represent a base type, of which the containing element instance represents a role.

Rule 10-6. Instance of RoleOf element indicates a role object

[Rule 10-6] (INS) (Interpretation)

When a parent element has a child element valid to an [·element declaration·](#definition_element_declaration) that is a [·RoleOf element·](#definition_RoleOf_element),

* The value of the parent element is a role object.
* The value of the child element is a base object of the role object.

An instance of a [·RoleOf element·](#definition_RoleOf_element) indicates the base object of a role. The prefix “RoleOf” on the element ensures that a role object may be distinguished from other objects and that its link to the base object is also distinguishable from the additional properties that are characteristics or other data of the role.

NIEM does not require that there be only one [·RoleOf element·](#definition_RoleOf_element) within a single type, nor does it require that a particular role object have only a single occurrence of a [·RoleOf element·](#definition_RoleOf_element). However, the use of multiple [·RoleOf elements·](#definition_RoleOf_element) may not make sense in all cases. An exchange specification may restrict [·RoleOf elements·](#definition_RoleOf_element) to a single occurrence within a type.

This specification does not require that a [·RoleOf element·](#definition_RoleOf_element) occur in an instance of a [·role type·](#definition_role_type). In such a case, the instance of the role type behaves like any other type; it simply does not have a reference to its base object. For example, we may talk about a weapon’s use in an incident without referring to the car that acted as the weapon.

### 10.2.3. External adapter types and external components

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. This section provides rules under which NIEM-conformant components may be constructed from schema components that are not NIEM-conformant.

[Definition: external schema document]

An **external schema document** is any [·schema document·](#definition_schema_document) that is not one of

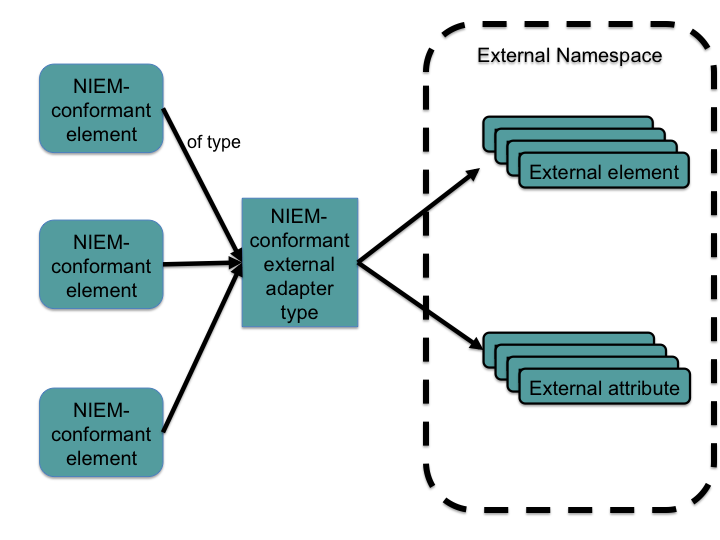
* a [·reference schema document·](#definition_reference_schema_document),
* an [·extension schema document·](#definition_extension_schema_document), or
* a [·schema document·](#definition_schema_document) that has the [·structures namespace·](#definition_structures_namespace) as its target namespace.

Note that, although the schema document for the [·structures namespace·](#definition_structures_namespace) is not conformant, it is not considered an external schema document because it defines the fundamental framework on which NIEM is built; it is considered a supporting namespace. It is not considered an external schema document because of its supporting nature and is thus a special case of this definition.

A [·schema component·](#definition_schema_component) defined by an [·external schema document·](#definition_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:

1. An external adapter type 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. This is an [·external adapter type·](#definition_external_adapter_type).
2. A type that is not an external adapter type, and which is defined by an extension schema document, may incorporate externally-defined attributes.

Figure 10-6: Use of external components to create a NIEM-conformant type



For example, a NIEM-conformant type may be created to represent a bibliographic reference from an external standard. Such an object may be composed of multiple elements (of multiple types) from the external standard. These pieces are put together to form a single NIEM-conformant [·external adapter type·](#definition_external_adapter_type). For example, an author element, a book element, and a publisher element may be used to define a single bibliographic entry type.

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

To construct such a component, a NIEM-conformant schema must import the namespace defined by the [·external schema document·](#definition_external_schema_document).

[Rule 11-50, *Reference schema document imports reference schema document*, below,](#rule_11-50) and [Rule 11-51, *Extension schema document imports reference or extension schema document*, below,](#rule_11-51) require that xs:import elements be consistent with an importing [·schema document·](#definition_schema_document).

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.

#### 10.2.3.1. Import of external namespace

Rule 10-7. Import of external namespace has data definition

[Rule 10-7] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:import[@appinfo:externalImportIndicator]">  
    <sch:assert test="some $definition in xs:annotation/xs:documentation[1] satisfies  
                        string-length(normalize-space(string($definition))) &gt; 0"  
      >An element xs:import that is annotated as importing an external schema document MUST be a documented component.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 10.2.3.2. External adapter types

[Definition: external adapter type]

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

Rule 10-8. External adapter type has indicator

[Rule 10-8] (REF, EXT) (Interpretation)

A type is an [·external adapter type·](#definition_external_adapter_type) if and only if it is a complex type with application information of attribute appinfo:externalAdapterTypeIndicator with a value of true.

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

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

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

Rule 10-9. Structure of external adapter type definition follows pattern

[Rule 10-9] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType[@appinfo:externalAdapterTypeIndicator]">  
    <sch:assert test="xs:complexContent/xs:extension[  
                        resolve-QName(@base, .) = xs:QName('structures:ObjectType')  
                      ]/xs:sequence"  
      >An external adapter type definition MUST be a complex type definition with complex content that extends structures:ObjectType, and that uses xs:sequence as its top-level compositor.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 10-10. Element use from external adapter type defined by external schema documents

[Rule 10-10] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[@ref  
                                and exists(ancestor::xs:complexType[exists(@appinfo:externalAdapterTypeIndicator)])]">  
    <sch:assert test="some $ref-namespace in namespace-uri-from-QName(resolve-QName(@ref, .)) satisfies  
                        nf:get-document-element(.)/xs:import[  
                          $ref-namespace = xs:anyURI(@namespace)  
                          and @appinfo:externalImportIndicator]"  
      >An element reference that appears within an external adapter type MUST have a target namespace that is imported as external.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 10-11. External adapter type not a base type

[Rule 10-11] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[(self::xs:extension or self::xs:restriction)  
                          and (some $base-qname in resolve-QName(@base, .),  
                                    $base-namespace in namespace-uri-from-QName($base-qname) satisfies  
                                 nf:get-target-namespace(.) = $base-namespace)]">  
    <sch:assert test="nf:resolve-type(., resolve-QName(@base, .))[  
                        empty(@appinfo:externalAdapterTypeIndicator)]"  
       >An external adapter type definition MUST NOT be a base type definition.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 10-12. External adapter type not a base type

[Rule 10-12] (SET) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[(self::xs:extension or self::xs:restriction)  
                          and (nf:has-effective-conformance-target-identifier(., xs:anyURI('http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ReferenceSchemaDocument'))  
                               or nf:has-effective-conformance-target-identifier(., xs:anyURI('http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ExtensionSchemaDocument')))  
                          and (some $base-qname in resolve-QName(@base, .),  
                                    $base-namespace in namespace-uri-from-QName($base-qname) satisfies  
                                 not($base-namespace = (nf:get-target-namespace(.), xs:anyURI('http://www.w3.org/2001/XMLSchema'))))]">  
    <sch:assert test="nf:resolve-type(., resolve-QName(@base, .))[  
                        empty(@appinfo:externalAdapterTypeIndicator)]"  
       >An external adapter type definition MUST NOT be a base type definition.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Each external adapter type is meant to stand alone; each type expresses a single concept, and is built from schema components from one or more external schema documents.

#### 10.2.3.3. External attribute use

Rule 10-13. External attribute use only in external adapter type

A complex type that is defined by a [·reference schema document·](#definition_reference_schema_document) is permitted to contain an attribute use schema component only if it is an [·external adapter type·](#definition_external_adapter_type). This does not apply to a complex type defined by an [·extension schema document·](#definition_extension_schema_document), which is permitted to use external attributes, as long as each attribute use is a [·documented component·](#definition_documented_component), per [Rule 10-14, *External attribute use has data definition*, below](#rule_10-14).

[Rule 10-13] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[some $ref-namespace in namespace-uri-from-QName(resolve-QName(@ref, .)),  
                                       $import in ancestor::xs:schema[1]/xs:import satisfies (  
                                    xs:anyURI($import/@namespace) = $ref-namespace  
                                    and exists($import/@appinfo:externalImportIndicator))]">  
    <sch:assert test="exists(ancestor::xs:complexType[1]/@appinfo:externalAdapterTypeIndicator)"  
      >An external attribute use MUST be in an external adapter type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the term [·external adapter type·](#definition_external_adapter_type).

Rule 10-14. External attribute use has data definition

[Rule 10-14] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[some $ref-namespace in namespace-uri-from-QName(resolve-QName(@ref, .)),  
                                       $import in ancestor::xs:schema[1]/xs:import satisfies (  
                                    xs:anyURI($import/@namespace) = $ref-namespace  
                                    and exists(@appinfo:externalImportIndicator))]">  
    <sch:assert test="some $documentation in xs:annotation/xs:documentation[1] satisfies  
                        string-length(normalize-space(string($documentation))) &gt; 0"  
      >An external attribute use MUST be a documented component with a non-empty data definition.</sch:assert>  
  </sch:rule>  
</sch:pattern>

The terms [·documented component·](#definition_documented_component) and [·data definition·](#definition_data_definition) are defined.

Rule 10-15. External attribute use not an ID

NIEM schemas use structures:id to enable references between components. Each NIEM-defined complex type must incorporate a definition for structures:id. [**[XML]**](#XML) [Section 3.3.1, *Attribute Types*](http://www.w3.org/TR/2008/REC-xml-20081126/#one-id-per-el) 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.

[Rule 10-15] (SET) (Constraint)

An attribute use schema component MUST NOT have an {attribute declaration} with an ID type.

The term “attribute use schema component” is defined by [**[XML Schema Structures]**](#XMLSchema-1) [Section 3.5.1, *The Attribute Use Schema Component*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#AU_details). Attribute type ID is defined by [**[XML]**](#XML) [Section 3.3.1, *Attribute Types*](http://www.w3.org/TR/2008/REC-xml-20081126/#sec-attribute-types).

#### 10.2.3.4. External element use

Rule 10-16. External element use has data definition

[Rule 10-16] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[  
      some $ref-namespace in namespace-uri-from-QName(resolve-QName(@ref, .)) satisfies  
        nf:get-document-element(.)/self::xs:schema//xs:import[  
          xs:anyURI(@namespace) = $ref-namespace  
          and @appinfo:externalImportIndicator]]">  
    <sch:assert test="some $documentation in xs:annotation/xs:documentation[1] satisfies  
                        string-length(normalize-space(string($documentation))) &gt; 0"  
      >An external attribute use MUST be a documented component with a non-empty data definition.</sch:assert>  
  </sch:rule>  
</sch:pattern>

The terms [·documented component·](#definition_documented_component) and [·data definition·](#definition_data_definition) are defined.

### 10.2.4. Code types

[Definition: code type]

A **code type** is a NIEM object type for which each simple value carried by the type corresponds to an entry in a list of distinct conceptual entities.

These types represent lists of values, each of which has a known meaning beyond the text representation. These values may be meaningful text or may be a string of alphanumeric identifiers that represent abbreviations for literals.

Many code types have simple content composed of xs:enumeration values. Code types may also be constructed using the [*NIEM Code Lists Specification*](https://reference.niem.gov/niem/specification/code-lists/4.0/niem-code-lists-4.0.html) [**[Code Lists]**](#code-lists-spec), which supports code lists defined using a variety of methods, including CSV spreadsheets.

Rule 10-17. Name of code type ends in “CodeType”

[Rule 10-17] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType[exists(xs:simpleContent[  
                       exists(xs:\*[local-name() = ('extension', 'restriction')  
                                   and (ends-with(@base, 'CodeSimpleType')  
                                   or ends-with(@base, 'CodeType'))])])]">  
    <sch:report role="warning"  
        test="not(ends-with(@name, 'CodeType'))"  
      >A complex type definition with a {base type definition} of a code type or code simple type SHOULD have a {name} that ends in 'CodeType'.</sch:report>  
  </sch:rule>  
</sch:pattern>

See [Section 11.1.2.3, *Code simple types*, below,](#section_11.1.2.3) for the definition of [·code simple type·](#definition_code_simple_type).

Rule 10-18. Code type corresponds to a code list

[Rule 10-18] (REF, EXT) (Constraint)

A complex type SHOULD have a name ending in “CodeType” if and only if it has a correspondence to a list of distinct conceptual entities.

Rule 10-19. Element of code type has code representation term

[Rule 10-19] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@name) and exists(@type) and ends-with(@type, 'CodeType')]">  
    <sch:report role="warning"  
        test="not(ends-with(@name, 'Code'))"  
      >An element with a type that is a code type SHOULD have a name with representation term "Code"</sch:report>  
  </sch:rule>  
</sch:pattern>

Using the qualifier Code (e.g. CodeType, CodeSimpleType) immediately identifies that a data component may carry values from a fixed list of codes. These types may be handled in specific ways, as lists of codes are expected to have their own lifecycles, including versions and periodic updates. Codes may also have responsible authorities behind them who provide concrete semantic bindings for the code values.

### 10.2.5. Proxy types

The NIEM 5.0 release schema http://release.niem.gov/niem/proxy/niem-xs/5.0/ provides complex type bases for some of the simple types in the XML Schema namespace. The complex types in this schema reuse the local names of the XML Schema simple types they extend, even though those names don’t follow the naming structure of most conformant complex types. There is a special exception to naming rules to allow the reuse of the XML Schema simple type names in conformant schemas. This is done to make conformant schemas more understandable to people that are familiar with the names of the XML Schema namespace simple types.

A complex type that is a direct extension of a simple type from the XML Schema namespace (e.g., xs:string, xs:integer, xs:boolean) is allowed to have the same local name as the XML Schema simple type, if and only if the extension adds no content other than the attribute group structures:SimpleObjectAttributeGroup. This allows for an intuitive name when using an XML Schema simple type in a conformant schema.

Rule 10-20. Proxy type has designated structure

[Rule 10-20] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType[some $name in @name,  
                                    $extension in xs:simpleContent/xs:extension,  
                                    $base-qname in resolve-QName($extension/@base, $extension) satisfies  
                                    $base-qname = QName('http://www.w3.org/2001/XMLSchema', $name)]">  
    <sch:assert test="xs:simpleContent[  
                        xs:extension[  
                          empty(xs:attribute)  
                          and count(xs:attributeGroup) = 1  
                          and xs:attributeGroup[  
                                resolve-QName(@ref, .) = xs:QName('structures:SimpleObjectAttributeGroup')]]]"  
      >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.</sch:assert>  
  </sch:rule>  
</sch:pattern>

## 10.3. Associations

Within NIEM data, an [·association·](#definition_association) is a specific relationship between objects. Associations are used when a simple NIEM property is insufficient to model the relationship clearly, or when you want to model properties of the relationship itself.

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

Figure 10-7: An association in an instance

<scr:PersonPhoneAssociation>  
  <nc:Person>  
    <nc:PersonName>  
      <nc:PersonFullName>John Doe</nc:PersonFullName>  
    </nc:PersonName>  
  </nc:Person>  
  <nc:TelephoneNumber>  
    <nc:FullTelephoneNumber>  
      <nc:TelephoneNumberFullID>(212)555-1212</nc:TelephoneNumberFullID>  
    </nc:FullTelephoneNumber>  
  </nc:TelephoneNumber>  
</scr:PersonPhoneAssociation>

This example shows an association between a person and a phone number. This relationship is defined by the [·association element declaration·](#X0800fc6c42d99fd31cd824ea485f2a4b7ee49f3) scr:PersonPhoneAssociation, the structure of which is defined by the [·association type·](#definition_association_type) scr:PersonPhoneAssociationType. In practice, an [·association type·](#definition_association_type) usually defines what kinds of things the association relates, while the association element may refine the meaning of the association.

An example of an association type defined by an XML Schema document follows, in [Figure 10-8, *A definition of an association type*, below](#figure_10-8).

Note that the NIEM Core schema in NIEM 5.0 defines a type nc:AssociationType, which acts as the base type for all of the other association types defined within NIEM Core. This is a convention adopted by that version of NIEM Core namespace, but is not a requirement of this document. An implementer of a NIEM-conformant schema is not required by this specification to base a new association type on nc:AssociationType.

Figure 10-8: A definition of an association type

  <xs:complexType name="PersonPhoneAssociationType">  
    <xs:annotation>  
      <xs:documentation>A data type for a relationship between a PERSON and a PHONE.</xs:documentation>  
    </xs:annotation>  
    <xs:complexContent>  
      <xs:extension base="nc:AssociationType">  
        <xs:sequence>  
          <xs:element ref="nc:Person" minOccurs="0" maxOccurs="unbounded"/>  
          <xs:element ref="nc:TelephoneNumber" minOccurs="0" maxOccurs="unbounded"/>  
        </xs:sequence>  
      </xs:extension>  
    </xs:complexContent>  
  </xs:complexType>  
  <xs:element name="PersonPhoneAssociation" type="scr:PersonPhoneAssociationType" nillable="true">  
    <xs:annotation>  
      <xs:documentation>An association between a PERSON and a PHONE.</xs:documentation>  
    </xs:annotation>  
  </xs:element>

This schema fragment shows the definition of an association type that defines a relationship between a person and a telephone number. This is followed by the definition of an association element declaration that uses the association type.

[Definition: association]

An *association* is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7) that establishes a relationship between two or more objects, optionally including properties of that relationship. An association is an instance of an [·association element declaration·](#X0800fc6c42d99fd31cd824ea485f2a4b7ee49f3), and has a type that is an [·association type·](#definition_association_type).

[Definition: association element declaration]

An **association element declaration** is an [·element declaration·](#definition_element_declaration) declared in a [·reference schema document·](#definition_reference_schema_document) or [·extension schema document·](#definition_extension_schema_document) that describes a relationship between two or more objects, optionally including properties of that relationship.

[Definition: association type]

In a NIEM-conformant schema, an **association type** is a [·complex type definition·](#definition_complex_type_definition) defined in a [·reference schema document·](#definition_reference_schema_document) or [·extension schema document·](#definition_extension_schema_document) that establishes a relationship between two or more objects, optionally including the properties of that relationship.

### 10.3.1. Association types

Rule 10-21. Association type derived from structures:AssociationType

[Rule 10-21] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType">  
    <sch:let name="is-association-type" value="exists(@name [ends-with(., 'AssociationType')])"/>  
    <sch:let name="has-association-base-type" value="  
      exists(xs:complexContent[  
        exists(xs:\*[local-name() = ('extension', 'restriction')  
                    and exists(@base [ends-with(., 'AssociationType')])])])"/>  
    <sch:assert test="$is-association-type = $has-association-base-type"  
      >A type MUST have an association type name if and only if it is derived from an association type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Associations types are easily identifiable as such and have a common base type, structures:AssociationType. Using the qualifier Association immediately identifies a type as an [·association type·](#definition_association_type).

### 10.3.2. Association element declarations

Rule 10-22. Association element type is an association type

[Rule 10-22] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@name)]">  
    <sch:assert test="exists(@type [ends-with(., 'AssociationType')])  
                      = exists(@name [ends-with(., 'Association')])"  
      >An element MUST have a name that ends in 'Association' if and only if it has a type that is an association type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Using the qualifier Association immediately identifies an element as representing an association. This document defines the term [·association type·](#definition_association_type).

## 10.4. Augmentations

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

* j:PersonMedicalInsuranceIndicator
* j:PersonProfessionalCertificateText
* j:PersonSightedIndicator
* j:PersonFBIIdentification

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

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

This approach turns into a morass that is difficult to use and maintain, and which does not scale to support the breadth of the NIEM community.

To handle this need, NIEM has adopted augmentations. There are several parts that fit together for the definition and use of augmentations:

* Each object type and association type carries an augmentation point element, which provides a place for domain- and extension-specific content in the type. Augmentation points are optional in extension schema documents, but must appear in reference schema documents. Augmentation points are also defined for the base types for [·object types·](#definition_object_type) and [·association types·](#definition_association_type):
* structures:ObjectType has augmentation point structures:ObjectAugmentationPoint.
* structures:AssociationType has augmentation point structures:AssociationAugmentationPoint.
* A developer of a domain or other schema that extends common types may define elements that are substitutable for an augmentation point. Each of these elements expresses an additional characteristic or relationship of the base type.
* A developer may also define an augmentation type, and a corresponding augmentation element of that type, which acts as a container of elements that apply to the base type. An augmentation element is defined to be substitutable for an augmentation point, which enables it to appear in instances of that base type.
* A developer of an information exchange may choose, through selection and subsetting reference schemas:
* Which types will carry augmentation point elements, and the cardinality (i.e. minOccurs, maxOccurs) of those augmentation point elements
* Which elements will be included that substitute for an augmentation point. Some of these may be direct elements, while others may have an augmentation type, and carry multiple elements for the base type.
* Which elements will be carried within the augmentation types; each of these will apply to its base type.
* Augmentation point elements also appear on structures:ObjectType and structures:AssociationType. This allows schema developers to define elements that may be applied to *any* object or association.

In addition, a developer may create an extension of a base type and have it carry augmentation elements, in order to avoid element substitution and flexible content models; whether to substitute or concretely use the elements is at the discretion of the developers and implementers of an information exchange.

The term [·augmentation·](#definition_augmentation) describes any element in a NIEM-conformant instance XML document that appears as a result of being substituted for an augmentation point element. Such an element may have a type that is an [·augmentation type·](#definition_augmentation_type), or it may be a direct element that represents a property.

[Definition: augmentation]

An **augmentation** is a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7) that:

1. is an instance of an [·augmentation element declaration·](#Xfaa24539372ac2c23b1e89a5bfc415f590cea1e),
2. is a child of a [·conformant element information item·](#Xb8c2328714130974c99941a242199ecbb7daea7) that is an instance of an [·augmentable type·](#definition_augmentable_type), and
3. appears in the instance as a substitution for an augmentation point element.

As an example, here is a definition for nc:PersonType, which includes its augmentation point element:

Figure 10-9: Definition of augmentable type nc:PersonType

<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>

Note that the augmentation point element has the same namespace and name as the type, with the suffix “Type” replaced by “AugmentationPoint”. The augmentation point is also defined with cardinality (minOccurs 0, maxOccurs unbounded) to ensure that it can support an arbitrary number of augmentations, and can be subset or constrained as needed. The augmentation point is the last element used by its type; it is always last in the sequence. The definition of the augmentation point is pretty simple:

Figure 10-10: Declaration of augmentation point element nc:PersonAugmentationPoint

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

The augmentation point element is abstract and typeless. This enables the substitution of elements that have simple content, elements that have complex content, or elements that are of an augmentation type. The JXDM domain defines an augmentation type and a corresponding augmentation element that substitutes for this augmentation point:

Figure 10-11: Declaration of augmentation type and element

<xs:complexType name="PersonAugmentationType">  
  <xs:annotation>  
    <xs:documentation>A data type for additional information about a person.</xs:documentation>  
  </xs:annotation>  
  <xs:complexContent>  
    <xs:extension base="structures:AugmentationType">  
      <xs:sequence>  
        <xs:element ref="j:PersonMedicalInsuranceIndicator" minOccurs="0" maxOccurs="unbounded"/>  
        <xs:element ref="j:PersonProfessionalCertificateText" minOccurs="0" maxOccurs="unbounded"/>  
        <xs:element ref="j:PersonSightedIndicator" minOccurs="0" maxOccurs="unbounded"/>  
        <xs:element ref="j:PersonFBIIdentification" minOccurs="0" maxOccurs="unbounded"/>  
      </xs:sequence>  
    </xs:extension>  
  </xs:complexContent>  
</xs:complexType>  
  
<xs:element name="PersonAugmentation" type="j:PersonAugmentationType"  
            substitutionGroup="nc:PersonAugmentationPoint">  
  <xs:annotation>  
    <xs:documentation>Additional information about a person.</xs:documentation>  
  </xs:annotation>  
</xs:element>

The augmentation type is derived from structures:AugmentationType and has a name ending in “AugmentationType”. The augmentation element uses the type, and has a name ending in “Augmentation”. Note that the thing that binds the augmentation type to the augmentation point is the augmentation element’s use of the augmentation point element as its substitution group. This means that an augmentation type can be reused for multiple types, by creating more augmentation elements that have that type, each with its own substitution group for a different augmentation point.

In addition to defining elements that use an augmentation type, a schema document may define an element that substitutes for an augmentation point element, but does not use an augmentation type. For example, the following CBRN namespace defines an augmentation to nc:ScheduleType. While an element of an augmentation type acts as a container, holding elements that apply to an augmented object, this element is a direct property of a schedule, providing a meaningful characteristic (hours of operation) for a schedule. The resulting syntax is briefer than it would be using an augmentation type; the resulting instance looks similar to how it might look if the schedule type had been extended, rather than augmented.

Figure 10-12: An augmentation that is not an augmentation type

<xs:element name="OperatingSchedule" type="cbrn:OperatingScheduleType"  
            substitutionGroup="nc:ScheduleAugmentationPoint">  
  <xs:annotation>  
    <xs:documentation>A schedule providing the beginning and ending hours of operation by weekday, for a designated time period.</xs:documentation>  
  </xs:annotation>  
</xs:element>

The type of this element is not an augmentation type. Instead, it is its own object, not meant to just add to another type:

Figure 10-13: A type used for an augmentation

<xs:complexType name="OperatingScheduleType">  
  <xs:annotation>  
    <xs:documentation>A data type for a schedule providing the beginning and ending hours of operation by weekday, for a designated time period.</xs:documentation>  
  </xs:annotation>  
  <xs:complexContent>  
    <xs:extension base="structures:ObjectType">  
      <xs:sequence>  
        <xs:element ref="cbrn:ScheduleStartDateTime" minOccurs="0" maxOccurs="unbounded"/>  
        <xs:element ref="cbrn:ScheduleEndDateTime" minOccurs="0" maxOccurs="unbounded"/>  
        <xs:element ref="cbrn:OperatingScheduleAugmentationPoint" minOccurs="0" maxOccurs="unbounded"/>  
      </xs:sequence>  
    </xs:extension>  
  </xs:complexContent>  
</xs:complexType>

This method can be used by an information exchange developer to define individual attributes that can be applied to any augmentable type.

Note that the augmentation method can introduce an additional element into every [·object type·](#definition_object_type) or [·association type·](#definition_association_type) in an exchange, which provides opportunity for some errors in schema development. It is important that developers of exchanges not introduce elements substitutable for an augmentation point into complex types multiple ways, as it can introduce XML Schema’s Unique Particle Attribution errors. A single complex type should not introduce an element via both element substitution and element reference. This constraint is also supported by [Rule 11-20, *Element or attribute declaration introduced only once into a type*, below](#rule_11-20).

### 10.4.1. Augmentable types

[Definition: augmentable type]

An *augmentable type* is [·complex type definition·](#definition_complex_type_definition) that

* is defined by either a [·reference schema document·](#definition_reference_schema_document) or by an [·extension schema document·](#definition_extension_schema_document), and
* is either
* an [·association type·](#definition_association_type), or
* an [·object type·](#definition_object_type) that has complex content and is not an external adapter type.

Rule 10-23. Augmentable type has augmentation point element

[Rule 10-23] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType[  
                       @name[not(ends-with(., 'MetadataType'))  
                             and not(ends-with(., 'AugmentationType'))]  
                       and empty(@appinfo:externalAdapterTypeIndicator)  
                       and xs:complexContent]">  
    <sch:let name="augmentation-point-qname"  
             value="QName(string(nf:get-target-namespace(.)),  
                          replace(./@name, 'Type$', 'AugmentationPoint'))"/>  
    <sch:assert test="xs:complexContent/xs:extension/xs:sequence/xs:element[  
                        @ref [resolve-QName(., ..) = $augmentation-point-qname]]"  
      >An augmentable type MUST contain an element use of its corresponding augmentation point element.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 10-24. Augmentable type has at most one augmentation point element

[Rule 10-24] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType[  
                       @name[not(ends-with(., 'MetadataType'))  
                             and not(ends-with(., 'AugmentationType'))]  
                       and empty(@appinfo:externalAdapterTypeIndicator)  
                       and xs:complexContent]">  
    <sch:let name="augmentation-point-qname"  
             value="QName(string(nf:get-target-namespace(.)),  
                          replace(./@name, 'Type$', 'AugmentationPoint'))"/>  
    <sch:assert test="count(xs:complexContent/xs:extension/xs:sequence/xs:element[  
                              @ref [resolve-QName(., ..) = $augmentation-point-qname]]) le 1"  
      >An augmentable type MUST contain no more than one element use of its corresponding augmentation point element.</sch:assert>  
  </sch:rule>  
</sch:pattern>

### 10.4.2. Augmentation point element declarations

Rule 10-25. Augmentation point element corresponds to its base type

[Rule 10-25] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@name[  
                                 matches(., 'AugmentationPoint$')])]">  
    <sch:let name="element-name" value="@name"/>  
    <sch:assert test="exists(  
                        parent::xs:schema/xs:complexType[  
                          @name = replace($element-name, 'AugmentationPoint$', 'Type')  
                          and exists(@name[  
                                  not(ends-with(., 'MetadataType'))  
                                  and not(ends-with(., 'AugmentationType'))])  
                                and empty(@appinfo:externalAdapterTypeIndicator)  
                                and exists(child::xs:complexContent)])"  
      >A schema document containing an element declaration for an augmentation point element MUST also contain a type definition for its base type, a corresponding augmentable type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the terms [·schema document·](#definition_schema_document), [·element declaration·](#definition_element_declaration), and [·augmentable type·](#definition_augmentable_type).

Rule 10-26. An augmentation point element has no type

[Rule 10-26] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@name[  
                                 matches(., 'AugmentationPoint$')])]">  
    <sch:assert test="empty(@type)"  
        >An augmentation point element MUST have no type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Because an augmentation point element has no type, it will be abstract, per [Rule 9-38, *Untyped element is abstract*, above](#rule_9-38)

Rule 10-27. An augmentation point element has no substitution group

[Rule 10-27] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@name[  
                                 matches(., 'AugmentationPoint$')])]">  
    <sch:assert test="empty(@substitutionGroup)"  
        >An augmentation point element MUST have no substitution group.</sch:assert>  
  </sch:rule>  
</sch:pattern>

### 10.4.3. Augmentation point element use

Rule 10-28. Augmentation point element is only referenced by its base type

[Rule 10-28] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType//xs:element[exists(@ref[  
                       matches(local-name-from-QName(resolve-QName(., ..)), 'AugmentationPoint$')]) ]">  
    <sch:assert test="QName(string(nf:get-target-namespace(ancestor::xs:complexType[1])), ancestor::xs:complexType[1]/@name)  
                      = QName(string(namespace-uri-from-QName(resolve-QName(@ref, .))),  
                          replace(local-name-from-QName(resolve-QName(@ref, .)), 'AugmentationPoint$', 'Type'))"  
      >An augmentation point element MUST only be referenced by its base type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 10-29. Augmentation point element use is optional

[Rule 10-29] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType//xs:element[exists(@ref[  
                           matches(local-name-from-QName(resolve-QName(., ..)), 'AugmentationPoint$')]) ]">  
    <sch:assert test="exists(@minOccurs) and xs:integer(@minOccurs) = 0"  
        >An augmentation point element particle MUST have attribute minOccurs equal to 0.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 10-30. Augmentation point element use is unbounded

[Rule 10-30] (REF) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType//xs:element[exists(@ref[  
                           matches(local-name-from-QName(resolve-QName(., ..)), 'AugmentationPoint$')]) ]">  
    <sch:assert test="exists(@maxOccurs) and string(@maxOccurs) = 'unbounded'"  
       >An augmentation point element particle MUST have attribute maxOccurs set to unbounded.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 10-31. Augmentation point element use must be last element in its base type

[Rule 10-31] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType//xs:element[exists(@ref[  
                           matches(local-name-from-QName(resolve-QName(., ..)), 'AugmentationPoint$')]) ]">  
    <sch:assert test="empty(following-sibling::\*)"  
       >An augmentation point element particle MUST be the last element occurrence in its content model.</sch:assert>  
  </sch:rule>  
</sch:pattern>

### 10.4.4. Augmentation types

[Definition: augmentation type]

An **augmentation type** is a [·complex type definition·](#definition_complex_type_definition) that

1. provides elements that establish relationships to a base object or association, rather than to the instance of the augmentation type itself, and
2. is designed to be the type of an [·augmentation element declaration·](#Xfaa24539372ac2c23b1e89a5bfc415f590cea1e).

[Definition: augmentation element declaration]

An *augmentation element declaration* is an [·element declaration·](#definition_element_declaration) that

1. has a type that is an [·augmentation type·](#definition_augmentation_type), and
2. carries elements that establish a relationship to its parent object or association
3. is designed to be used as an [·augmentation·](#definition_augmentation) of its parent object or association

This term may be mistaken for the term [·augmentation·](#definition_augmentation). An [·augmentation element declaration·](#Xfaa24539372ac2c23b1e89a5bfc415f590cea1e) is an element declaration defined by a schema, while an [·augmentation·](#definition_augmentation) is an element information item within an XML document that appears as the result of being substituted for an augmentation point.

Rule 10-32. Element within instance of augmentation type modifies base

[Rule 10-32] (INS) (Interpretation)

Given:

* Some element has a value *$base-value*.
* Value *$base-value* contains element *$augmentation*.
* Element *$augmentation* has property [element declaration] with a value that is an [·augmentation element declaration·](#Xfaa24539372ac2c23b1e89a5bfc415f590cea1e).
* Element *$augmentation* has a value, by reference or by content.
* The value of *$augmentation* contains element *$relationship*.
* Element *$relationship* has a value, by reference or by content.

Element *$relationship* establishes a relationship between *$base-value* and the value of *$relationship*.

Rule 10-33. Only an augmentation type name ends in “AugmentationType”

[Rule 10-33] (REF, EXT) (Interpretation)

A [·schema component·](#definition_schema_component) that has a name that ends in “AugmentationType” MUST be an [·augmentation type·](#definition_augmentation_type).

The primary indicator that a complex type is an [·augmentation type·](#definition_augmentation_type) is its name. Using the qualifier Augmentation immediately identifies a type as an augmentation type.

Rule 10-34. Schema component with name ending in “AugmentationType” is an augmentation type

[Rule 10-34] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[ends-with(@name, 'AugmentationType')]">  
    <sch:assert test="self::xs:complexType/xs:complexContent/xs:\*[  
                        (self::xs:extension or self::xs:restriction)  
                        and ends-with(@base, 'AugmentationType')]"  
       >An augmentation type definition schema component with {name} ending in 'AugmentationType' MUST be an augmentation type definition that is a complex type definition with complex content that extends or restricts an augmentation type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

The [·base type definition·](#definition_base_type_definition) of an augmentation type is required to be from a conformant namespace by [Rule 11-3, *Base type definition defined by conformant schema*, below](#rule_11-3).

Rule 10-35. Type derived from structures:AugmentationType is an augmentation type

[Rule 10-35] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[(self::xs:restriction or self::xs:extension)  
                          and ends-with(@base, 'AugmentationType')]">  
    <sch:assert test="ancestor::xs:complexType[ends-with(@name, 'AugmentationType')]"  
                >A type definition derived from an augmentation type MUST be an augmentation type definition</sch:assert>  
  </sch:rule>  
</sch:pattern>

This rule ensures that any type that is derived from an augmentation type, including structures:AugmentationType, is itself an augmentation type.

### 10.4.5. Augmentation element declarations

Rule 10-36. Augmentation element type is an augmentation type

[Rule 10-36] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@name)]">  
    <sch:assert test="exists(@type [ends-with(., 'AugmentationType')])  
                      = exists(@name [ends-with(., 'Augmentation')])"  
      >An element declaration MUST have a name that ends in "Augmentation" if and only if it has a type that is an augmentation type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Using the qualifier Augmentation immediately identifies an element as representing an augmentation.

Rule 10-37. Augmentation elements are not used directly

[Rule 10-37] (REF, SET) (Constraint)

Within a [·reference schema document·](#definition_reference_schema_document), a complex type definition MUST NOT have an element use of

* an [·augmentation element declaration·](#Xfaa24539372ac2c23b1e89a5bfc415f590cea1e), or
* an element declaration that is in the substitution group of an augmentation point element declaration.

## 10.5. Metadata

There are rules for the use of metadata in instance in [Section 12.4, *Instance metadata*, below](#section_12.4).

### 10.5.1. Metadata types

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

[Definition: metadata type]

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

Rule 10-38. Metadata type has data about data

[Rule 10-38] (REF, EXT) (Constraint)

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

A metadata type establishes a specific, named aggregation of data about data. Any type transitively derived from structures:MetadataType is a metadata type. Such metadata types should be used as is and additional metadata types defined for additional content.

Rule 10-39. Metadata types are derived from structures:MetadataType

[Rule 10-39] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType">  
    <sch:let name="is-metadata-type" value="exists(@name [ends-with(., 'MetadataType')])"/>  
    <sch:let name="has-metadata-base-type" value="exists(xs:complexContent[  
        exists(xs:\*[local-name() = ('extension', 'restriction')  
                    and exists(@base [ends-with(., 'MetadataType')])])])"/>  
    <sch:assert test="$is-metadata-type = $has-metadata-base-type"  
      >A type MUST be a metadata type if and only if it is derived from a metadata type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

A metadata type is derived from another metadata type, terminating in the base type structures:MetadataType. A type is easily identified as a metadata type by its name, qualified with the term Metadata.

### 10.5.2. Metadata element declarations

[Definition: metadata element declaration]

A *metadata element declaration* is an element declaration defined by a [·reference schema document·](#definition_reference_schema_document) or an [·extension schema document·](#definition_extension_schema_document) that defines a metadata object. A metadata element declaration has a name ending in “Metadata”, and a {type definition} that is a [·metadata type·](#definition_metadata_type).

There are limitations on the meaning of a metadata element in an instance; it does not establish existence of an object, nor is it a property of its containing object.

Rule 10-40. Metadata element declaration type is a metadata type

[Rule 10-40] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@name)]">  
    <sch:assert test="exists(@type [ends-with(., 'MetadataType')])  
                      = exists(@name [ends-with(., 'Metadata')])"  
      >An element MUST have a name that ends in 'Metadata' if and only if it has a type that is a metadata type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Using the qualifier Metadata immediately identifies an element as representing metadata. This document defines the term [·metadata type·](#definition_metadata_type).

Rule 10-41. Metadata element has applicable elements

Each metadata element declaration may be applied to a set of elements. Any element to which a metadata element may be validly applied is called an “applicable element” for the metadata element. A metadata element that does not explicitly specify applicability information may be applied to any NIEM-conformant element.

[Rule 10-41] (REF, EXT, SET) (Interpretation)

The set of applicable elements for a metadata element declaration are as follows:

* A metadata element declaration that has neither an attribute appinfo:appliesToTypes nor an attribute appinfo:appliesToElements may be applied to any element.
* A metadata element declaration that has either an attribute appinfo:appliesToTypes or an attribute appinfo:appliesToElements may be applied to
* any element whose qualified name is in the value of appinfo:appliesToElements, or any element with a declaration that is in the substitution group of the declaration of such an element, and to
* any element with a type with a qualified name that is in the value of appinfo:appliesToTypes, or any element with a type that is validly derived from such a type.

[Figure 10-14, *Sample use of appinfo:appliesToTypes*, below,](#figure_10-14) shows an example of appinfo:appliesToTypes, defining a metadata element that is applicable to all objects and all associations.

Figure 10-14: Sample use of appinfo:appliesToTypes

<xs:element name="Metadata" type="nc:MetadataType" nillable="true"  
    appinfo:appliesToTypes="structures:ObjectType structures:AssociationType">  
  <xs:annotation>  
    <xs:documentation>Information that further qualifies primary data; data about data.</xs:documentation>  
  </xs:annotation>  
</xs:element>

## 10.6. Container elements

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

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

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

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

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

The appearance of a container element within a NIEM type carries no additional semantics about the relationship between the property and the containing type. The use of container elements indicates only that there is a relationship; it does not provide any semantics for interpreting that relationship.

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

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

When a NIEM-conformant schema requires a new container element, it may define a new element with a concrete type and a general name, with general semantics. Any schema may define a container element when it requires one.

## 10.7. The “Representation” pattern

One need frequently faced by schema developers is for multiple representations for a single concept. For example, for a general concept of *a point in time*, there are numerous base representations, and innumerable ways to combine them. For example, the [·XML Schema definition language·](#X7e4888920c30d9bbda28173a5b8b96fa0c14a07) defines the types xs:dateTime, xs:time, xs:date, xs:gYearMonth, xs:gYear, xs:gMonthDay, xs:gDay, and xs:gMonth, each representing a point in time, or perhaps a span of time. There is a need in XML Schema to be able to represent a general concept like *a point in time*, along with a variety of representations for the concept.

Note that this need is a bit different than specialization of relationships (e.g., rdfs:subPropertyOf), in that it specifically is to be used to describe and carry the variety of representations for a particular concept. The need is to be able to represent an abstract point in time using a single type, and then to add a set of specific representations for that basic concept.

There are a variety of methods that could be used to satisfy this need. A method could be devised that uses an abstract type, along with a set of concrete extensions of that abstract type. However, there would still be a need for a set of concrete elements to carry the types, as type substitution (distinct from *element substitution*) is not widely adopted across the NIEM community. As well, it is difficult to create a network of complex types with complex content and complex types with simple content that all substitute for a base type; the XML Schema syntax is complicated. Other alternatives each have their pros and cons.

To handle this need, NIEM has adopted the “Representation” pattern, in which a type contains a representation element, and the various representations for that type are in the substitution group for that representation element.

For example, the definition of nc:DateType uses this pattern:

Figure 10-15: A type definition that uses the “Representation” pattern

<xs:complexType name="DateType">  
  <xs:annotation>  
    <xs:documentation>A data type for a calendar date.</xs:documentation>  
  </xs:annotation>  
  <xs:complexContent>  
    <xs:extension base="structures:ObjectType">  
      <xs:sequence>  
        <xs:element ref="nc:DateRepresentation" minOccurs="0" maxOccurs="unbounded"/>  
        <xs:element ref="nc:DateAccuracy" minOccurs="0" maxOccurs="1"/>  
        <xs:element ref="nc:DateMarginOfErrorDuration" minOccurs="0" maxOccurs="1"/>  
        <xs:element ref="nc:DateAugmentationPoint" minOccurs="0" maxOccurs="unbounded"/>  
      </xs:sequence>  
    </xs:extension>  
  </xs:complexContent>  
</xs:complexType>

The type has elements that represent accuracy and margin of error, as well as an augmentation point. The actual date value, however, is carried by an element that substitutes for nc:DateRepresentation. That base element, and several substitutions are defined by the NIEM Core namespace:

Figure 10-16: Element declarations that implement representations

<xs:element name="DateRepresentation" abstract="true">  
  <xs:annotation>  
    <xs:documentation>A data concept for a representation of a date.</xs:documentation>  
  </xs:annotation>  
</xs:element>  
  
<xs:element name="Date" type="niem-xs:date" substitutionGroup="nc:DateRepresentation" nillable="true">  
  <xs:annotation>  
    <xs:documentation>A full date.</xs:documentation>  
  </xs:annotation>  
</xs:element>  
  
<xs:element name="DateTime" type="niem-xs:dateTime" substitutionGroup="nc:DateRepresentation">  
  <xs:annotation>  
    <xs:documentation>A full date and time.</xs:documentation>  
  </xs:annotation>  
</xs:element>  
  
<xs:element name="YearDate" type="niem-xs:gYear" substitutionGroup="nc:DateRepresentation">  
  <xs:annotation>  
    <xs:documentation>A year.</xs:documentation>  
  </xs:annotation>  
</xs:element>  
  
<xs:element name="YearMonthDate" type="niem-xs:gYearMonth" substitutionGroup="nc:DateRepresentation">  
  <xs:annotation>  
    <xs:documentation>A year and month.</xs:documentation>  
  </xs:annotation>  
</xs:element>

This method has several benefits:

* Its implementation requires a minimum number of additional schema components. There is only the one abstract representation element; all other elements are concrete and needed to carry strongly typed values.
* It is simple to implement. The method requires only elements, and the definition of each element is quite simple.
* It is straightforward to use, as its implementation is based on element substitution, which is already widely used by NIEM.
* It is highly extensible. Any domain or exchange specification may define additional representations for any concept, by defining new elements substitutable for the representation element.

[Definition: representation element declaration]

A **representation element declaration** is an [·element declaration·](#definition_element_declaration) that is abstract, and that has a {name} that ends in “Representation”.

Rule 10-42. Name of element that ends in “Representation” is abstract

[Rule 10-42] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[@name[ends-with(., 'Representation')]]">  
    <sch:report role="warning"  
        test="empty(@abstract) or xs:boolean(@abstract) = false()"  
      >An element declaration with a name that ends in 'Representation' SHOULD have the {abstract} property with a value of "true".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 10-43. A substitution for a representation element declaration is a value for a type

[Rule 10-43] (REF, EXT) (Interpretation)

Any element declaration that is substitutable for a representation element declaration represents an optional representation of a value of a type that carries the representation element.

## 10.8. Naming rules

This section outlines the rules used to create names for NIEM data components previously discussed in this document. Data component names must be understood easily both by humans and by machine processes. These rules improve name consistency by restricting characters, terms, and syntax that could otherwise allow too much variety and potential ambiguity. These rules also improve readability of names for humans, facilitate parsing of individual terms that compose names, and support various automated tasks associated with dictionary and controlled vocabulary maintenance.

Rule 10-44. Schema component name composed of English words

[Rule 10-44] (REF, EXT) (Constraint)

The name of any XML Schema component defined by the schema SHOULD be composed of words from the English language, using the prevalent U.S. spelling, as provided by [**[OED]**](#OED).

The English language has many spelling variations for the same word. For example, American English “program” has a corresponding British spelling “programme.” This variation has the potential to cause interoperability problems when XML components are exchanged because of the different names used by the same elements. Providing users with a dictionary standard for spelling will mitigate this potential interoperability issue.

Rule 10-45. Schema component name has xml:lang

[**[XML]**](#XML) [Section 2.12, *Language Identification*](http://www.w3.org/TR/2008/REC-xml-20081126/#sec-lang-tag) defines attribute xml:lang, which establishes the language used in the contents and attributes of an XML document. To facilitate clarity and future support for languages other than US English, each component name defined in a conformant schema is in the scope of an occurrence of xml:lang, which defines the language used in the component name.

[Rule 10-45] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@name)]">  
    <sch:let name="xml-lang-attribute" value="ancestor-or-self::\*[exists(@xml:lang)][1]/@xml:lang"/>  
    <sch:assert test="exists($xml-lang-attribute)  
                      and string-length(normalize-space($xml-lang-attribute)) gt 0"  
                >The name of an XML Schema component defined by the schema MUST be in the scope of an occurrence of attribute xml:lang that has a value that is not empty.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 10-46. Schema component names have only specific characters

[Rule 10-46] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@name)]">  
    <sch:assert test="matches(@name, '^[A-Za-z0-9\-\_\.]\*$')"  
      >The name of an XML Schema component defined by the schema must be composed of only the characters uppercase 'A' through 'Z', lowercase 'a' through 'z', numbers '0' through '9', underscore, hyphen, and period.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Only the following characters are allowed in the name of an XML Schema component defined by the schema:

* 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 10-47. Punctuation in component name is a separator

[Rule 10-47] (REF, EXT) (Constraint)

The characters hyphen (“-”), underscore (“\_”) MAY appear in a component name only when used as a separator between parts of a word, phrase, or value, which would otherwise be incomprehensible without the use of a separator. The period character (“.”) MAY appear in component names only when appearing as a separator (as above) or as a decimal within a numeric value. A punctuation character MUST NOT be used as a substitute for camel case in component names, or as a method to avoid camel case in component names.

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.

Names of NIEM components follow the rules of XML Schema, by [Rule 7-3, *Document is a schema document*, above](#rule_7-3). NIEM components also follow the rules specified herein for each type of XML Schema component.

### 10.8.1. Character case

Names of conformant components use *camel case* formatting. 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 10-48. Names use camel case

[Rule 10-48] (REF, EXT) (Constraint)

The name of any XML Schema component defined by the schema MUST use the camel case formatting convention.

Rule 10-49. Attribute name begins with lower case letter

[Rule 10-49] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@name)]">  
    <sch:assert test="matches(@name, '^[a-z]')"  
      >Within the schema, any attribute declaration MUST have a name that begins with a lowercase letter  
      ('a'-'z').</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 10-50. Name of schema component other than attribute and proxy type begins with upper case letter

[Rule 10-50] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute">  
    <sch:report test="false()" role="warning">This rule does not apply to an attribute.</sch:report>  
  </sch:rule>  
  <sch:rule context="xs:complexType[some $name in @name,  
                                    $extension in xs:simpleContent/xs:extension,  
                                    $base-qname in resolve-QName($extension/@base, $extension) satisfies  
                                    $base-qname = QName('http://www.w3.org/2001/XMLSchema', $name)]">  
    <sch:report test="false()" role="warning">This rule does not apply to a proxy types.</sch:report>  
  </sch:rule>  
  <sch:rule context="xs:\*[exists(@name)]">  
    <sch:assert test="matches(@name, '^[A-Z]')"  
      >Within the schema, an XML Schema component that is not an attribute declaration or proxy type MUST have a name that begins with an upper-case letter ('A'-'Z').</sch:assert>  
  </sch:rule>  
</sch:pattern>

The preceding rules establish *lowerCamelCase* for NIEM attributes, and *UpperCamelCase* for all other NIEM components, except proxy types, defined by [Section 10.2.5, *Proxy types*, above](#section_10.2.5).

### 10.8.2. Use of acronyms and abbreviations

Acronyms and abbreviations have the ability to improve readability and comprehensibility of large, complex, or frequently used terms. They also obscure meaning and impair understanding when their definitions are not clear or when they are used injudiciously. They should be used with great care. Acronyms and abbreviations that are used must be documented and used consistently.

Rule 10-51. Names use common abbreviations

[Rule 10-51] (REF, EXT) (Constraint)

The schema SHOULD use, in defined names, the abbreviations identified by [Table 10-1, *Abbreviations used in schema component names*](#table_10-1), rather than their full meanings.

Table 10-1: Abbreviations used in schema component names

|  |  |
| --- | --- |
| Abbreviation | Full Meaning |
| ID | Identifier |
| URI | Uniform Resource Identifier |

Consistent, controlled, and documented abridged terms that are commonly used in place of full terms can support readability, clarity, and reduction of name length.

#### 10.8.2.1. Use of Acronyms, Initialisms, Abbreviations, and Jargon

The NDR establishes appinfo for local terminology that introduces such terms into the schema document in which the appinfo appears.

[Definition: local term]

In a conformant schema document, a local term is a word, phrase, acronym, or other string of characters for which all of the following are true:

1. It is not a term defined in the [**[OED]**](#OED), or it has a non-OED definition, or its word sense may not be clear.
2. It is introduced into the schema document via an appinfo element appinfo:LocalTerm.
3. It may be used in the local name of a schema component within the schema document.

Local terminology is introduced into a schema document by the use of appinfo:LocalTerm within the schema.

See additional rules constraining the use of local terminology in [Section 10.9.2, *Local terminology*, below](#section_10.9.2).

Rule 10-52. Local term declaration is local to its schema document

[Rule 10-52] (REF, EXT) (Interpretation)

An element appinfo:LocalTerm MUST establish the meaning of a [·local term·](#definition_local_term) only within the XML Schema document in which it occurs. There MUST NOT be any transitive inheritance of local terminology within schema documents that import the containing schema document.

Rule 10-53. Local terminology interpretation

[Rule 10-53] (REF, EXT) (Interpretation)

An element information item appinfo:LocalTerm MUST establish a term as follows:

* The value of the attribute term is the [·local term·](#definition_local_term); it may occur as a term within the name of a schema component within the schema document.
* The value of the attribute literal is the meaning of the local term, provided as a full, plain-text form of the term. This may be useful when a local term is an abbreviation, acronym, or diminutive form of a longer term.
* The value of the attribute definition is a dictionary-style description of the meaning of the local term.
* The value of the attribute sourceURIs is a list of URIs, each of which is an identifier or locator for an originating or authoritative document defining the term.
* Each child element information item appinfo:SourceText is a plain text citation of, reference to, or bibliographic entry for an originating or authoritative document defining the term.

### 10.8.3. Word forms

Rule 10-54. Singular form is preferred in name

[Rule 10-54] (REF, EXT) (Constraint)

A noun used as a term in the name of an XML Schema component MUST be in singular form unless the concept itself is plural.

Rule 10-55. Present tense is preferred in name

[Rule 10-55] (REF, EXT) (Constraint)

A verb used as a term in the name of an XML Schema component MUST be used in the present tense unless the concept itself is past tense.

Rule 10-56. Name does not have nonessential words

[Rule 10-56] (REF, EXT) (Constraint)

Articles, conjunctions, and prepositions MUST NOT be used in NIEM component names except where they are required for clarity or by standard convention.

Articles (e.g., a, an, the), conjunctions (e.g., and, or, but), and prepositions (e.g., at, by, for, from, in, of, to) are all disallowed in NIEM component names, unless they are required. For example, PowerOfAttorneyCode requires the preposition. These rules constrain slight variations in word forms and types to improve consistency and reduce potentially ambiguous or confusing component names.

Rule 10-57. Element or attribute name follows pattern

Elements and attributes in NIEM-conformant schemas are given names that follow a specific pattern. This pattern comes from [**[ISO 11179-5]**](#ISO11179-5).

[Rule 10-57] (REF, EXT) (Constraint)

Except as specified elsewhere in this document, any element or attribute defined within the schema SHOULD have a name that takes the form:

* Object-class qualifier terms (0 or more).
* An object class term (1).
* Property qualifier terms (0 or more).
* A property term (1).
* Representation qualifier terms (0 or more).
* A representation term (0 or 1).

Consistent naming rules are helpful for users who wish to understand components with which they are unfamiliar, as well as for users to find components with known semantics. This rule establishes the basic structure for an element or attribute name, in line with the rules for names under [**[ISO 11179-5]**](#ISO11179-5). Note that many elements with complex type should not have a representation term.

### 10.8.4. Object-class term

NIEM adopts an object-oriented approach to representation of data. Object classes represent what [**[ISO 11179-5]**](#ISO11179-5) refers to as “things of interest in a universe of discourse that may be found in a model of that universe.” An object class or object term is a word that represents a class of real-world entities or concepts. An object-class term describes the applicable context for a NIEM component.

Rule 10-58. Object-class term identifies concrete category

[Rule 10-58] (REF, EXT) (Constraint)

The object-class term of a NIEM component MUST consist of a term identifying a category of concrete concepts or entities.

The object-class term indicates the object category that this data component describes or represents. This term provides valuable context and narrows the scope of the component to an actual class of things or concepts.

Example:

Concept term: Activity

Entity term: Vehicle

### 10.8.5. Property term

Objects or concepts are usually described in terms of their characteristic properties, data attributes, or constituent subparts. Most objects can be described by several characteristics. Therefore, a property term in the name of a data component represents a characteristic or subpart of an object class and generally describes the essence of that data component.

Rule 10-59. Property term describes characteristic or subpart

[Rule 10-59] (REF, EXT) (Constraint)

A property term MUST describe or represent a characteristic or subpart of an entity or concept.

The property term describes the central meaning of the data component.

### 10.8.6. Qualifier terms

Qualifier terms modify object, property, representation, or other qualifier terms to increase semantic precision and reduce ambiguity. Qualifier terms may precede or succeed the terms they modify. The goal for the placement of qualifier terms is to generally follow the rules of ordinary English while maintaining clarity.

Rule 10-60. Name may have multiple qualifier terms

[Rule 10-60] (REF, EXT) (Constraint)

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

Rule 10-61. Name has minimum necessary number of qualifier terms

[Rule 10-61] (REF, EXT) (Constraint)

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

Rule 10-62. Order of qualifiers is not significant

[Rule 10-62] (REF, EXT) (Constraint)

The order of qualifiers MUST NOT be used to differentiate names.

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.

### 10.8.7. Representation terms

The representation terms for a component 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.

Rule 10-63. Redundant term in name is omitted

[Rule 10-63] (REF, EXT) (Constraint)

If any word in the representation term is redundant with any word in the property term, one occurrence SHOULD be deleted.

This rule, carried over from 11179, is designed to prevent repeating terms unnecessarily within component names. For example, this rule allows designers to avoid naming an element “PersonFirstNameName.”

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 10-2, *Property representation terms*, below,](#table_10-2) lists the primary representation terms and a definition for the concept associated with the use of that term. The table also lists secondary representation terms that may represent more specific uses of the concept associated with the primary representation term.

Table 10-2: Property representation terms

|  |  |  |
| --- | --- | --- |
| Primary Representation Term | Secondary Representation Term | Definition |
| Amount | - | A number of monetary units specified in a currency where the unit of currency is explicit or implied. |
| BinaryObject | - | A set of finite-length sequences of binary octets. |
|  | Graphic | A diagram, graph, mathematical curves, or similar representation |
|  | Picture | A visual representation of a person, object, or scene |
|  | Sound | A representation for audio |
|  | Video | A motion picture representation; may include audio encoded within |
| Code |  | A character string (i.e., letters, figures, and symbols) that for brevity, language independence, or precision represents a definitive value of an attribute. |
| DateTime |  | A particular point in the progression of time together with relevant supplementary information. |
|  | Date | A continuous or recurring period of time, of a duration greater than or equal to a day. |
|  | Time | A particular point in the progression of time within an unspecified 24-hour day. |
|  | Duration | An amount of time; the length of a time span. |
| ID |  | A character string to identify and distinguish uniquely one instance of an object in an identification scheme from all other objects in the same scheme together with relevant supplementary information. |
|  | URI | A string of characters used to identify (or name) a resource. The main purpose of this identifier is to enable interaction with representations of the resource over a network, typically the World Wide Web, using specific protocols. A URI is either a Uniform Resource Locator (URL) or a Uniform Resource Name (URN). The specific syntax for each is defined by [**[RFC 3986]**](#RFC3986). |
| Indicator |  | A list of two mutually exclusive Boolean values that express the only possible states of a property. |
| Measure |  | A numeric value determined by measuring an object along with the specified unit of measure. |
| Numeric |  | Numeric information that is assigned or is determined by calculation, counting, or sequencing. It does not require a unit of quantity or unit of measure. |
|  | Value | A result of a calculation. |
|  | Rate | A relative speed of change or progress. |
|  | Percent | A representation of a unitless ratio, expressed as parts of a hundred, with 100 percent representing a ratio of 1 to 1. |
| Quantity |  | A counted number of non-monetary units possibly including fractions. |
| Text | - | A character string (i.e., a finite sequence of characters) generally in the form of words of a language. |
|  | Name | A word or phrase that constitutes the distinctive designation of a person, place, thing, or concept. |
| List |  | A sequence of values. This representation term is used in tandem with another of the listed representation terms. |
| Abstract |  | An element that may represent a concept, rather than a concrete property. This representation term may be used in tandem with another of the listed representation terms. |
| Representation |  | An element that acts as a placeholder for alternative representations of the value of a type (see [Section 10.7, *The “Representation” pattern*, above](#section_10.7)). |

Rule 10-64. Element with simple content has representation term

[Rule 10-64] (REF, EXT) (Constraint)

Within the schema, the name of an element declaration that is of simple content SHOULD use an appropriate representation term as found in [Table 10-2, *Property representation terms*](#table_10-2).

This rule is also supported by [Rule 11-15, *Name of element declaration with simple content has representation term*, below,](#rule_11-15) and [Rule 11-16, *Name of element declaration with simple content has representation term*, below](#rule_11-16), which provide tests that a top-level declaration has a representation term.

Rule 10-65. Element with complex content has representation term when appropriate

[Rule 10-65] (REF, EXT) (Constraint)

Within the schema, the name of an element declaration that is of complex content, and that corresponds to a concept listed in [Table 10-2, *Property representation terms*](#table_10-2), SHOULD use a representation term from that table.

An element that represents a value listed in the table should have a representation term. It should do so even if its type is complex with multiple parts. For example, a type with multiple fields may represent an audio binary, a date, or a name.

Rule 10-66. Element with complex content has representation term only when appropriate

[Rule 10-66] (REF, EXT) (Constraint)

Within the schema, the name of an element declaration that is of complex content and that does not correspond to a concept listed in [Table 10-2, *Property representation terms*](#table_10-2) SHOULD NOT use a representation term.

## 10.9. Machine-readable annotations

XML Schema provides *application information* schema components to provide for automatic processing and machine-readable content for schemas, as described by [**[XML Schema Structures]**](#XMLSchema-1) [Section 3.13.2, *XML Representation of Annotation Schema Components*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#application_information). XML Schema also allows the use of attributes (with namespaces other than the XML Schema namespace) to carry additional information in schemas. NIEM uses these machine-readable annotations convey information that is outside schema definition and outside human-readable text definitions.

XML elements, attributes, and text content may appear as machine-readable annotations within an XML Schema document. The methods provided by XML Schema for machine-readable annotations are:

1. An element in the XML Schema namespace (e.g., xs:schema, xs:complexType, …) may carry attributes from namespaces other than the XML Schema namespace. By the rules of XML Schema, any XML Schema element may have attributes that are from other namespaces. These attributes do not participate in XML Schema validation, but may carry information useful to tools that process schemas. In [**[XML Schema Structures]**](#XMLSchema-1), these attributes are described in the XML Representation summary of XML Schema elements as “{any attributes with non-schema namespace . . .}”, for example in [Section 3.2.2, *XML Representation of Attribute Declaration Schema Components*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#element-attribute).
2. XML Schemas may include xs:appinfo elements, which may include arbitrary XML content. This XML does not participate in XML Schema validation, but may communicate useful information to schema readers or processors. These are described by [**[XML Schema Structures]**](#XMLSchema-1) in [Section 3.13.1, *The Annotation Schema Component*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#application_information) and [Section 3.13.2, *XML Representation of Annotation Schema Components*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#element-appinfo).

This document defines the term [·machine-readable annotation·](#definition_machine-readable_annotation) to normatively refer to such annotations within XML Schema documents:

[Definition: machine-readable annotation]

An information item within a schema is defined to be a machine-readable annotation when all of the following are true:

1. It is one of the following:
2. It is an element information item that is a child of an xs:appinfo element.
3. It is an attribute information item that satisfies the “{any attributes with non- schema namespace . . .}” clause within the declaration of an XML Schema element.
4. The namespace name property of the item is not the XML namespace, XML Schema namespace, or the XSI namespace.

Attributes from the XML namespace, the XML Schema namespace, and the XML Schema instance namespace have special meanings within XML Schema, and may have effects on validation, and so are not considered machine-readable annotations.

[Definition: application information]

A component is said to have **application information** of some element *$element* when the XML Schema element that defines the component has an immediate child element xs:annotation, which has an immediate child element xs:appinfo, which has as an immediate child the element *$element*.

If a component is described as having some [·application information·](#definition_application_information), this means that the elements in question appear in an xs:appinfo annotation of the element that defines the component.

The majority of uses of [·application information·](#definition_application_information) from the [·appinfo namespace·](#definition_appinfo_namespace) are described in the modeling rules for the specific component.

Rule 10-67. Machine-readable annotations are valid

[Rule 10-67] (REF, EXT) (Constraint)

Every element information item or attribute information item that appears as a machine-readable annotation in a schema MUST be a valid instance, according to its specification.

The specification for an element or attribute may be via an XML Schema, a Schematron schema, via a DTD, by some other specification, or by other means. This rule is intended to allow NIEM schema developers to leverage relevant vocabularies without being limited by the vocabulary’s method of specification, while ensuring that developers do not subvert or misuse those vocabularies.

### 10.9.1. The NIEM appinfo namespace

NIEM defines a single namespace that holds components for use in NIEM-conformant schema application information. This namespace is referred to as the [·appinfo namespace·](#definition_appinfo_namespace).

[Definition: appinfo namespace]

The **appinfo namespace** is the namespace represented by the URI “http://release.niem.gov/niem/appinfo/5.0/”.

The [·appinfo namespace·](#definition_appinfo_namespace) defines attributes which provide additional semantics for components built by NIEM-conformant schemas. The XML Schema document for the appinfo namespace appears in [Appendix C, *Appinfo namespace*, below](#appendix_C).

#### 10.9.1.1. Deprecation

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

[Definition: deprecated component]

A **deprecated component** is one that developers are discouraged from using, typically because a better alternative exists, yet which is maintained in the schema for compatibility with previous versions of the namespace.

The definition for [·deprecated component·](#definition_deprecated_component) is adapted from [**[JLS]**](#JLS) [Section 9.6.4.6, *@Deprecated*](http://docs.oracle.com/javase/specs/jls/se8/html/jls-9.html#jls-9.6.4.6).

Rule 10-68. Component marked as deprecated is deprecated component

[Rule 10-68] (REF, EXT) (Interpretation)

A [·schema component·](#definition_schema_component) that has an attribute appinfo:deprecated with a value of “true” MUST be a [·deprecated component·](#definition_deprecated_component).

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

Rule 10-69. Deprecated annotates schema component

[Rule 10-69] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[exists(@appinfo:deprecated)]">  
    <sch:assert test="namespace-uri-from-QName(node-name(.)) = xs:anyURI('http://www.w3.org/2001/XMLSchema')"  
            >The attribute appinfo:deprecated MUST be owned by an element with a namespace name http://www.w3.org/2001/XMLSchema.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 10.9.1.2. External adapters

The annotation attributes appinfo:externalImportIndicator and appinfo:externalAdapterTypeIndicator document, in a machine-readable way, which components are external (i.e., defined by schemas that are not NIEM-conformant), and which support the use of external components.

Rule 10-70. External import indicator annotates import

[Rule 10-70] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[exists(@appinfo:externalImportIndicator)]">  
    <sch:assert test="exists(self::xs:import)"  
        >The attribute {http://release.niem.gov/niem/appinfo/5.0/}externalImportIndicator MUST be owned by an element xs:import.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 10-71. External adapter type indicator annotates complex type

[Rule 10-71] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[exists(@appinfo:externalAdapterTypeIndicator)]">  
    <sch:assert test="exists(self::xs:complexType)"  
            >The attribute appinfo:externalAdapterTypeIndicator MUST be owned by an element xs:complexType.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 10.9.1.3. appinfo:appliesToTypes annotation

Rule 10-72. appinfo:appliesToTypes annotates metadata element

[Rule 10-72] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[exists(@appinfo:appliesToTypes)]">  
    <sch:assert test="exists(self::xs:element[exists(@name)  
                               and ends-with(@name, 'Metadata')])"  
      >The attribute appinfo:appliesToTypes MUST be owned by a metadata element.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 10-73. appinfo:appliesToTypes references types

[Rule 10-73] (SET) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[exists(@appinfo:appliesToTypes)]">  
    <sch:assert test="every $item in tokenize(normalize-space(@appinfo:appliesToTypes), ' ') satisfies  
                        exists(nf:resolve-type(., resolve-QName($item, .)))"  
      >Every item in @appinfo:appliesToTypes MUST resolve to a type.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 10.9.1.4. appinfo:appliesToElements annotation

Rule 10-74. appinfo:appliesToElements annotates metadata element

[Rule 10-74] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[exists(@appinfo:appliesToElements)]">  
    <sch:assert test="exists(self::xs:element[  
                          exists(@name)  
                          and ends-with(@name, 'Metadata')])"  
            >The attribute appinfo:appliesToElements MUST be owned by a metadata element.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 10-75. appinfo:appliesToElements references elements

[Rule 10-75] (SET) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[exists(@appinfo:appliesToElements)]">  
    <sch:assert test="every $item in tokenize(normalize-space(@appinfo:appliesToElements), ' ') satisfies  
                        count(nf:resolve-element(., resolve-QName($item, .))) = 1"  
      >Every item in @appinfo:appliesToElements MUST resolve to an element.</sch:assert>  
  </sch:rule>  
</sch:pattern>

### 10.9.2. Local terminology

Rule 10-76. appinfo:LocalTerm annotates schema

[Rule 10-76] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="appinfo:LocalTerm">  
    <sch:assert test="parent::xs:appinfo[parent::xs:annotation[parent::xs:schema]]"  
      >The element appinfo:LocalTerm MUST be application information on an element xs:schema.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the term [·application information·](#definition_application_information).

Rule 10-77. appinfo:LocalTerm has literal or definition

[Rule 10-77] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="appinfo:LocalTerm">  
    <sch:assert test="exists(@literal) or exists(@definition)"  
            >The element {http://release.niem.gov/niem/appinfo/5.0/}LocalTerm MUST have a literal or definition.</sch:assert>  
  </sch:rule>  
</sch:pattern>

## 10.10. NIEM structural facilities

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

[Definition: structures namespace]

The **structures namespace** is the namespace represented by the URI “http://release.niem.gov/niem/structures/5.0/”.

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

Rule 10-78. Use structures consistent with specification

[Rule 10-78] (REF, EXT, INS, SET) (Constraint)

Any schema or instance MUST use the NIEM [·structures namespace·](#definition_structures_namespace) consistent with the schema as it is defined in [Appendix B, *Structures namespace*, below](#appendix_B).

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. However, users may profile the structures namespace, as needed.

# 11. Rules for NIEM modeling, by XML Schema component

This section focuses on building NIEM data models using XML schema. Whereas [Section 9, *Rules for a NIEM profile of XML Schema*, above,](#section_9) addressed shrinking the XML Schema definition language to a smaller set of features, this section constructs new NIEM-specific features to address modeling and interoperability problems. This includes naming rules, categories of types, and augmentations.

## 11.1. Type definition components

Rule 11-1. Name of type ends in “Type”

[Rule 11-1] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType[some $name in @name,  
                                    $extension in xs:simpleContent/xs:extension,  
                                    $base-qname in resolve-QName($extension/@base, $extension) satisfies  
                                    $base-qname = QName('http://www.w3.org/2001/XMLSchema', $name)]">  
    <sch:report test="false()" role="warning">The name of a proxy type does not end in "Type".</sch:report>  
  </sch:rule>  
  <sch:rule context="xs:\*[(self::xs:simpleType or self::xs:complexType) and exists(@name)]">  
    <sch:assert test="ends-with(@name, 'Type')"  
      >A type definition schema component that does not define a proxy type MUST have a name that ends in "Type".</sch:assert>  
  </sch:rule>  
</sch:pattern>

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.

Note that the first sch:rule and subsequent sch:report serve to provide an exception to the rule for proxy types. It does not establish a constraint on the data.

Rule 11-2. Only types have name ending in “Type” or “SimpleType”

[Rule 11-2] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@name) and ends-with(@name, 'SimpleType')]">  
    <sch:assert test="local-name() = 'simpleType'"  
                >A schema component with a name that ends in 'SimpleType' MUST be a simple type definition.</sch:assert>  
  </sch:rule>  
  <sch:rule context="xs:\*[exists(@name) and ends-with(@name, 'Type')]">  
    <sch:assert test="local-name() = 'complexType'"  
                >A schema component with a name that ends in 'Type' and does not end in 'SimpleType' MUST be a complex type definition.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Use of the representation terms “Type” and “SimpleType” helps identify complex and simple types, respectively, in a NIEM-conformant schema, and helps to prevent naming collisions with elements and attributes.

### 11.1.1. Type definition hierarchy

Rule 11-3. Base type definition defined by conformant schema

[Rule 11-3] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:\*[exists(@base)]">  
    <sch:assert test="some $base-namespace in namespace-uri-from-QName(resolve-QName(@base, .)) satisfies (  
                        $base-namespace = (nf:get-target-namespace(.), xs:anyURI('http://www.w3.org/2001/XMLSchema'))  
                        or exists(ancestor::xs:schema[1]/xs:import[exists(@namespace)  
                                                                   and $base-namespace = xs:anyURI(@namespace)  
                                                                   and empty(@appinfo:externalImportIndicator)]))"  
      >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.</sch:assert>  
  </sch:rule>  
</sch:pattern>

### 11.1.2. Simple type definition

Rule 11-4. Name of simple type ends in “SimpleType”

[Rule 11-4] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:simpleType[@name]">  
    <sch:assert test="ends-with(@name, 'SimpleType')"  
      >A simple type definition schema component MUST have a name that ends in "SimpleType".</sch:assert>  
  </sch:rule>  
</sch:pattern>

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.

#### 11.1.2.1. Derivation by list

Rule 11-5. Use lists only when data is uniform

[Rule 11-5] (REF, EXT) (Constraint)

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

The use of lists should be reserved for cases where the data is fairly uniform.

Items in a list are not individually addressable by NIEM metadata techniques. The items may not be individually referenced by elements or attributes; one will have a value of the entire list, including all the items in the list. NIEM provides no method for individually addressing an item in a list. If an individual item in a list needs to be marked up in a manner different than other items in the list, the use of individual elements may be preferred to the definition of a list simple type.

Rule 11-6. List item type defined by conformant schemas

[Rule 11-6] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:list[exists(@itemType)]">  
    <sch:let name="namespace" value="namespace-uri-from-QName(resolve-QName(@itemType, .))"/>  
    <sch:assert test="$namespace = (nf:get-target-namespace(.), xs:anyURI('http://www.w3.org/2001/XMLSchema'))  
                      or exists(ancestor::xs:schema[1]/xs:import[exists(@namespace)  
                                    and $namespace = xs:anyURI(@namespace)  
                                    and empty(@appinfo:externalImportIndicator)])"  
      >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.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 11.1.2.2. Derivation by union

Rule 11-7. Union member types defined by conformant schemas

[Rule 11-7] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:union[exists(@memberTypes)]">  
    <sch:assert test="every $qname in tokenize(normalize-space(@memberTypes), ' '),  
                            $namespace in namespace-uri-from-QName(resolve-QName($qname, .))  
                      satisfies ($namespace = nf:get-target-namespace(.)  
                                 or exists(ancestor::xs:schema[1]/xs:import[exists(@namespace)  
                                           and $namespace = xs:anyURI(@namespace)  
                                           and empty(@appinfo:externalImportIndicator)]))"  
                >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.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 11.1.2.3. Code simple types

[Definition: code simple type]

A **code simple type** is a simple type definition schema component for which each value carried by the type corresponds to an entry in a list of distinct conceptual entities.

These types represent lists of values, each of which has a known meaning beyond the text representation. These values may be meaningful text or may be a string of alphanumeric identifiers that represent abbreviations for literals.

Many code simple types are composed of xs:enumeration values. Code simple types may also be constructed using the [*NIEM Code Lists Specification*](https://reference.niem.gov/niem/specification/code-lists/4.0/niem-code-lists-4.0.html) [**[Code Lists]**](#code-lists-spec), which supports code lists defined using a variety of methods, including CSV spreadsheets.

Rule 11-8. Name of a code simple type ends in “CodeSimpleType”

[Rule 11-8] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:simpleType[exists(@name)  
      and (xs:restriction/xs:enumeration  
           or xs:restriction[ends-with(local-name-from-QName(resolve-QName(@base, .)), 'CodeSimpleType')])]">  
    <sch:report test="not(ends-with(@name, 'CodeSimpleType'))" role="warning"  
      >A simple type definition schema component that has an enumeration facet or that is derived from a code simple type SHOULD have a name that ends in "CodeSimpleType".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-9. Code simple type corresponds to a code list

[Rule 11-9] (REF, EXT) (Constraint)

A simple type SHOULD have a name ending in “CodeSimpleType” if and only if it has a correspondence to a list of distinct conceptual entities.

Rule 11-10. Attribute of code simple type has code representation term

[Rule 11-10] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@name) and exists(@type) and ends-with(@type, 'CodeSimpleType')]">  
    <sch:report test="not(ends-with(@name, 'Code'))" role="warning"  
      >An attribute with a type that is a code simple type SHOULD have a name with representation term "Code"</sch:report>  
  </sch:rule>  
</sch:pattern>

Using the qualifier Code (e.g. CodeType, CodeSimpleType) immediately identifies that a data component may carry values from a fixed list of codes. Such a component may be handled in specific ways, as lists of codes are expected to have their own lifecycles, including versions and periodic updates. Codes may also have responsible authorities behind them who provide concrete semantic bindings for the code values.

### 11.1.3. Complex type definition

Rule 11-11. Complex type with simple content has structures:SimpleObjectAttributeGroup

Within a [·reference schema document·](#definition_reference_schema_document), a complex type with simple content can be created in one of two ways:

* By extension of an existing complex type with simple content.
* By extension of an existing simple type.

Both of these methods use the element xs:extension. Although these two methods have similar syntax, there are subtle differences. NIEM’s conformance rules ensure that any complex type has the necessary attributes for representing IDs, references, metadata, and relationship metadata. Case 1 does not require adding these attributes, as they are guaranteed to occur in the base type. However, in case 2, in which a new complex type is created from a simple type, the attributes for complex types must be added. This is done by reference to the attribute group structures:SimpleObjectAttributeGroup.

[Rule 11-11] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:simpleContent/xs:extension[  
      some $base-qname in resolve-QName(@base, .) satisfies  
        namespace-uri-from-QName($base-qname) = xs:anyURI('http://www.w3.org/2001/XMLSchema')  
        or ends-with(local-name-from-QName($base-qname), 'SimpleType')]">  
    <sch:assert test="xs:attributeGroup[  
                        resolve-QName(@ref, .) = xs:QName('structures:SimpleObjectAttributeGroup')]"  
      >A complex type definition with simple content schema component with a derivation method of extension that has a base type definition that is a simple type MUST incorporate the attribute group {http://release.niem.gov/niem/structures/5.0/}SimpleObjectAttributeGroup.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This rule ensures that a complex type with simple content that is created as an immediate extension of a simple type adds the attributes required for specific NIEM linking mechanisms.

This creates a pattern for complex type with simple content definition as follows:

Figure 11-1: Example of complex type with simple content derived from a simple type

  <xs:complexType name="PassportCategoryCodeType">  
    <xs:annotation>  
      <xs:documentation>A data type for a kind of passport.</xs:documentation>  
    </xs:annotation>  
    <xs:simpleContent>  
      <xs:extension base="nc:PassportCategoryCodeSimpleType">  
        <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>  
      </xs:extension>  
    </xs:simpleContent>  
  </xs:complexType>

## 11.2. Declaration components

### 11.2.1. Element declaration

Rule 11-12. Element type does not have a simple type name

This rule, in conjunction with [Rule 11-4, *Name of simple type ends in “SimpleType”*, above](#rule_11-4), ensures that all conformant elements will have complex types that contain attributes from the structures namespace, enabling a consistent approach for using IDs, references, metadata, relationship metadata, and security markup data.

[Rule 11-12] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@type)]">  
    <sch:assert test="not(ends-with(@type, 'SimpleType'))"  
                >The {type definition} of an element declaration MUST NOT have a {name} that ends in 'SimpleType'.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the term [·element declaration·](#definition_element_declaration).

Rule 11-13. Element type is from conformant namespace

[Rule 11-13] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@type)]">  
    <sch:assert test="for $type-qname in resolve-QName(@type, .),  
                          $type-namespace in namespace-uri-from-QName($type-qname) return  
                        $type-namespace = nf:get-target-namespace(.)  
                        or exists(nf:get-document-element(.)/xs:import[  
                                    xs:anyURI(@namespace) = $type-namespace  
                                    and empty(@appinfo:externalImportIndicator)])"  
                >The {type definition} of an element declaration MUST have a {target namespace} that is the target namespace, or one that is imported as conformant.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the term [·element declaration·](#definition_element_declaration).

Additional prohibitions on element types are defined by [Rule 9-41, *Element type not in the XML namespace*, above,](#rule_9-41) and [Rule 9-42, *Element type is not simple type*, above](#rule_9-42).

Rule 11-14. Name of element that ends in “Abstract” is abstract

[Rule 11-14] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[@name]">  
    <sch:report role="warning"  
        test="not(exists(@abstract [xs:boolean(.) = true()])  
                  eq (ends-with(@name, 'Abstract')  
                      or ends-with(@name, 'AugmentationPoint')  
                      or ends-with(@name, 'Representation')))"  
      >An element declaration SHOULD have a name that ends in 'Abstract', 'AugmentationPoint', or 'Representation' if and only if it has the {abstract} property with a value of "true".</sch:report>  
  </sch:rule>  
</sch:pattern>

#### 11.2.1.1. Object element declarations

This rule checks all cases that are testable in a single schema document.

Rule 11-15. Name of element declaration with simple content has representation term

[Rule 11-15] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[@name and @type  
                                and (some $type-qname in resolve-QName(@type, .) satisfies (  
                                       nf:get-target-namespace(.) = namespace-uri-from-QName($type-qname)  
                                       and nf:resolve-type(., $type-qname)/xs:simpleContent))]">  
    <sch:report role="warning"  
        test="every $representation-term  
              in ('Amount', 'BinaryObject', 'Graphic', 'Picture', 'Sound', 'Video', 'Code', 'DateTime', 'Date', 'Time', 'Duration', 'ID', 'URI', 'Indicator', 'Measure', 'Numeric', 'Value', 'Rate', 'Percent', 'Quantity', 'Text', 'Name', 'List')  
              satisfies not(ends-with(@name, $representation-term))"  
      >The name of an element declaration that is of simple content SHOULD use a representation term.</sch:report>  
  </sch:rule>  
</sch:pattern>

Representation terms are defined by [Table 10-2, *Property representation terms*, above](#table_10-2). This Schematron rule supports [Rule 10-64, *Element with simple content has representation term*, above](#rule_10-64).

Rule 11-16. Name of element declaration with simple content has representation term

This rule only checks the cases not testable in the (REF, EXT) version.

[Rule 11-16] (SET) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[@name and @type  
       and (nf:has-effective-conformance-target-identifier(., xs:anyURI('http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ReferenceSchemaDocument'))  
            or nf:has-effective-conformance-target-identifier(., xs:anyURI('http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ExtensionSchemaDocument')))  
       and (some $type-qname in resolve-QName(@type, .) satisfies (  
              nf:get-target-namespace(.) != namespace-uri-from-QName($type-qname)  
              and nf:resolve-type(., $type-qname)/xs:simpleContent))]">  
    <sch:report role="warning"  
        test="every $representation-term  
              in ('Amount', 'BinaryObject', 'Graphic', 'Picture', 'Sound', 'Video', 'Code', 'DateTime', 'Date', 'Time', 'Duration', 'ID', 'URI', 'Indicator', 'Measure', 'Numeric', 'Value', 'Rate', 'Percent', 'Quantity', 'Text', 'Name', 'List')  
              satisfies not(ends-with(@name, $representation-term))"  
      >the name of an element declaration that is of simple content SHOULD use a representation term.</sch:report>  
  </sch:rule>  
</sch:pattern>

Representation terms are defined by [Table 10-2, *Property representation terms*, above](#table_10-2). This rule supports [Rule 10-64, *Element with simple content has representation term*, above](#rule_10-64).

### 11.2.2. Element substitution group

Rule 11-17. Element substitution group defined by conformant schema

[Rule 11-17] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(@substitutionGroup)]">  
    <sch:let name="namespace" value="namespace-uri-from-QName(resolve-QName(@substitutionGroup, .))"/>  
    <sch:assert test="$namespace = nf:get-target-namespace(.)  
                      or exists(ancestor::xs:schema[1]/xs:import[exists(@namespace)  
                                    and $namespace = xs:anyURI(@namespace)  
                                    and empty(@appinfo:externalImportIndicator)])"  
      >An element substitution group MUST have either the target namespace or a namespace that is imported as conformant.</sch:assert>  
  </sch:rule>  
</sch:pattern>

### 11.2.3. Attribute declaration

Rule 11-18. Attribute type defined by conformant schema

[Rule 11-18] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@type)]">  
    <sch:let name="namespace" value="namespace-uri-from-QName(resolve-QName(@type, .))"/>  
    <sch:assert test="$namespace = (nf:get-target-namespace(.), xs:anyURI('http://www.w3.org/2001/XMLSchema'))  
                      or exists(ancestor::xs:schema[1]/xs:import[exists(@namespace)  
                                    and $namespace = xs:anyURI(@namespace)  
                                    and empty(@appinfo:externalImportIndicator)])"  
      >The type of an attribute declaration MUST have the target namespace or the XML Schema namespace or a namespace that is imported as conformant.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 11-19. Attribute name uses representation term

[Rule 11-19] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[exists(@name)]">  
    <sch:report role="warning"  
        test="every $representation-term  
              in ('Amount', 'BinaryObject', 'Graphic', 'Picture', 'Sound', 'Video', 'Code', 'DateTime', 'Date', 'Time', 'Duration', 'ID', 'URI', 'Indicator', 'Measure', 'Numeric', 'Value', 'Rate', 'Percent', 'Quantity', 'Text', 'Name', 'List')  
              satisfies not(ends-with(@name, $representation-term))"  
      >An attribute name SHOULD end with a representation term.</sch:report>  
  </sch:rule>  
</sch:pattern>

### 11.2.4. Notation declaration

NIEM does not define any additional features relating to notation declarations. This section is present to maintain with [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2, *XML Schema Abstract Data Model*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#concepts-data-model), See [Section 9.2.4, *Notation declaration*, above,](#section_9.2.4) for rules related to notation declarations.

## 11.3. Model group components

### 11.3.1. Model group

NIEM does not define any additional features relating to model groups. This section is present to maintain with [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2, *XML Schema Abstract Data Model*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#concepts-data-model), See [Section 9.3.1, *Model group*, above,](#section_9.3.1) for rules related to model groups.

### 11.3.2. Particle

#### 11.3.2.1. Element use

Rule 11-20. Element or attribute declaration introduced only once into a type

[Rule 11-20] (REF, EXT) (Constraint)

Within the schema, an element declaration or attribute declaration MUST NOT be introduced more than once into a type definition. This applies to content acquired by a type by any means, including from a base type definition, via element substitution groups, or through the use of attribute groups.

This rule ensures that a type definition does not incorporate a component multiple times. As information exchange specifications often contain multiple versions of schemas, including reference schemas as well as subset and constraint schemas, it may be easy to omit an element or attribute in one version of the schema, only to reincorporate it via an extension. This can cause difficulties in integrating such schemas, as it may be impossible to use a reference schema if an attribute is added twice, in both a base type and an extension type, since that would make it an invalid schema.

Incorporating a component multiple times can also make it difficult to avoid violating XML Schema’s unique particle attribution constraint, which is described by [**[XML Schema Structures]**](#XMLSchema-1) [Section 3.8.6, *Constraints on Model Group Schema Components*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#cos-nonambig). This can create difficulty if an element is added both directly, and via a substitution group head. In such a case, a parser may not be able to determine which element use is responsible for an element in an instance, which is a violation of the UPA constraint.

This rule is also intended to prevent developers from creating complicated sequences of recurring elements. Such definitions are difficult for developers to satisfy in code, and can cause havoc with XML Schema language binding tools. If an element is needed more than once, or if a particular sequence of elements is needed, a developer should consider the use of flexible content models (via substitution groups) along with additional rules.

Rule 11-21. Element reference defined by conformant schema

[Rule 11-21] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[exists(ancestor::xs:complexType[empty(@appinfo:externalAdapterTypeIndicator)]) and @ref]">  
    <sch:let name="namespace" value="namespace-uri-from-QName(resolve-QName(@ref, .))"/>  
    <sch:assert test="$namespace = nf:get-target-namespace(.)  
                      or exists(ancestor::xs:schema[1]/xs:import[exists(@namespace)  
                                    and $namespace = xs:anyURI(@namespace)  
                                    and empty(@appinfo:externalImportIndicator)])"  
      >An element reference MUST be to a component that has a namespace that is either the target namespace of the schema document in which it appears, or which is imported as conformant by that schema document.</sch:assert>  
  </sch:rule>  
</sch:pattern>

The term [·schema document·](#definition_schema_document) is a defined term.

### 11.3.3. Attribute use

Rule 11-22. Referenced attribute defined by conformant schemas

[Rule 11-22] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attribute[@ref]">  
    <sch:let name="namespace" value="namespace-uri-from-QName(resolve-QName(@ref, .))"/>  
    <sch:assert test="some $namespace in namespace-uri-from-QName(resolve-QName(@ref, .)) satisfies (  
                        $namespace = nf:get-target-namespace(.)  
                        or ancestor::xs:schema[1]/xs:import[  
                             @namespace  
                             and $namespace = xs:anyURI(@namespace)  
                             and empty(@appinfo:externalImportIndicator)])"  
      >An attribute {}ref MUST have the target namespace or a namespace that is imported as conformant.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 11.3.3.1. Attribute group use

Rule 11-23. Schema uses only known attribute groups

In conformant schemas, use of attribute groups is restricted. The only attribute group defined by NIEM for use in conformant schemas is structures:SimpleObjectAttributeGroup. This attribute group provides the attributes necessary for IDs, references, metadata, and relationship metadata. In addition, there are attributes defined by ISM and NTK namespaces, which may be used in conformant schemas. Rationale for this use is provided in [Section 7.6, *IC-ISM and IC-NTK*, above](#section_7.6).

[Rule 11-23] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:attributeGroup[@ref]">  
    <sch:assert test="some $ref in resolve-QName(@ref, .) satisfies (  
                        $ref = xs:QName('structures:SimpleObjectAttributeGroup')  
                        or namespace-uri-from-QName($ref) = (xs:anyURI('urn:us:gov:ic:ism'),  
                                                             xs:anyURI('urn:us:gov:ic:ntk')))"  
      >An attribute group reference MUST be structures:SimpleObjectAttributeGroup or have the IC-ISM or IC-NTK namespace.</sch:assert>  
  </sch:rule>  
</sch:pattern>

### 11.3.4. Wildcard

NIEM does not define any additional features relating to wildcards. This section is present to maintain with [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2, *XML Schema Abstract Data Model*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#concepts-data-model), See [Section 9.3.4, *Wildcard*, above,](#section_9.3.4) for rules related to wildcards.

## 11.4. Identity-constraint definition components

NIEM does not define any additional features relating to identity-constraint definition components. This section is present to maintain with [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2, *XML Schema Abstract Data Model*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#concepts-data-model), See [Section 9.4, *Identity-constraint definition components*, above,](#section_9.4) for rules related to identity-constraint definition components.

## 11.5. Group definition components

### 11.5.1. Model group definition

NIEM does not define any additional features relating to model group definitions. This section is present to maintain with [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2, *XML Schema Abstract Data Model*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#concepts-data-model), See [Section 9.5.1, *Model group definition*, above,](#section_9.5.1) for rules related to model group definitions.

### 11.5.2. Attribute group definition

NIEM does not define any additional features relating to attribute group definitions. This section is present to maintain with [**[XML Schema Structures]**](#XMLSchema-1) [Section 2.2, *XML Schema Abstract Data Model*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#concepts-data-model), See [Section 9.5.2, *Attribute group definition*, above,](#section_9.5.2) for rules related to attribute group definitions.

## 11.6. Annotation components

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

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

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

### 11.6.1. Human-readable documentation

Note that [Rule 11-28, *Data definition follows 11179-4 requirements*, below,](#rule_11-28) and [Rule 11-29, *Data definition follows 11179-4 recommendations*, below,](#rule_11-29) apply [**[ISO 11179-4]**](#ISO11179-4) definition rules to documented components.

Rule 11-24. Data definition does not introduce ambiguity

[Rule 11-24] (REF, EXT) (Constraint)

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.

This document defines the term [·data definition·](#definition_data_definition).

Rule 11-25. Object class has only one meaning

[Rule 11-25] (REF, EXT) (Constraint)

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

Rule 11-26. Data definition of a part does not redefine the whole

[Rule 11-26] (REF, EXT) (Constraint)

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

Data definitions should be concise, precise, and unambiguous without embedding additional definitions of data elements that have already been defined once elsewhere (such as object classes). [**[ISO 11179-4]**](#ISO11179-4) 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]**](#ISO11179-4) are not violated. This further enhances brevity and precision.

Rule 11-27. Do not leak representation into data definition

[Rule 11-27] (REF, EXT) (Constraint)

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

A component definition is intended to describe semantic meaning only, not representation or structure. How a component with simple content is represented is indicated through the representation term, but the primary source of representational information should come from the XML Schema definition of the types themselves. A developer should try to keep a component’s data definition decoupled from its representation.

Rule 11-28. Data definition follows 11179-4 requirements

[Rule 11-28] (REF, EXT) (Constraint)

Each [·data definition·](#definition_data_definition) MUST conform to the requirements for data definitions provided by [**[ISO 11179-4]**](#ISO11179-4) Section 5.2, *Requirements*.

Rule 11-29. Data definition follows 11179-4 recommendations

[Rule 11-29] (REF, EXT) (Constraint)

Each [·data definition·](#definition_data_definition) SHOULD conform to the recommendations for data definitions provided by [**[ISO 11179-4]**](#ISO11179-4) Section 5.3, *Recommendations*.

Rule 11-30. xs:documentation has xml:lang

[**[XML]**](#XML) [Section 2.12, *Language Identification*](http://www.w3.org/TR/2008/REC-xml-20081126/#sec-lang-tag) defines attribute xml:lang, which establishes the language used in the contents and attributes of an XML document. To facilitate clarity and future support for languages other than US English, each occurrence of xs:documentation in a conformant schema is in the scope of an occurrence of xml:lang, which defines the language used in the documentation.

[Rule 11-30] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:documentation">  
    <sch:let name="xml-lang-attribute" value="ancestor-or-self::\*[exists(@xml:lang)][1]/@xml:lang"/>  
    <sch:assert test="exists($xml-lang-attribute)  
                      and string-length(normalize-space($xml-lang-attribute)) gt 0"  
                >An occurrence of xs:documentation within the schema MUST be in the scope of an occurrence of attribute xml:lang that has a value that is not empty.</sch:assert>  
  </sch:rule>  
</sch:pattern>

#### 11.6.1.1. 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.

##### 11.6.1.1.1. Element and attribute opening phrases

Rule 11-31. Standard opening phrase for augmentation point element data definition

[Rule 11-31] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[ends-with(@name, 'AugmentationPoint')]/xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(starts-with(lower-case(normalize-space(.)), 'an augmentation point '))"  
      >The data definition for an augmentation point element SHOULD begin with standard opening phrase "An augmentation point...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-32. Standard opening phrase for augmentation element data definition

[Rule 11-32] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[ends-with(@name, 'Augmentation')]/xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="every $phrase  
              in ('supplements ', 'additional information about ')  
              satisfies not(starts-with(lower-case(normalize-space(.)), $phrase))"  
      >The data definition for an augmentation element SHOULD begin with the standard opening phrase "Supplements..." or "Additional information about...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-33. Standard opening phrase for metadata element data definition

[Rule 11-33] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[ends-with(@name, 'Metadata')  
                                and not(xs:boolean(@abstract) eq true())]/xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)), '(metadata about|information that further qualifies)'))"  
      >The data definition for a metadata element SHOULD begin with the standard opening phrase "Metadata about..." or "Information that further qualifies...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-34. Standard opening phrase for association element data definition

[Rule 11-34] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[ends-with(@name, 'Association')  
                                and not(xs:boolean(@abstract) eq true())]/xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)), '^an?( .\*)? (relationship|association)'))"  
      >The data definition for an association element that is not abstract SHOULD begin with the standard opening phrase "An (optional adjectives) (relationship|association)...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-35. Standard opening phrase for abstract element data definition

[Rule 11-35] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:element[xs:boolean(@abstract) = true()  
                       and not(ends-with(@name, 'AugmentationPoint'))]/xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(starts-with(lower-case(normalize-space(.)), 'a data concept'))"  
      >The data definition for an abstract element SHOULD begin with the standard opening phrase "A data concept...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-36. Standard opening phrase for date element or attribute data definition

[Rule 11-36] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[(self::xs:element or self::xs:attribute)  
                       and ends-with(@name, 'Date') and not(xs:boolean(@abstract) eq true())]  
                      /xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)), '^an?( .\*)? (date|month|year)'))"  
      >The data definition for an element or attribute with a date representation term SHOULD begin with the standard opening phrase "(A|An) (optional adjectives) (date|month|year)...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-37. Standard opening phrase for quantity element or attribute data definition

[Rule 11-37] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[(self::xs:element or self::xs:attribute)  
                       and ends-with(@name, 'Quantity') and not(xs:boolean(@abstract) eq true())]  
                      /xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)), '^an?( .\*)? (count|number)'))"  
      >The data definition for an element or attribute with a quantity representation term SHOULD begin with the standard opening phrase "An (optional adjectives) (count|number)...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-38. Standard opening phrase for picture element or attribute data definition

[Rule 11-38] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[(self::xs:element or self::xs:attribute)  
                       and ends-with(@name, 'Picture') and not(xs:boolean(@abstract) eq true())]  
                      /xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)), '^an?( .\*)? (image|picture|photograph)'))"  
      >The data definition for an element or attribute with a picture representation term SHOULD begin with the standard opening phrase "An (optional adjectives) (image|picture|photograph)".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-39. Standard opening phrase for indicator element or attribute data definition

[Rule 11-39] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[(self::xs:element or self::xs:attribute)  
                       and ends-with(@name, 'Indicator') and not(xs:boolean(@abstract) eq true())]  
                      /xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)), '^true if .\*; false (otherwise|if)'))"  
      >The data definition for an element or attribute with an indicator representation term SHOULD begin with the standard opening phrase "True if ...; false (otherwise|if)...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-40. Standard opening phrase for identification element or attribute data definition

[Rule 11-40] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[(self::xs:element or self::xs:attribute)  
                       and ends-with(@name, 'Identification') and not(xs:boolean(@abstract) eq true())]  
                      /xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)), '^an?( .\*)? identification'))"  
      >The data definition for an element or attribute with an identification representation term SHOULD begin with the standard opening phrase "(A|An) (optional adjectives) identification...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-41. Standard opening phrase for name element or attribute data definition

[Rule 11-41] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[(self::xs:element or self::xs:attribute)  
                       and ends-with(@name, 'Name') and not(xs:boolean(@abstract) eq true())]  
                      /xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)), '^(a|an)( .\*)? name'))"  
      >The data definition for an element or attribute with a name representation term SHOULD begin with the standard opening phrase "(A|An) (optional adjectives) name...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-42. Standard opening phrase for element or attribute data definition

[Rule 11-42] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[(self::xs:element or self::xs:attribute)  
                       and @name  
                       and not(ends-with(@name, 'Indicator'))  
                       and not(ends-with(@name, 'Augmentation'))  
                       and not(ends-with(@name, 'Metadata'))  
                       and not(xs:boolean(@abstract) eq true())]  
                      /xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)), '^an? '))"  
      >The data definition for an element or attribute declaration SHOULD begin with the standard opening phrase "(A|An)".</sch:report>  
  </sch:rule>  
</sch:pattern>

##### 11.6.1.1.2. Complex type opening phrases

Rule 11-43. Standard opening phrase for association type data definition

[Rule 11-43] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType[ends-with(@name, 'AssociationType')]/xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)), '^a data type for (a relationship|an association)'))"  
      >The data definition for an association type SHOULD begin with the standard opening phrase "A data type for (a relationship|an association)...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-44. Standard opening phrase for augmentation type data definition

[Rule 11-44] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType[ends-with(@name, 'AugmentationType')]/xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)),  
                          '^a data type (that supplements|for additional information about)'))"  
      >The data definition for an augmentation type SHOULD begin with the standard opening phrase "A data type (that supplements|for additional information about)...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-45. Standard opening phrase for metadata type data definition

[Rule 11-45] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType[ends-with(@name, 'MetadataType')]/xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)),  
                          '^a data type for (metadata about|information that further qualifies)'))"  
      >The data definition for a metadata type SHOULD begin with the standard opening phrase "A data type for (metadata about|information that further qualifies)...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-46. Standard opening phrase for complex type data definition

[Rule 11-46] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:complexType/xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)), '^a data type'))"  
      >The data definition for a complex type SHOULD begin with the standard opening phrase "A data type...".</sch:report>  
  </sch:rule>  
</sch:pattern>

Rule 11-47. Standard opening phrase for simple type data definition

[Rule 11-47] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:simpleType/xs:annotation/xs:documentation[1]">  
    <sch:report role="warning"  
        test="not(matches(lower-case(normalize-space(.)), '^a data type'))"  
      >The data definition for a simple type SHOULD begin with a standard opening phrase "A data type...".</sch:report>  
  </sch:rule>  
</sch:pattern>

## 11.7. Schema as a whole

### 11.7.1. xs:schema document element restrictions

Rule 11-48. Same namespace means same components

[Rule 11-48] (REF, EXT) (Interpretation)

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

Rule 11-49. Different version means different view

[Rule 11-49] (REF, EXT) (Interpretation)

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

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

## 11.8. Schema assembly

Rule 11-50. Reference schema document imports reference schema document

[Rule 11-50] (SET) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:import[  
                         nf:has-effective-conformance-target-identifier(., xs:anyURI('http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ReferenceSchemaDocument'))  
                         and exists(@namespace)  
                         and empty(@appinfo:externalImportIndicator)  
                         and not(xs:anyURI(@namespace) = (xs:anyURI('http://release.niem.gov/niem/structures/5.0/'),  
                                                          xs:anyURI('http://www.w3.org/XML/1998/namespace')))]">  
    <sch:assert test="some $schema in nf:resolve-namespace(., @namespace) satisfies  
                        nf:has-effective-conformance-target-identifier($schema, xs:anyURI('http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ReferenceSchemaDocument'))"  
      >A namespace imported as conformant from a reference schema document MUST identify a namespace defined by a reference schema document.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the term [·reference schema document·](#definition_reference_schema_document).

Rule 11-51. Extension schema document imports reference or extension schema document

[Rule 11-51] (SET) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:import[  
                         nf:has-effective-conformance-target-identifier(., xs:anyURI('http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ExtensionSchemaDocument'))  
                         and exists(@namespace)  
                         and empty(@appinfo:externalImportIndicator)  
                         and not(xs:anyURI(@namespace) = (xs:anyURI('http://release.niem.gov/niem/structures/5.0/'),  
                                                          xs:anyURI('http://www.w3.org/XML/1998/namespace')))]">  
    <sch:assert test="some $schema in nf:resolve-namespace(., @namespace) satisfies (  
                        nf:has-effective-conformance-target-identifier($schema, xs:anyURI('http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ReferenceSchemaDocument'))  
                        or nf:has-effective-conformance-target-identifier($schema, xs:anyURI('http://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/#ExtensionSchemaDocument')))"  
      >A namespace imported as conformant from an extension schema document MUST identify a namespace defined by a reference schema document or an extension schema document.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This document defines the terms [·extension schema document·](#definition_extension_schema_document) and [·reference schema document·](#definition_reference_schema_document).

### 11.8.1. Supporting namespaces are imported as conformant

There are several namespaces that are treated specially by the NIEM NDR. When the following namespaces are imported into a conformant schema document, they must be imported as if they are conformant. That is, the xs:import element must not have an attribute appinfo:externalImportIndicator with a value of “true”.

Rule 11-52. Structures imported as conformant

[Rule 11-52] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:import[exists(@namespace)  
                               and xs:anyURI(@namespace) = xs:anyURI('http://release.niem.gov/niem/structures/5.0/')]">  
    <sch:assert test="empty(@appinfo:externalImportIndicator)"  
      >An import of the structures namespace MUST NOT be labeled as an external import.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 11-53. XML namespace imported as conformant

[Rule 11-53] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:import[exists(@namespace)  
                               and xs:anyURI(@namespace) = xs:anyURI('http://www.w3.org/XML/1998/namespace')]">  
    <sch:assert test="empty(@appinfo:externalImportIndicator)"  
      >An import of the XML namespace MUST NOT be labeled as an external import.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 11-54. Each namespace may have only a single root schema in a schema set

[Rule 11-54] (SET) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:schema[exists(@targetNamespace)  
                               and (some $element  
                                   in nf:resolve-namespace(., xs:anyURI(@targetNamespace))  
                                   satisfies $element is .)]">  
    <sch:assert test="count(nf:resolve-namespace(., xs:anyURI(@targetNamespace))) = 1"  
                >A namespace may appear as a root schema in a schema set only once.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Rule 11-55. Consistently marked namespace imports

XML Schemas allows multiple xs:import elements for the same namespace, which allows for multiple sets of annotations and schema locations.

[Rule 11-55] (REF, EXT) (Constraint)

<sch:pattern>  
  <sch:rule context="xs:import">  
    <sch:let name="namespace" value="@namespace"/>  
    <sch:let name="is-conformant" value="empty(@appinfo:externalImportIndicator)"/>  
    <sch:let name="first" value="exactly-one(parent::xs:schema/xs:import[@namespace = $namespace][1])"/>  
    <sch:assert test=". is $first  
                      or $is-conformant = empty($first/@appinfo:externalImportIndicator)"  
            >All xs:import elements that have the same namespace MUST have the same conformance marking via appinfo:externalImportIndicator.</sch:assert>  
  </sch:rule>  
</sch:pattern>

# 12. XML instance document rules

This specification attempts to restrict XML instance data as little as possible while still maintaining interoperability.

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

Rule 12-1. Instance must be schema-valid

[Rule 12-1] (INS) (Constraint)

The XML document MUST be schema-valid, as assessed against a [·conformant schema document set·](#Xeff662d8d67f5b4e72a2955d729e0ac83cad2d6), composed of authoritative, comprehensive schema documents for the relevant namespaces.

The schemas that define the exchange must be authoritative. Each is the reference schema or extension schema for the namespace it defines. Application developers may use other schemas for various purposes, but for the purposes of determining conformance, the authoritative schemas are relevant.

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

NIEM embraces the use of XML Schema instance attributes, including xsi:type, xsi:nil, and xsi:schemaLocation, as specified by [**[XML Schema Structures]**](#XMLSchema-1).

## 12.1. The meaning of NIEM data

The main way that NIEM XML data represents relationships and values is via the hierarchy of XML elements in an XML document. For example, the following fragment of an XML document:

Figure 12-1: Example of content elements

<nc:Person>  
  <nc:PersonName>  
    <nc:PersonFullName>John Doe</nc:PersonFullName>  
  </nc:PersonName>  
</nc:Person>

In this instance, the XML elements describe relationships between data objects:

Table 12-1: Meaning of NIEM XML

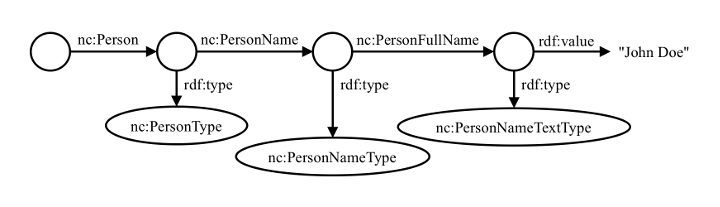
|  |  |
| --- | --- |
| XML description | The Meaning of the data |
| The top element occurs within some context, about which we do not know anything. | There is some object, representing whatever is outside the outer element. |
| The top element is nc:Person. The NIEM reference schema defines the type of the element as nc:PersonType. | There is a relationship, called nc:Person, between the unknown context object and an object of type nc:PersonType. |
| The next element is nc:PersonName. The schema indicates that element is of type nc:PersonNameType. | There is a relationship called nc:PersonName between the object of type nc:PersonType and an object of type nc:PersonNameType. |
| The next element is nc:PersonFullName. The schema shows that the element is of type nc:PersonNameTextType. | There is a relationship, called nc:PersonFullName from the object of type nc:PersonNameType and an object of type nc:PersonNameTextType. |
| Within that element is the simple value “John Doe”. The schema tells us the content of that element is of simple type xs:string. | The object of type nc:PersonNameTextType has a value that is the literal “John Doe”. |

To summarize:

1. Data objects and literal values are expressed in XML as the values of elements and attributes.
2. The names of the XML elements and attributes identify and describe the relationships between the objects and literals.
3. The names of the XML Schema complex types and simple types describe the data objects.

NIEM is designed so that NIEM XML data is a form of RDF data. This is described in some detail by [Section 5, *The NIEM conceptual model*, above](#section_5), in particular [Section 5.6.3, *Instance document mapped to RDF*, above](#section_5.6.3). The XML data, above, corresponds to the following graph. Here, the circles are data objects, and the arrows show relationships, between the objects, and between the objects and their types:

Figure 12-2: Diagram showing the meaning of XML data



Rule 12-2. Empty content has no meaning

[Rule 12-2] (INS) (Interpretation)

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

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

## 12.2. Identifiers and references

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

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

NIEM provides two different ways to solve this problem: the use of local references pointing to local identifiers, and the use of uniform resource identifiers (URIs). These two methods are similar, and can interoperate, but have distinctions, as described by [Section 12.2.3, *Differentiating reference-to-identifier links and use of URIs*, below](#section_12.2.3).

Rule 12-3. Element has only one resource identifying attribute

[Rule 12-3] (INS) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[exists(@structures:id) or exists(@structures:ref) or exists(@structures:uri)]">  
    <sch:assert test="count(@structures:id | @structures:ref | @structures:uri) le 1"  
      >An element MUST NOT have more than one attribute that is structures:id, structures:ref, or structures:uri.</sch:assert>  
  </sch:rule>  
</sch:pattern>

### 12.2.1. Local identifiers and references

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

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

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

Figure 12-3: Example of structures:id and structures:ref

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

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

Rule 12-4. Attribute structures:ref must reference structures:id

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 12-4] (INS) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[@structures:ref]">  
    <sch:let name="ref" value="@structures:ref"/>  
    <sch:assert test="exists(//\*[@structures:id = $ref])"  
      >The value of an attribute structures:ref MUST match the value of an attribute structures:id of some element in the XML document.</sch:assert>  
  </sch:rule>  
</sch:pattern>

This mirrors the terminology in [**[XML]**](#XML) [subsection *Validity constraint: IDREF*](http://www.w3.org/TR/2008/REC-xml-20081126/#idref) within [Section 3.3.1, *Attribute Types*](http://www.w3.org/TR/2008/REC-xml-20081126/#sec-attribute-types), except it requires the target attribute to be structures:id, rather than any attribute of type ID.

Rule 12-5. Linked elements have same validation root

NIEM supports type-safe references: 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 12-5] (INS) (Constraint)

Given that:

* *$element* is an [element information item](http://www.w3.org/TR/2004/REC-xml-infoset-20040204/#infoitem.element)
* *$element* has attribute structures:ref with value *$ref*
* *$element* has [property [validation context]](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#e-validation_context) with value called the *referencing element validation root*
* *$target* is an [element information item](http://www.w3.org/TR/2004/REC-xml-infoset-20040204/#infoitem.element)
* *$target* has attribute structures:id with value *$ref*
* *$target* has [property [validation context]](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#e-validation_context) with value called the *referenced element 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.

The term “validation root” is defined by [**[XML Schema Structures]**](#XMLSchema-1) [Section 5.2, *Assessing Schema-Validity*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#key-vr). It is established as a part of validity assessment of an XML document. It is required because relationships between the types of elements cannot be established if those elements were not assessed together.

Rule 12-6. Attribute structures:ref references element of correct type

[Rule 12-6] (INS) (Constraint)

Given that:

* *$element* is an [element information item](http://www.w3.org/TR/2004/REC-xml-infoset-20040204/#infoitem.element)
* *$element* has attribute structures:ref with value *$ref*
* *$element* has [property [type definition]](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#e-type_definition) with value called the *referencing element type definition*.
* *$target* is an [element information item](http://www.w3.org/TR/2004/REC-xml-infoset-20040204/#infoitem.element)
* *$target* has attribute structures:id with value *$ref*
* *$target* has [property [type definition]](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#e-type_definition) with value called the *referenced element type definition*

Every element that has an attribute structures:ref MUST have a *referenced element type definition* that is validly derived from the *referencing element type definition*.

The term **validly derived** is as established by [**[XML Schema Structures]**](#XMLSchema-1) [subsection *Schema Component Constraint: Type Derivation OK (Complex)*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#cos-ct-derived-ok) within [Section 3.4.6, *Constraints on Complex Type Definition Schema Components*](http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/#coss-ct).

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

### 12.2.2. Uniform resource identifiers in NIEM data

NIEM supports [linked data](https://www.w3.org/standards/semanticweb/data) through the use of uniform resource identifiers (URIs), expressed through the attribute structures:uri in XML documents . This attribute works much like structures:ref and structures:id, and overlaps somewhat. Linked data introduces key terminology:

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

As described in [Section 5.4, *Unique identification of data objects*, above](#section_5.4), structures:uri, structures:id, and structures:ref each denote a resource identifier. Although a structures:ref must always refer to a structures:id, and a value of structures:id must be unique within its document, a structures:uri may refer to any of structures:uri, structures:ref, or structures:id.

Rule 12-7. structures:uri denotes resource identifier

[Rule 12-7] (INS) (Interpretation)

The value of an attribute structures:uri is a URI-reference, as defined by [**[RFC 3986]**](#RFC3986), which denotes a resource identifier on the element holding the attribute, in accordance with evaluation consistent with [**[RFC 3986]**](#RFC3986) and [**[XML Base]**](#XMLBase).

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

Figure 12-4: Example of structures:uri holding an absolute URI

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

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

Figure 12-5: Example of structures:uri holding a relative URI, with an xml:base

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

The following example shows a URI fragment. The example has no xml:base, so supposing the example was from file https://example.org/path/to/file.xml, the person object has the identifier https://example.org/path/to/file.xml#first.

Figure 12-6: Example of structures:uri holding a fragment

<example:ArrestMessage>  
  <j:Arrest>  
    <j:ArrestSubject>  
      <nc:RoleOfPerson structures:uri="#first"/>  
    </j:ArrestSubject>  
  </j:Arrest>  
</example:ArrestMessage>

The attributes structures:id and structures:ref each have a mapping to equivalent values of structures:uri.

Rule 12-8. structures:id and structures:ref denote resource identifier

[Rule 12-8] (INS) (Interpretation)

The value of an attribute structures:id with a value of *$value*, or an attribute structures:ref with a value of *$value*, denotes a resource identifier on the element holding the attribute, as would be denoted by an attribute structures:uri with a value of “#”*$value*.

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

Consistent with [Section 5.4, *Unique identification of data objects*, above](#section_5.4), a set of elements that each have the same resource identifier denote the same object, which has that given identifier. This means that, in an XML representation, the properties of an object may be spread across a set of elements that share an identifier.

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

Figure 12-7: Example of structures:uri, structures:id, and structures:ref identifying the same object.

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

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

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

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

### 12.2.4. Reference and content elements have same meaning

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

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

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

[Rule 12-9] (INS) (Interpretation)

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

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

Being of type xs:ID and xs:IDREF, validating schema parsers will perform certain checks on the values of structures:id and structures:ref. Specifically, no two IDs may have the same value. This includes structures:id and other IDs that may be used within an XML document. Also, any value of structures:ref must also appear as the value of an ID.

By this rule, the following three XML fragments have a very similar meaning. First, [Figure 12-8, *Example with no reference*, below,](#figure_12-8) shows a witness that is a role of a person.

Figure 12-8: Example with no reference

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

[Figure 12-9, *Example with a backward reference*, below,](#figure_12-9) also expresses a witness object that is a role of a person. It first expresses the person object, then the witness object as a role of a that person, expressed as a reference back to the person.

Figure 12-9: Example with a backward reference

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

[Figure 12-10, *Example with a forward reference*, below,](#figure_12-10) shows a witness as a role of a person, with a separate person object expressed as a forward reference to the person object that is expressed later, within the definition of the witness.

Figure 12-10: Example with a forward reference

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

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

## 12.3. Property order and sequence identifiers

The properties of a NIEM data object, by default, have no specific order, regardless of their lexical order in XML data. A NIEM data object instance has a set of properties; these properties will appear in an XML instance as child elements of one or more parent elements. An object may be composed from multiple XML element instances, through the use of structures:id, structures:ref, and structures:uri. Order of child elements within each parent element is defined by the schema, which can specify exact element order, or can have flexible ordering (for example, elements provided via a substitution group may appear in any order). The order of parent and child elements in a document does not dictate the order of properties of an object.

A system may interpret the order of properties based on the order of XML elements within XML data; this interpretation is fine, but it is not required by the NIEM specifications. In many cases, the order of XML elements is sufficient to describe the order of properties. However, there are cases where the order of XML elements is not sufficient. One example is a proper name where the natural order of the name’s components may not be the same as the order of the elements as defined by the XML Schema. In this case, the structure defined by nc:PersonNameType defines a sequence of name parts, including given name followed by surname. This works well enough for Western names:

Figure 12-11: An instance of a name type

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

The order above does not work well for Chinese personal names, where the surname precedes the given name. For example, the basketball player Yao Ming has a given name of Ming and a surname of Yao. This cannot be expressed by the simple sequence used above, because it orders the given name before the surname. NIEM provides the attribute structures:sequenceID to explicitly express the order of properties of an object, regardless of the order of XML elements. The following example shows the structures:sequenceID attribute being used. This data is interpreted as “Yao” occurring before “Ming”.

Figure 12-12: An instance of a name type that uses structures:sequenceID

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

Use of the structures:sequenceID is called for whenever an exchange wants to concretely express the order of properties of an object. This is most useful when elements of a complex type do not appear in the desired order. This can occur when the type of an object is derived from a base type via xs:extension, in which case elements owned by the base type occur first, followed by elements added to the derived type. Another useful case is where properties of an object are be expressed across multiple elements that have the same resource identifier provided via attributes structures:id, structures:ref, or structures:uri. These properties may be provided an explicit order using structures:sequenceID.

The order of data is as yielded by the xsl:sort element, which is defined by XSLT, with data-type of xsl:number, and order of ascending. Content with identical structures:sequenceID values has undefined order.

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

Rule 12-10. Order of properties is expressed via structures:sequenceID

The order of properties within an object may be expressed through the use of attribute structures:sequenceID.

[Rule 12-10] (INS) (Interpretation)

Given two properties of object *$Object*,

* property *$Property1* with attribute structures:sequenceID with value *$Value1*, and
* property *$Property2* with attribute structures:sequenceID with value *$Value2*

If *$Value1* is less than *$Value2*, then *$Property1* MUST be interpreted as occurring before *$Property2* within *$Object*.

If *$Value2* is less than *$Value1*, then *$Property2* MUST be interpreted as occurring before *$Property1* within *$Object*.

The value of an attribute structures:sequenceID MUST be interpreted as an integer value. Comparisons between their values must be interpreted as comparisons between integers.

The relative order of two properties, where either does not have attribute structures:sequenceID is unspecified. The relative order of two properties that have the same value for attribute structures:sequenceID is unspecified.

This specification does not designate order of properties of an object where attribute structures:sequenceID does not appear.

## 12.4. Instance metadata

NIEM defines the term [·metadata type·](#definition_metadata_type) to refer to complex types that define data about other data. Metadata properties in the NIEM data model describe characteristics of data, including information about how data is collected, who reported it, creation dates, effective dates, and expiration dates. Metadata can be defined an used by exchange developers to describe characteristics of their data. Metadata is distinct from data held by regular objects, which generally describe things in the world. A developer can use metadata to describe information about the collection of data about a person, with modifying the data definitions that describe the characteristics of a person.

NIEM provides two different attributes to to attach metadata to data, and each attribute carries a meaning distinct from the other:

* Attribute structures:metadata applies metadata to an object. This attribute occurring on an element applies metadata to the object held by the element.
* Attribute structures:relationshipMetadata applies metadata to the relationship between an object and its parent. An occurrence of this attribute on an element *$element* applies metadata to the relationship established by *$element*, between the object that holds *$element* object held by *$element*.

The example below shows metadata applied to an object. In this example, the object representing the person named “Theresa Turvey” has a repository identifier (“an identifier assigned to the repository from which the information originated”) of “A-237-Z”.

Figure 12-13: A simple example of object metadata

<nc:Person structures:metadata="object-metadata-1">  
  <nc:PersonName>  
    <nc:PersonFullName>Theresa Turvey</nc:PersonFullName>  
  </nc:PersonName>  
</nc:Person>  
<nc:Metadata structures:id="object-metadata-1">  
  <nc:RepositoryID>A-237-Z</nc:RepositoryID>  
</nc:Metadata>

The example below shows metadata applied to the relationship between a person and a location. This example shows a celebrity’s birth location. The source for the birth location information is a Wikipedia page. The source information is not applicable to the entire person object, nor is it applicable to the entire location object. Is is only applicable to the *birth location* relationship between the person and the location.

Figure 12-14: A simple example of relationship metadata

<nc:Person>  
  <nc:PersonBirthLocation structures:relationshipMetadata="wp-ds">  
    <nc:Address>  
      <nc:AddressCityName>Bethesda</nc:AddressCityName>  
      <nc:AddressState>  
        <nc:StateName>Maryland</nc:StateName>  
      </nc:AddressState>  
    </nc:Address>  
  </nc:PersonBirthLocation>  
  <nc:PersonName>  
    <nc:PersonFullName>Daniel Stern</nc:PersonFullName>  
  </nc:PersonName>  
</nc:Person>  
<nc:Metadata structures:id="wp-ds">  
  <nc:SourceIDText>https://en.wikipedia.org/wiki/Daniel\_Stern\_(actor)</nc:SourceIDText>  
</nc:Metadata>

Characteristics of metadata include:

* Metadata objects may appear outside the data they describe. They applied by reference via the metadata attributes.
* Metadata objects may be applied to multiple objects within the same document.
* Each metadata attribute may hold references to multiple metadata objects, which enables messages to apply multiple metadata properties to objects and relationships.

An instance would not be valid XML if the structures:metadata or structures:relationshipMetadata attributes contained references for which there were no defined IDs. The instance would not be NIEM-conformant if the references were not to IDs defined with the structures:id attribute.

Application of metadata to a type or element to which it is not applicable is not NIEM-conformant. A metadata element may be labeled as applicable to multiple elements via attribute appinfo:appliesToElements, or to multiple types via attribute appinfo:appliesToTypes. In either case it may apply to an instance of any of the listed elements or types. For an example, see [Figure 10-14, *Sample use of appinfo:appliesToTypes*, above](#figure_10-14). A metadata element with neither attribute appinfo:appliesToElements nor attribute appinfo:appliesToTypes may be applied to any element or to an instance of any type.

Rule 12-11. Metadata applies to referring entity

[Rule 12-11] (INS) (Interpretation)

Within an element instance, when an object *$O* links to a metadata object via an attribute structures:metadata, the information in the metadata object MUST be applied to the object *$O*.

structures:metadata applies metadata to an object.

Rule 12-12. Referent of structures:relationshipMetadata annotates relationship

[Rule 12-12] (INS) (Interpretation)

Within an element instance, when an object *$O1* contains an element *$E*, with content object *$O2* or with a reference to object *$O2*, and *$O2* links to a metadata object via an attribute structures:relationshipMetadata, the information in the metadata object MUST be applied to the relationship *$E* between *$O1* and *$O2*.

structures:relationshipMetadata applies metadata to a relationship between two objects.

Rule 12-13. Values of structures:metadata refer to values of structures:id

[Rule 12-13] (INS) (Constraint)

Given that each IDREF in the value of an attribute structures:metadata must match the value of an ID attribute on some element in the XML document, that ID attribute MUST be an occurrence of the attribute structures:id.

Rule 12-14. Values of structures:relationshipMetadata refer to values of structures:id

[Rule 12-14] (INS) (Constraint)

Given that each IDREF in the value of an attribute structures:relationshipMetadata must match the value of an ID attribute on some element in the XML document, that ID attribute MUST be an occurrence of the attribute structures:id.

Rule 12-15. structures:metadata and structures:relationshipMetadata refer to metadata elements

[Rule 12-15] (INS) (Constraint)

Each element referenced by an attribute structures:metadata or an attribute structures:relationshipMetadata MUST have [element declaration] that is a [·metadata element declaration·](#definition_metadata_element_declaration).

Although not implemented in Schematron, this rule covers the cases not covered by [Rule 12-16, *Attribute structures:metadata references metadata element*, below](#rule_12-16).

Rule 12-16. Attribute structures:metadata references metadata element

[Rule 12-16] (INS) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[exists(@structures:metadata)]">  
    <sch:assert test="every $metadata-ref in tokenize(normalize-space(@structures:metadata), ' ') satisfies  
                        exists(//\*[exists(@structures:id [. = $metadata-ref])  
                                   and ends-with(local-name(), 'Metadata')])"  
      >Each item in the value of an attribute structures:metadata MUST appear as the value of an attribute structures:id with an owner element that is a metadata element.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Note that this will NOT diagnose a scenario in which the element with a name ending in “Metadata” is an external element; additional tests would be required to catch that.

Rule 12-17. Attribute structures:relationshipMetadata references metadata element

[Rule 12-17] (INS) (Constraint)

<sch:pattern>  
  <sch:rule context="\*[exists(@structures:relationshipMetadata)]">  
    <sch:assert test="every $metadata-ref in tokenize(normalize-space(@structures:relationshipMetadata), ' ') satisfies  
                        exists(//\*[exists(@structures:id [. = $metadata-ref])  
                                   and ends-with(local-name(), 'Metadata')])"  
      >Each item in the value of an attribute structures:relationshipMetadata MUST appear as the value of an attribute structures:id with an owner element that is a metadata element.</sch:assert>  
  </sch:rule>  
</sch:pattern>

Note that this will NOT diagnose a scenario in which the element with a name ending in “Metadata” is an external element; additional tests would be required to catch that.

Rule 12-18. Metadata is applicable to element

[Rule 12-18] (INS) (Constraint)

Given that an element *$SUBJECT-ELEMENT* uses a metadata element *$METADATA-ELEMENT* through a value in either an attribute structures:metadata or an attribute structures:relationshipMetadata, the element *$SUBJECT-ELEMENT* MUST be an applicable element for *$METADATA-ELEMENT*.

The applicable elements for a metadata element are identified by [Rule 10-41, *Metadata element has applicable elements*, above](#rule_10-41).

# Appendix A. References

**[BCP 14]**: Internet Engineering Task Force Best Current Practice 14. Available from <https://www.ietf.org/rfc/bcp/bcp14.txt>. BCP 14 is composed of:

**[RFC 2119]**: Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14, RFC 2119, March 1997. Available from <http://www.ietf.org/rfc/rfc2119.txt>.

**[RFC 8174]**: Leiba, B., “Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words”, BCP 14, RFC 8174, May 2017. Available from <https://www.ietf.org/rfc/rfc8174.txt>.

**[ClarkNS]**: Clark, J. “XML Namespaces”, 4 February 1999. Available from <http://www.jclark.com/xml/xmlns.htm>.

**[Code Lists]**: Webb Roberts. “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>.

**[ConfReq]**: Lynne Rosenthal, and Mark Skall, eds. “Conformance Requirements for Specifications v1.0.” The Organization for the Advancement of Structured Information Standards (OASIS), March 15, 2002. <https://www.oasis-open.org/committees/download.php/305/conformance_requirements-v1.pdf>.

**[CTAS]**: Roberts, Webb. “NIEM Conformance Targets Attribute Specification, Version 3.0.” NIEM Technical Architecture Committee, July 31, 2014. <http://reference.niem.gov/niem/specification/conformance-targets-attribute/3.0/NIEM-CTAS-3.0-2014-07-31.html>.

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

**[ISO 11179-5]**: “ISO/IEC 11179-5:2005, Information technology — Metadata registries (MDR) — Part 5: Naming and identification principles”. Available from <http://standards.iso.org/ittf/PubliclyAvailableStandards/c035347_ISO_IEC_11179-5_2005(E).zip>.

**[JLS]**: James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. “The Java Language Specification, Java SE 8 Edition.” Oracle Corp, March 3, 2014. <http://docs.oracle.com/javase/specs/jls/se8/html/>.

**[JSON LD]**: Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, and Niklas Lindstrom. “JSON-LD 1.0, A JSON-Based Serialization for Linked Data, W3C Recommendation.” Edited by Manu Sporny, Gregg Kellogg, and Markus Lanthaler. W3C, January 16, 2014. ``https://www.w3.org/TR/2014/REC-json-ld-20140116/.

**[N-ary]**: “Defining N-ary Relations on the Semantic Web”, W3C Working Group Note, 12 April 2006. Available from <http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412//>.

**[N-Quads]**: Gavin Carothers. “RDF 1.1 N-Quads.” W3C, February 25, 2014. <https://www.w3.org/TR/2014/REC-n-quads-20140225/>.

**[OED]**: “Oxford English Dictionary, Third Edition”, Oxford University Press, November 2010. <http://dictionary.oed.com/>.

**[RDF Concepts]**: Richard Cyganiak, David Wood, and Markus Lanthaler, eds. “RDF 1.1 Concepts and Abstract Syntax.” The World Wide Web Consortium (W3C), February 25, 2014. <http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>.

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

**[Schematron]**: “ISO/IEC STANDARD 19757-3: Information technology — Document Schema Definition Languages (DSDL) Part 3: Rule-based validation — Schematron”, ISO/IEC, 1 June 2006. Retrieved from <http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip>.

**[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 Base]**: Jonathan Marsh, and Richard Tobin, eds. “XML Base (Second Edition), W3C Recommendation.” W3C, January 28, 2009. Available from <http://www.w3.org/TR/2009/REC-xmlbase-20090128/>.

**[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 Datatypes]**: “XML Schema Part 2: Datatypes Second Edition”, W3C Recommendation, 28 October 2004. Available at <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>.

**[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/>.

**[XPath 2]**: Berglund, Anders, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon. “XML Path Language (XPath) 2.0 (Second Edition)”, W3C Recommendation, 3 January 2011. <http://www.w3.org/TR/2010/REC-xpath20-20101214/>.

# Appendix B. Structures namespace

<?xml version="1.0" encoding="UTF-8"?>  
<xs:schema  
  targetNamespace="http://release.niem.gov/niem/structures/5.0/"  
  version="5.0"  
  xml:lang="en-US"  
  xmlns:structures="http://release.niem.gov/niem/structures/5.0/"  
  xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  
  <xs:annotation>  
    <xs:documentation>The structures namespace provides base types and other components for definition of NIEM-conformant XML schemas.</xs:documentation>  
  </xs:annotation>  
  
  <xs:attribute name="id" type="xs:ID">  
    <xs:annotation>  
      <xs:documentation>A document-relative identifier for an XML element.</xs:documentation>  
    </xs:annotation>  
  </xs:attribute>  
  
  <xs:attribute name="ref" type="xs:IDREF">  
    <xs:annotation>  
      <xs:documentation>A document-relative reference to an XML element.</xs:documentation>  
    </xs:annotation>  
  </xs:attribute>  
  
  <xs:attribute name="uri" type="xs:anyURI">  
    <xs:annotation>  
      <xs:documentation>An internationalized resource identifier or uniform resource identifier for a node or object.</xs:documentation>  
    </xs:annotation>  
  </xs:attribute>  
  
  <xs:attribute name="metadata" type="xs:IDREFS">  
    <xs:annotation>  
      <xs:documentation>A list of metadata objects that apply to a node or object represented by an XML element.</xs:documentation>  
    </xs:annotation>  
  </xs:attribute>  
  
  <xs:attribute name="relationshipMetadata" type="xs:IDREFS">  
    <xs:annotation>  
      <xs:documentation>A list of metadata objects that apply to a relationship or property occurrence represented by an XML element.</xs:documentation>  
    </xs:annotation>  
  </xs:attribute>  
  
  <xs:attribute name="sequenceID" type="xs:positiveInteger">  
    <xs:annotation>  
      <xs:documentation>An identifier that establishes the relative order of a property occurrence among sibling properties of a node or object.</xs:documentation>  
    </xs:annotation>  
  </xs:attribute>  
  
  <xs:attributeGroup name="SimpleObjectAttributeGroup">  
    <xs:annotation>  
      <xs:documentation>A group of attributes that are applicable to objects, to be used when defining a complex type that is an extension of a simple type.</xs:documentation>  
    </xs:annotation>  
    <xs:attribute ref="structures:id"/>  
    <xs:attribute ref="structures:ref"/>  
    <xs:attribute ref="structures:uri"/>  
    <xs:attribute ref="structures:metadata"/>  
    <xs:attribute ref="structures:relationshipMetadata"/>  
    <xs:attribute ref="structures:sequenceID"/>  
    <xs:anyAttribute namespace="urn:us:gov:ic:ism urn:us:gov:ic:ntk" processContents="lax"/>  
  </xs:attributeGroup>  
  
  <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:id"/>  
    <xs:attribute ref="structures:ref"/>  
    <xs:attribute ref="structures:uri"/>  
    <xs:attribute ref="structures:metadata"/>  
    <xs:attribute ref="structures:relationshipMetadata"/>  
    <xs:attribute ref="structures:sequenceID"/>  
    <xs:anyAttribute namespace="urn:us:gov:ic:ism urn:us:gov:ic:ntk" processContents="lax"/>  
  </xs:complexType>  
  
  <xs:element name="ObjectAugmentationPoint" abstract="true">  
    <xs:annotation>  
      <xs:documentation>An augmentation point for type structures:ObjectType.</xs:documentation>  
    </xs:annotation>  
  </xs:element>  
  
  <xs:complexType name="AssociationType" abstract="true">  
    <xs:annotation>  
      <xs:documentation>A data type for a relationship between two or more objects, including any properties of that relationship.</xs:documentation>  
    </xs:annotation>  
    <xs:sequence>  
      <xs:element ref="structures:AssociationAugmentationPoint" minOccurs="0" maxOccurs="unbounded"/>  
    </xs:sequence>  
    <xs:attribute ref="structures:id"/>  
    <xs:attribute ref="structures:ref"/>  
    <xs:attribute ref="structures:uri"/>  
    <xs:attribute ref="structures:metadata"/>  
    <xs:attribute ref="structures:relationshipMetadata"/>  
    <xs:attribute ref="structures:sequenceID"/>  
    <xs:anyAttribute namespace="urn:us:gov:ic:ism urn:us:gov:ic:ntk" processContents="lax"/>  
  </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:complexType name="MetadataType" abstract="true">  
    <xs:annotation>  
      <xs:documentation>A data type for data about data.</xs:documentation>  
    </xs:annotation>  
    <xs:attribute ref="structures:id"/>  
    <xs:attribute ref="structures:ref"/>  
    <xs:attribute ref="structures:uri"/>  
    <xs:anyAttribute namespace="urn:us:gov:ic:ism urn:us:gov:ic:ntk" processContents="lax"/>  
  </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:attribute ref="structures:id"/>  
    <xs:attribute ref="structures:ref"/>  
    <xs:attribute ref="structures:uri"/>  
    <xs:anyAttribute namespace="urn:us:gov:ic:ism urn:us:gov:ic:ntk" processContents="lax"/>  
  </xs:complexType>  
  
</xs:schema>

# Appendix C. Appinfo namespace

<?xml version="1.0" encoding="UTF-8"?>  
<xs:schema  
  targetNamespace="http://release.niem.gov/niem/appinfo/5.0/"  
  version="5.0"  
  xml:lang="en-US"  
  xmlns:appinfo="http://release.niem.gov/niem/appinfo/5.0/"  
  xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  
  <xs:annotation>  
    <xs:documentation>The appinfo schema provides support for high level data model concepts and additional syntax to support the NIEM conceptual model and validation of NIEM-conformant instances.</xs:documentation>  
  </xs:annotation>  
  
  <xs:attribute name="deprecated">  
    <xs:annotation>  
      <xs:documentation>The deprecated attribute provides a method for identifying schema components as being deprecated. A deprecated component is one that is provided, but the use of which is not recommended.</xs:documentation>  
    </xs:annotation>  
    <xs:simpleType>  
      <xs:restriction base="xs:boolean">  
        <xs:pattern value="true"/>  
      </xs:restriction>  
    </xs:simpleType>  
  </xs:attribute>  
  
  <xs:attribute name="appliesToTypes">  
    <xs:annotation>  
      <xs:documentation>The appliesToTypes attribute appears on the element declaration of a metadata element. It indicates a set of types to which the metadata element may be applied. The metadata element will also be applicable to any type that is derived from a listed type.</xs:documentation>  
    </xs:annotation>  
    <xs:simpleType>  
      <xs:list itemType="xs:QName"/>  
    </xs:simpleType>  
  </xs:attribute>  
  
  <xs:attribute name="appliesToElements">  
    <xs:annotation>  
      <xs:documentation>The appliesToElements attribute appears on the element declaration of a metadata element. It indicates a set of elements to which the metadata element may be applied. The metadata element will also be applicable to any element that is in the substitution group of a listed element.</xs:documentation>  
    </xs:annotation>  
    <xs:simpleType>  
      <xs:list itemType="xs:QName"/>  
    </xs:simpleType>  
  </xs:attribute>  
  
  <xs:attribute name="externalAdapterTypeIndicator">  
    <xs:annotation>  
      <xs:documentation>The externalAdapterTypeIndicator attribute indicates that a complex type is an external adapter type. An external adapter type is composed of elements and attributes from non-NIEM-conformant schemas.</xs:documentation>  
    </xs:annotation>  
    <xs:simpleType>  
      <xs:restriction base="xs:boolean">  
        <xs:pattern value="true"/>  
      </xs:restriction>  
    </xs:simpleType>  
  </xs:attribute>  
  
  <xs:attribute name="externalImportIndicator">  
    <xs:annotation>  
      <xs:documentation>The externalImportIndicator attribute is true if and only if a namespace identified via xs:import is expected to be non-conformant.</xs:documentation>  
    </xs:annotation>  
    <xs:simpleType>  
      <xs:restriction base="xs:boolean">  
        <xs:pattern value="true"/>  
      </xs:restriction>  
    </xs:simpleType>  
  </xs:attribute>  
  
  <xs:element name="LocalTerm">  
    <xs:complexType>  
      <xs:sequence>  
  <xs:element name="SourceText" type="appinfo:NonemptyStringSimpleType" minOccurs="0" maxOccurs="unbounded" form="qualified"/>  
      </xs:sequence>  
      <xs:attribute name="term" type="appinfo:NonemptyStringSimpleType" use="required"/>  
      <xs:attribute name="literal" type="appinfo:NonemptyStringSimpleType"/>  
      <xs:attribute name="definition" type="appinfo:NonemptyStringSimpleType"/>  
      <xs:attribute name="sourceURIs">  
        <xs:simpleType>  
          <xs:restriction>  
            <xs:simpleType>  
              <xs:list itemType="xs:anyURI"/>  
            </xs:simpleType>  
            <xs:minLength value="1"/>  
          </xs:restriction>  
        </xs:simpleType>  
      </xs:attribute>  
    </xs:complexType>  
  </xs:element>  
  
  <xs:simpleType name="NonemptyStringSimpleType">  
    <xs:restriction base="xs:string">  
      <xs:minLength value="1"/>  
    </xs:restriction>  
  </xs:simpleType>  
  
</xs:schema>

# Appendix D. Index of definitions

* [appinfo namespace](#definition_appinfo_namespace): [Section 10.9.1, *The NIEM appinfo namespace*](#section_10.9.1)
* [application information](#definition_application_information): [Section 10.9, *Machine-readable annotations*](#section_10.9)
* [association](#definition_association): [Section 10.3, *Associations*](#section_10.3)
* [association element declaration](#X0800fc6c42d99fd31cd824ea485f2a4b7ee49f3): [Section 10.3, *Associations*](#section_10.3)
* [association type](#definition_association_type): [Section 10.3, *Associations*](#section_10.3)
* [attribute](#definition_attribute): [Section 3.3, *XML Information Set terminology*](#section_3.3)
* [augmentable type](#definition_augmentable_type): [Section 10.4.1, *Augmentable types*](#section_10.4.1)
* [augmentation](#definition_augmentation): [Section 10.4, *Augmentations*](#section_10.4)
* [augmentation element declaration](#Xfaa24539372ac2c23b1e89a5bfc415f590cea1e): [Section 10.4.4, *Augmentation types*](#section_10.4.4)
* [augmentation type](#definition_augmentation_type): [Section 10.4.4, *Augmentation types*](#section_10.4.4)
* [base type definition](#definition_base_type_definition): [Section 3.4.1, *Schema components*](#section_3.4.1)
* [code simple type](#definition_code_simple_type): [Section 11.1.2.3, *Code simple types*](#section_11.1.2.3)
* [code type](#definition_code_type): [Section 10.2.4, *Code types*](#section_10.2.4)
* [complex type definition](#definition_complex_type_definition): [Section 3.4.1, *Schema components*](#section_3.4.1)
* [conformance target](#definition_conformance_target): [Section 3.6, *Conformance Targets Attribute Specification terminology*](#section_3.6)
* [conformance target identifier](#definition_conformance_target_identifier): [Section 3.6, *Conformance Targets Attribute Specification terminology*](#section_3.6)
* [conformant element information item](#Xb8c2328714130974c99941a242199ecbb7daea7): [Section 4.1.4, *Instance documents and elements*](#section_4.1.4)
* [conformant instance XML document](#X6a4ab7051f7203e0b47588881449d56367bed27): [Section 4.1.4, *Instance documents and elements*](#section_4.1.4)
* [conformant schema document set](#Xeff662d8d67f5b4e72a2955d729e0ac83cad2d6): [Section 4.1.3, *Schema document set*](#section_4.1.3)
* [constraint rule](#definition_constraint_rule): [Section 2.4.1, *Rules*](#section_2.4.1)
* [data definition](#definition_data_definition): [Section 7.4, *ISO 11179 Part 4*](#section_7.4)
* [deprecated component](#definition_deprecated_component): [Section 10.9.1.1, *Deprecation*](#section_10.9.1.1)
* [document element](#definition_document_element): [Section 3.3, *XML Information Set terminology*](#section_3.3)
* [documented component](#definition_documented_component): [Section 7.4, *ISO 11179 Part 4*](#section_7.4)
* [effective conformance target identifier](#X6d0b4369562c0703761463ba00cd0e2b474de22): [Section 3.6, *Conformance Targets Attribute Specification terminology*](#section_3.6)
* [element](#definition_element): [Section 3.3, *XML Information Set terminology*](#section_3.3)
* [element declaration](#definition_element_declaration): [Section 3.4.1, *Schema components*](#section_3.4.1)
* [extension schema document](#definition_extension_schema_document): [Section 4.1.2, *Extension schema document*](#section_4.1.2)
* [external adapter type](#definition_external_adapter_type): [Section 10.2.3.2, *External adapter types*](#section_10.2.3.2)
* [external schema document](#definition_external_schema_document): [Section 10.2.3, *External adapter types and external components*](#section_10.2.3)
* [informative](#definition_informative): [Section 2.4, *Normative and informative content*](#section_2.4)
* [instance document](#definition_instance_document): [Section 3.4, *XML Schema terminology*](#section_3.4)
* [interpretation rule](#definition_interpretation_rule): [Section 2.4.1, *Rules*](#section_2.4.1)
* [local term](#definition_local_term): [Section 10.8.2.1, *Use of Acronyms, Initialisms, Abbreviations, and Jargon*](#section_10.8.2.1)
* [machine-readable annotation](#definition_machine-readable_annotation): [Section 10.9, *Machine-readable annotations*](#section_10.9)
* [metadata element declaration](#definition_metadata_element_declaration): [Section 10.5.2, *Metadata element declarations*](#section_10.5.2)
* [metadata type](#definition_metadata_type): [Section 10.5.1, *Metadata types*](#section_10.5.1)
* [normative](#definition_normative): [Section 2.4, *Normative and informative content*](#section_2.4)
* [object type](#definition_object_type): [Section 10.2.1, *General object types*](#section_10.2.1)
* [reference schema document](#definition_reference_schema_document): [Section 4.1.1, *Reference schema document*](#section_4.1.1)
* [representation element declaration](#X1bc7717493f0a423581c67a1dc9df1fdfdc9316): [Section 10.7, *The “Representation” pattern*](#section_10.7)
* [role type](#definition_role_type): [Section 10.2.2, *Role types and roles*](#section_10.2.2)
* [RoleOf element](#definition_RoleOf_element): [Section 10.2.2, *Role types and roles*](#section_10.2.2)
* [schema component](#definition_schema_component): [Section 3.4, *XML Schema terminology*](#section_3.4)
* [schema document](#definition_schema_document): [Section 3.4, *XML Schema terminology*](#section_3.4)
* [simple type definition](#definition_simple_type_definition): [Section 3.4.1, *Schema components*](#section_3.4.1)
* [structures namespace](#definition_structures_namespace): [Section 10.10, *NIEM structural facilities*](#section_10.10)
* [valid](#definition_valid): [Section 3.4, *XML Schema terminology*](#section_3.4)
* [XML document](#definition_XML_document): [Section 3.2, *XML terminology*](#section_3.2)
* [XML Schema](#definition_XML_Schema): [Section 3.4, *XML Schema terminology*](#section_3.4)
* [XML Schema definition language](#X7e4888920c30d9bbda28173a5b8b96fa0c14a07): [Section 3.4, *XML Schema terminology*](#section_3.4)
* [XML Schema document set](#definition_XML_Schema_document_set): [Section 3.4, *XML Schema terminology*](#section_3.4)

# Appendix E. Index of rules

* [Rule 4-1, *Schema marked as reference schema document must conform*](#rule_4-1): [Section 4.1, *Conformance targets defined*](#section_4.1)
* [Rule 4-2, *Schema marked as extension schema document must conform*](#rule_4-2): [Section 4.1, *Conformance targets defined*](#section_4.1)
* [Rule 4-3, *Schema is CTAS-conformant*](#rule_4-3): [Section 4.3, *Conformance target identifiers*](#section_4.3)
* [Rule 4-4, *Document element has attribute ct:conformanceTargets*](#rule_4-4): [Section 4.3, *Conformance target identifiers*](#section_4.3)
* [Rule 4-5, *Schema claims reference schema conformance target*](#rule_4-5): [Section 4.3, *Conformance target identifiers*](#section_4.3)
* [Rule 4-6, *Schema claims extension conformance target*](#rule_4-6): [Section 4.3, *Conformance target identifiers*](#section_4.3)
* [Rule 5-1, *structures:uri denotes resource identifier*](#rule_5-1): [Section 5.1, *Purpose of the NIEM conceptual model*](#section_5.1)
* [Rule 7-1, *Document is an XML document*](#rule_7-1): [Section 7.1, *Conformance to XML*](#section_7.1)
* [Rule 7-2, *Document uses XML namespaces properly*](#rule_7-2): [Section 7.2, *Conformance to XML Namespaces*](#section_7.2)
* [Rule 7-3, *Document is a schema document*](#rule_7-3): [Section 7.3, *Conformance to XML Schema*](#section_7.3)
* [Rule 7-4, *Document element is xs:schema*](#rule_7-4): [Section 7.3, *Conformance to XML Schema*](#section_7.3)
* [Rule 7-5, *Component name follows ISO 11179 Part 5 Annex A*](#rule_7-5): [Section 7.5, *ISO 11179 Part 5*](#section_7.5)
* [Rule 9-1, *No base type in the XML namespace*](#rule_9-1): [Section 9.1.1.1, *Types prohibited as base types*](#section_9.1.1.1)
* [Rule 9-2, *No base type of xs:ID*](#rule_9-2): [Section 9.1.1.1, *Types prohibited as base types*](#section_9.1.1.1)
* [Rule 9-3, *No base type of xs:IDREF*](#rule_9-3): [Section 9.1.1.1, *Types prohibited as base types*](#section_9.1.1.1)
* [Rule 9-4, *No base type of xs:IDREFS*](#rule_9-4): [Section 9.1.1.1, *Types prohibited as base types*](#section_9.1.1.1)
* [Rule 9-5, *No base type of xs:anyType*](#rule_9-5): [Section 9.1.1.1, *Types prohibited as base types*](#section_9.1.1.1)
* [Rule 9-6, *No base type of xs:anySimpleType*](#rule_9-6): [Section 9.1.1.1, *Types prohibited as base types*](#section_9.1.1.1)
* [Rule 9-7, *No base type of xs:NOTATION*](#rule_9-7): [Section 9.1.1.1, *Types prohibited as base types*](#section_9.1.1.1)
* [Rule 9-8, *No base type of xs:ENTITY*](#rule_9-8): [Section 9.1.1.1, *Types prohibited as base types*](#section_9.1.1.1)
* [Rule 9-9, *No base type of xs:ENTITIES*](#rule_9-9): [Section 9.1.1.1, *Types prohibited as base types*](#section_9.1.1.1)
* [Rule 9-10, *Simple type definition is top-level*](#rule_9-10): [Section 9.1.2, *Simple type definition*](#section_9.1.2)
* [Rule 9-11, *No simple type disallowed derivation*](#rule_9-11): [Section 9.1.2, *Simple type definition*](#section_9.1.2)
* [Rule 9-12, *Simple type has data definition*](#rule_9-12): [Section 9.1.2, *Simple type definition*](#section_9.1.2)
* [Rule 9-13, *No use of “fixed” on simple type facets*](#rule_9-13): [Section 9.1.2, *Simple type definition*](#section_9.1.2)
* [Rule 9-14, *Enumeration has data definition*](#rule_9-14): [Section 9.1.2, *Simple type definition*](#section_9.1.2)
* [Rule 9-15, *No list item type of xs:ID*](#rule_9-15): [Section 9.1.2.1, *Simple types prohibited as list item types*](#section_9.1.2.1)
* [Rule 9-16, *No list item type of xs:IDREF*](#rule_9-16): [Section 9.1.2.1, *Simple types prohibited as list item types*](#section_9.1.2.1)
* [Rule 9-17, *No list item type of xs:anySimpleType*](#rule_9-17): [Section 9.1.2.1, *Simple types prohibited as list item types*](#section_9.1.2.1)
* [Rule 9-18, *No list item type of xs:ENTITY*](#rule_9-18): [Section 9.1.2.1, *Simple types prohibited as list item types*](#section_9.1.2.1)
* [Rule 9-19, *No union member types of xs:ID*](#rule_9-19): [Section 9.1.2.2, *Simple types prohibited as union member types*](#section_9.1.2.2)
* [Rule 9-20, *No union member types of xs:IDREF*](#rule_9-20): [Section 9.1.2.2, *Simple types prohibited as union member types*](#section_9.1.2.2)
* [Rule 9-21, *No union member types of xs:IDREFS*](#rule_9-21): [Section 9.1.2.2, *Simple types prohibited as union member types*](#section_9.1.2.2)
* [Rule 9-22, *No union member types of xs:anySimpleType*](#rule_9-22): [Section 9.1.2.2, *Simple types prohibited as union member types*](#section_9.1.2.2)
* [Rule 9-23, *No union member types of xs:ENTITY*](#rule_9-23): [Section 9.1.2.2, *Simple types prohibited as union member types*](#section_9.1.2.2)
* [Rule 9-24, *No union member types of xs:ENTITIES*](#rule_9-24): [Section 9.1.2.2, *Simple types prohibited as union member types*](#section_9.1.2.2)
* [Rule 9-25, *Complex type definition is top-level*](#rule_9-25): [Section 9.1.3, *Complex type definition*](#section_9.1.3)
* [Rule 9-26, *Complex type has data definition*](#rule_9-26): [Section 9.1.3, *Complex type definition*](#section_9.1.3)
* [Rule 9-27, *No mixed content on complex type*](#rule_9-27): [Section 9.1.3.1, *No mixed content*](#section_9.1.3.1)
* [Rule 9-28, *No mixed content on complex content*](#rule_9-28): [Section 9.1.3.1, *No mixed content*](#section_9.1.3.1)
* [Rule 9-29, *Complex type content is explicitly simple or complex*](#rule_9-29): [Section 9.1.3, *Complex type definition*](#section_9.1.3)
* [Rule 9-30, *Complex content uses extension*](#rule_9-30): [Section 9.1.3.2, *Complex content*](#section_9.1.3.2)
* [Rule 9-31, *Base type of complex type with complex content must have complex content*](#rule_9-31): [Section 9.1.3.2.1, *Base type of complex type with complex content has complex content*](#section_9.1.3.2.1)
* [Rule 9-32, *Base type of complex type with complex content must have complex content*](#rule_9-32): [Section 9.1.3.2.1, *Base type of complex type with complex content has complex content*](#section_9.1.3.2.1)
* [Rule 9-33, *Simple content uses extension*](#rule_9-33): [Section 9.1.3.3, *Simple content*](#section_9.1.3.3)
* [Rule 9-34, *No complex type disallowed substitutions*](#rule_9-34): [Section 9.1.3, *Complex type definition*](#section_9.1.3)
* [Rule 9-35, *No complex type disallowed derivation*](#rule_9-35): [Section 9.1.3, *Complex type definition*](#section_9.1.3)
* [Rule 9-36, *Element declaration is top-level*](#rule_9-36): [Section 9.2.1, *Element declaration*](#section_9.2.1)
* [Rule 9-37, *Element declaration has data definition*](#rule_9-37): [Section 9.2.1, *Element declaration*](#section_9.2.1)
* [Rule 9-38, *Untyped element is abstract*](#rule_9-38): [Section 9.2.1, *Element declaration*](#section_9.2.1)
* [Rule 9-39, *Element of type xs:anySimpleType is abstract*](#rule_9-39): [Section 9.2.1, *Element declaration*](#section_9.2.1)
* [Rule 9-40, *Element type not in the XML Schema namespace*](#rule_9-40): [Section 9.2.1, *Element declaration*](#section_9.2.1)
* [Rule 9-41, *Element type not in the XML namespace*](#rule_9-41): [Section 9.2.1, *Element declaration*](#section_9.2.1)
* [Rule 9-42, *Element type is not simple type*](#rule_9-42): [Section 9.2.1, *Element declaration*](#section_9.2.1)
* [Rule 9-43, *No element disallowed substitutions*](#rule_9-43) : [Section 9.2.1, *Element declaration*](#section_9.2.1)
* [Rule 9-44, *No element disallowed derivation*](#rule_9-44): [Section 9.2.1, *Element declaration*](#section_9.2.1)
* [Rule 9-45, *No element default value*](#rule_9-45): [Section 9.2.1.1, *No element value constraints*](#section_9.2.1.1)
* [Rule 9-46, *No element fixed value*](#rule_9-46): [Section 9.2.1.1, *No element value constraints*](#section_9.2.1.1)
* [Rule 9-47, *Element declaration is nillable*](#rule_9-47): [Section 9.2.1, *Element declaration*](#section_9.2.1)
* [Rule 9-48, *Attribute declaration is top-level*](#rule_9-48): [Section 9.2.3, *Attribute declaration*](#section_9.2.3)
* [Rule 9-49, *Attribute declaration has data definition*](#rule_9-49): [Section 9.2.3, *Attribute declaration*](#section_9.2.3)
* [Rule 9-50, *Attribute declaration has type*](#rule_9-50): [Section 9.2.3, *Attribute declaration*](#section_9.2.3)
* [Rule 9-51, *No attribute type of xs:ID*](#rule_9-51): [Section 9.2.3.1, *Prohibited attribute types*](#section_9.2.3.1)
* [Rule 9-52, *No attribute type of xs:IDREF*](#rule_9-52): [Section 9.2.3.1, *Prohibited attribute types*](#section_9.2.3.1)
* [Rule 9-53, *No attribute type of xs:IDREFS*](#rule_9-53): [Section 9.2.3.1, *Prohibited attribute types*](#section_9.2.3.1)
* [Rule 9-54, *No attribute type of xs:ENTITY*](#rule_9-54): [Section 9.2.3.1, *Prohibited attribute types*](#section_9.2.3.1)
* [Rule 9-55, *No attribute type of xs:ENTITIES*](#rule_9-55): [Section 9.2.3.1, *Prohibited attribute types*](#section_9.2.3.1)
* [Rule 9-56, *No attribute type of xs:anySimpleType*](#rule_9-56): [Section 9.2.3.1, *Prohibited attribute types*](#section_9.2.3.1)
* [Rule 9-57, *No attribute default values*](#rule_9-57): [Section 9.2.3.2, *No attribute value constraints*](#section_9.2.3.2)
* [Rule 9-58, *No fixed values for optional attributes*](#rule_9-58): [Section 9.2.3.2, *No attribute value constraints*](#section_9.2.3.2)
* [Rule 9-59, *No use of element xs:notation*](#rule_9-59): [Section 9.2.4, *Notation declaration*](#section_9.2.4)
* [Rule 9-60, *Model group does not affect meaning*](#rule_9-60): [Section 9.3.1, *Model group*](#section_9.3.1)
* [Rule 9-61, *No xs:all*](#rule_9-61): [Section 9.3.1, *Model group*](#section_9.3.1)
* [Rule 9-62, *xs:sequence must be child of xs:extension*](#rule_9-62): [Section 9.3.1.1, *Sequence*](#section_9.3.1.1)
* [Rule 9-63, *xs:sequence must be child of xs:extension or xs:restriction*](#rule_9-63): [Section 9.3.1.1, *Sequence*](#section_9.3.1.1)
* [Rule 9-64, *No xs:choice*](#rule_9-64): [Section 9.3.1.2, *Choice*](#section_9.3.1.2)
* [Rule 9-65, *xs:choice must be child of xs:sequence*](#rule_9-65): [Section 9.3.1.2, *Choice*](#section_9.3.1.2)
* [Rule 9-66, *Sequence has minimum cardinality 1*](#rule_9-66): [Section 9.3.2.1, *Sequence cardinality*](#section_9.3.2.1)
* [Rule 9-67, *Sequence has maximum cardinality 1*](#rule_9-67): [Section 9.3.2.1, *Sequence cardinality*](#section_9.3.2.1)
* [Rule 9-68, *Choice has minimum cardinality 1*](#rule_9-68): [Section 9.3.2.2, *Choice cardinality*](#section_9.3.2.2)
* [Rule 9-69, *Choice has maximum cardinality 1*](#rule_9-69): [Section 9.3.2.2, *Choice cardinality*](#section_9.3.2.2)
* [Rule 9-70, *No use of xs:any*](#rule_9-70): [Section 9.3.4, *Wildcard*](#section_9.3.4)
* [Rule 9-71, *No use of xs:anyAttribute*](#rule_9-71): [Section 9.3.4, *Wildcard*](#section_9.3.4)
* [Rule 9-72, *No use of xs:unique*](#rule_9-72): [Section 9.4, *Identity-constraint definition components*](#section_9.4)
* [Rule 9-73, *No use of xs:key*](#rule_9-73): [Section 9.4, *Identity-constraint definition components*](#section_9.4)
* [Rule 9-74, *No use of xs:keyref*](#rule_9-74): [Section 9.4, *Identity-constraint definition components*](#section_9.4)
* [Rule 9-75, *No use of xs:group*](#rule_9-75): [Section 9.5.1, *Model group definition*](#section_9.5.1)
* [Rule 9-76, *No definition of attribute groups*](#rule_9-76): [Section 9.5.2, *Attribute group definition*](#section_9.5.2)
* [Rule 9-77, *Comment is not recommended*](#rule_9-77): [Section 9.6, *Annotation components*](#section_9.6)
* [Rule 9-78, *Documentation element has no element children*](#rule_9-78): [Section 9.6, *Annotation components*](#section_9.6)
* [Rule 9-79, *xs:appinfo children are comments, elements, or whitespace*](#rule_9-79): [Section 9.6.1, *Application information annotation*](#section_9.6.1)
* [Rule 9-80, *Appinfo child elements have namespaces*](#rule_9-80): [Section 9.6.1, *Application information annotation*](#section_9.6.1)
* [Rule 9-81, *Appinfo descendants are not XML Schema elements*](#rule_9-81): [Section 9.6.1, *Application information annotation*](#section_9.6.1)
* [Rule 9-82, *Schema has data definition*](#rule_9-82): [Section 9.7, *Schema as a whole*](#section_9.7)
* [Rule 9-83, *Schema document defines target namespace*](#rule_9-83): [Section 9.7, *Schema as a whole*](#section_9.7)
* [Rule 9-84, *Target namespace is absolute URI*](#rule_9-84): [Section 9.7, *Schema as a whole*](#section_9.7)
* [Rule 9-85, *Schema has version*](#rule_9-85): [Section 9.7, *Schema as a whole*](#section_9.7)
* [Rule 9-86, *No disallowed substitutions*](#rule_9-86): [Section 9.7, *Schema as a whole*](#section_9.7)
* [Rule 9-87, *No disallowed derivations*](#rule_9-87): [Section 9.7, *Schema as a whole*](#section_9.7)
* [Rule 9-88, *No use of xs:redefine*](#rule_9-88): [Section 9.8, *Schema assembly*](#section_9.8)
* [Rule 9-89, *No use of xs:include*](#rule_9-89): [Section 9.8, *Schema assembly*](#section_9.8)
* [Rule 9-90, *xs:import must have namespace*](#rule_9-90): [Section 9.8, *Schema assembly*](#section_9.8)
* [Rule 9-91, *XML Schema document set must be complete*](#rule_9-91): [Section 9.8, *Schema assembly*](#section_9.8)
* [Rule 9-92, *Namespace referenced by attribute type is imported*](#rule_9-92): [Section 9.8.1, *Namespaces for referenced components are imported*](#section_9.8.1)
* [Rule 9-93, *Namespace referenced by attribute base is imported*](#rule_9-93): [Section 9.8.1, *Namespaces for referenced components are imported*](#section_9.8.1)
* [Rule 9-94, *Namespace referenced by attribute itemType is imported*](#rule_9-94): [Section 9.8.1, *Namespaces for referenced components are imported*](#section_9.8.1)
* [Rule 9-95, *Namespaces referenced by attribute memberTypes is imported*](#rule_9-95): [Section 9.8.1, *Namespaces for referenced components are imported*](#section_9.8.1)
* [Rule 9-96, *Namespace referenced by attribute ref is imported*](#rule_9-96): [Section 9.8.1, *Namespaces for referenced components are imported*](#section_9.8.1)
* [Rule 9-97, *Namespace referenced by attribute substitutionGroup is imported*](#rule_9-97): [Section 9.8.1, *Namespaces for referenced components are imported*](#section_9.8.1)
* [Rule 10-1, *Complex type has a category*](#rule_10-1): [Section 10.1, *Categories of NIEM type definitions*](#section_10.1)
* [Rule 10-2, *Object type with complex content is derived from structures:ObjectType*](#rule_10-2): [Section 10.2.1.1, *Object types with complex content*](#section_10.2.1.1)
* [Rule 10-3, *RoleOf element type is an object type*](#rule_10-3): [Section 10.2.2, *Role types and roles*](#section_10.2.2)
* [Rule 10-4, *Only object type has RoleOf element*](#rule_10-4): [Section 10.2.2, *Role types and roles*](#section_10.2.2)
* [Rule 10-5, *RoleOf elements indicate the base types of a role type*](#rule_10-5): [Section 10.2.2, *Role types and roles*](#section_10.2.2)
* [Rule 10-6, *Instance of RoleOf element indicates a role object*](#rule_10-6): [Section 10.2.2, *Role types and roles*](#section_10.2.2)
* [Rule 10-7, *Import of external namespace has data definition*](#rule_10-7): [Section 10.2.3.1, *Import of external namespace*](#section_10.2.3.1)
* [Rule 10-8, *External adapter type has indicator*](#rule_10-8): [Section 10.2.3.2, *External adapter types*](#section_10.2.3.2)
* [Rule 10-9, *Structure of external adapter type definition follows pattern*](#rule_10-9): [Section 10.2.3.2, *External adapter types*](#section_10.2.3.2)
* [Rule 10-10, *Element use from external adapter type defined by external schema documents*](#rule_10-10): [Section 10.2.3.2, *External adapter types*](#section_10.2.3.2)
* [Rule 10-11, *External adapter type not a base type*](#rule_10-11): [Section 10.2.3.2, *External adapter types*](#section_10.2.3.2)
* [Rule 10-12, *External adapter type not a base type*](#rule_10-12): [Section 10.2.3.2, *External adapter types*](#section_10.2.3.2)
* [Rule 10-13, *External attribute use only in external adapter type*](#rule_10-13): [Section 10.2.3.3, *External attribute use*](#section_10.2.3.3)
* [Rule 10-14, *External attribute use has data definition*](#rule_10-14): [Section 10.2.3.3, *External attribute use*](#section_10.2.3.3)
* [Rule 10-15, *External attribute use not an ID*](#rule_10-15): [Section 10.2.3.3, *External attribute use*](#section_10.2.3.3)
* [Rule 10-16, *External element use has data definition*](#rule_10-16): [Section 10.2.3.4, *External element use*](#section_10.2.3.4)
* [Rule 10-17, *Name of code type ends in “CodeType”*](#rule_10-17): [Section 10.2.4, *Code types*](#section_10.2.4)
* [Rule 10-18, *Code type corresponds to a code list*](#rule_10-18): [Section 10.2.4, *Code types*](#section_10.2.4)
* [Rule 10-19, *Element of code type has code representation term*](#rule_10-19): [Section 10.2.4, *Code types*](#section_10.2.4)
* [Rule 10-20, *Proxy type has designated structure*](#rule_10-20): [Section 10.2.5, *Proxy types*](#section_10.2.5)
* [Rule 10-21, *Association type derived from structures:AssociationType*](#rule_10-21): [Section 10.3.1, *Association types*](#section_10.3.1)
* [Rule 10-22, *Association element type is an association type*](#rule_10-22): [Section 10.3.2, *Association element declarations*](#section_10.3.2)
* [Rule 10-23, *Augmentable type has augmentation point element*](#rule_10-23): [Section 10.4.1, *Augmentable types*](#section_10.4.1)
* [Rule 10-24, *Augmentable type has at most one augmentation point element*](#rule_10-24): [Section 10.4.1, *Augmentable types*](#section_10.4.1)
* [Rule 10-25, *Augmentation point element corresponds to its base type*](#rule_10-25): [Section 10.4.2, *Augmentation point element declarations*](#section_10.4.2)
* [Rule 10-26, *An augmentation point element has no type*](#rule_10-26): [Section 10.4.2, *Augmentation point element declarations*](#section_10.4.2)
* [Rule 10-27, *An augmentation point element has no substitution group*](#rule_10-27): [Section 10.4.2, *Augmentation point element declarations*](#section_10.4.2)
* [Rule 10-28, *Augmentation point element is only referenced by its base type*](#rule_10-28): [Section 10.4.3, *Augmentation point element use*](#section_10.4.3)
* [Rule 10-29, *Augmentation point element use is optional*](#rule_10-29): [Section 10.4.3, *Augmentation point element use*](#section_10.4.3)
* [Rule 10-30, *Augmentation point element use is unbounded*](#rule_10-30): [Section 10.4.3, *Augmentation point element use*](#section_10.4.3)
* [Rule 10-31, *Augmentation point element use must be last element in its base type*](#rule_10-31): [Section 10.4.3, *Augmentation point element use*](#section_10.4.3)
* [Rule 10-32, *Element within instance of augmentation type modifies base*](#rule_10-32): [Section 10.4.4, *Augmentation types*](#section_10.4.4)
* [Rule 10-33, *Only an augmentation type name ends in “AugmentationType”*](#rule_10-33): [Section 10.4.4, *Augmentation types*](#section_10.4.4)
* [Rule 10-34, *Schema component with name ending in “AugmentationType” is an augmentation type*](#rule_10-34): [Section 10.4.4, *Augmentation types*](#section_10.4.4)
* [Rule 10-35, *Type derived from structures:AugmentationType is an augmentation type*](#rule_10-35): [Section 10.4.4, *Augmentation types*](#section_10.4.4)
* [Rule 10-36, *Augmentation element type is an augmentation type*](#rule_10-36): [Section 10.4.5, *Augmentation element declarations*](#section_10.4.5)
* [Rule 10-37, *Augmentation elements are not used directly*](#rule_10-37): [Section 10.4.5, *Augmentation element declarations*](#section_10.4.5)
* [Rule 10-38, *Metadata type has data about data*](#rule_10-38): [Section 10.5.1, *Metadata types*](#section_10.5.1)
* [Rule 10-39, *Metadata types are derived from structures:MetadataType*](#rule_10-39): [Section 10.5.1, *Metadata types*](#section_10.5.1)
* [Rule 10-40, *Metadata element declaration type is a metadata type*](#rule_10-40): [Section 10.5.2, *Metadata element declarations*](#section_10.5.2)
* [Rule 10-41, *Metadata element has applicable elements*](#rule_10-41): [Section 10.5.2, *Metadata element declarations*](#section_10.5.2)
* [Rule 10-42, *Name of element that ends in “Representation” is abstract*](#rule_10-42): [Section 10.7, *The “Representation” pattern*](#section_10.7)
* [Rule 10-43, *A substitution for a representation element declaration is a value for a type*](#rule_10-43): [Section 10, *Rules for NIEM modeling, by NIEM concept*](#section_10)
* [Rule 10-44, *Schema component name composed of English words*](#rule_10-44): [Section 10.8, *Naming rules*](#section_10.8)
* [Rule 10-45, *Schema component name has xml:lang*](#rule_10-45): [Section 10.8, *Naming rules*](#section_10.8)
* [Rule 10-46, *Schema component names have only specific characters*](#rule_10-46): [Section 10.8, *Naming rules*](#section_10.8)
* [Rule 10-47, *Punctuation in component name is a separator*](#rule_10-47): [Section 10.8, *Naming rules*](#section_10.8)
* [Rule 10-48, *Names use camel case*](#rule_10-48): [Section 10.8.1, *Character case*](#section_10.8.1)
* [Rule 10-49, *Attribute name begins with lower case letter*](#rule_10-49): [Section 10.8.1, *Character case*](#section_10.8.1)
* [Rule 10-50, *Name of schema component other than attribute and proxy type begins with upper case letter*](#rule_10-50): [Section 10.8.1, *Character case*](#section_10.8.1)
* [Rule 10-51, *Names use common abbreviations*](#rule_10-51): [Section 10.8.2, *Use of acronyms and abbreviations*](#section_10.8.2)
* [Rule 10-52, *Local term declaration is local to its schema document*](#rule_10-52): [Section 10.8.2.1, *Use of Acronyms, Initialisms, Abbreviations, and Jargon*](#section_10.8.2.1)
* [Rule 10-53, *Local terminology interpretation*](#rule_10-53): [Section 10.8.2.1, *Use of Acronyms, Initialisms, Abbreviations, and Jargon*](#section_10.8.2.1)
* [Rule 10-54, *Singular form is preferred in name*](#rule_10-54): [Section 10.8.3, *Word forms*](#section_10.8.3)
* [Rule 10-55, *Present tense is preferred in name*](#rule_10-55): [Section 10.8.3, *Word forms*](#section_10.8.3)
* [Rule 10-56, *Name does not have nonessential words*](#rule_10-56): [Section 10.8.3, *Word forms*](#section_10.8.3)
* [Rule 10-57, *Element or attribute name follows pattern*](#rule_10-57): [Section 10.8, *Naming rules*](#section_10.8)
* [Rule 10-58, *Object-class term identifies concrete category*](#rule_10-58): [Section 10.8.4, *Object-class term*](#section_10.8.4)
* [Rule 10-59, *Property term describes characteristic or subpart*](#rule_10-59): [Section 10.8.5, *Property term*](#section_10.8.5)
* [Rule 10-60, *Name may have multiple qualifier terms*](#rule_10-60): [Section 10.8.6, *Qualifier terms*](#section_10.8.6)
* [Rule 10-61, *Name has minimum necessary number of qualifier terms*](#rule_10-61): [Section 10.8.6, *Qualifier terms*](#section_10.8.6)
* [Rule 10-62, *Order of qualifiers is not significant*](#rule_10-62): [Section 10.8.6, *Qualifier terms*](#section_10.8.6)
* [Rule 10-63, *Redundant term in name is omitted*](#rule_10-63): [Section 10.8.7, *Representation terms*](#section_10.8.7)
* [Rule 10-64, *Element with simple content has representation term*](#rule_10-64): [Section 10.8.7, *Representation terms*](#section_10.8.7)
* [Rule 10-65, *Element with complex content has representation term when appropriate*](#rule_10-65): [Section 10.8.7, *Representation terms*](#section_10.8.7)
* [Rule 10-66, *Element with complex content has representation term only when appropriate*](#rule_10-66): [Section 10.8.7, *Representation terms*](#section_10.8.7)
* [Rule 10-67, *Machine-readable annotations are valid*](#rule_10-67): [Section 10.9, *Machine-readable annotations*](#section_10.9)
* [Rule 10-68, *Component marked as deprecated is deprecated component*](#rule_10-68): [Section 10.9.1.1, *Deprecation*](#section_10.9.1.1)
* [Rule 10-69, *Deprecated annotates schema component*](#rule_10-69): [Section 10.9.1.1, *Deprecation*](#section_10.9.1.1)
* [Rule 10-70, *External import indicator annotates import*](#rule_10-70): [Section 10.9.1.2, *External adapters*](#section_10.9.1.2)
* [Rule 10-71, *External adapter type indicator annotates complex type*](#rule_10-71): [Section 10.9.1.2, *External adapters*](#section_10.9.1.2)
* [Rule 10-72, *appinfo:appliesToTypes annotates metadata element*](#rule_10-72): [Section 10.9.1.3, *appinfo:appliesToTypes annotation*](#section_10.9.1.3)
* [Rule 10-73, *appinfo:appliesToTypes references types*](#rule_10-73): [Section 10.9.1.3, *appinfo:appliesToTypes annotation*](#section_10.9.1.3)
* [Rule 10-74, *appinfo:appliesToElements annotates metadata element*](#rule_10-74): [Section 10.9.1.4, *appinfo:appliesToElements annotation*](#section_10.9.1.4)
* [Rule 10-75, *appinfo:appliesToElements references elements*](#rule_10-75): [Section 10.9.1.4, *appinfo:appliesToElements annotation*](#section_10.9.1.4)
* [Rule 10-76, *appinfo:LocalTerm annotates schema*](#rule_10-76): [Section 10.9.2, *Local terminology*](#section_10.9.2)
* [Rule 10-77, *appinfo:LocalTerm has literal or definition*](#rule_10-77): [Section 10.9.2, *Local terminology*](#section_10.9.2)
* [Rule 10-78, *Use structures consistent with specification*](#rule_10-78): [Section 10.10, *NIEM structural facilities*](#section_10.10)
* [Rule 11-1, *Name of type ends in “Type”*](#rule_11-1): [Section 11.1, *Type definition components*](#section_11.1)
* [Rule 11-2, *Only types have name ending in “Type” or “SimpleType”*](#rule_11-2): [Section 11.1, *Type definition components*](#section_11.1)
* [Rule 11-3, *Base type definition defined by conformant schema*](#rule_11-3): [Section 11.1.1, *Type definition hierarchy*](#section_11.1.1)
* [Rule 11-4, *Name of simple type ends in “SimpleType”*](#rule_11-4): [Section 11.1.2, *Simple type definition*](#section_11.1.2)
* [Rule 11-5, *Use lists only when data is uniform*](#rule_11-5): [Section 11.1.2.1, *Derivation by list*](#section_11.1.2.1)
* [Rule 11-6, *List item type defined by conformant schemas*](#rule_11-6): [Section 11.1.2.1, *Derivation by list*](#section_11.1.2.1)
* [Rule 11-7, *Union member types defined by conformant schemas*](#rule_11-7): [Section 11.1.2.2, *Derivation by union*](#section_11.1.2.2)
* [Rule 11-8, *Name of a code simple type ends in “CodeSimpleType”*](#rule_11-8): [Section 11.1.2.3, *Code simple types*](#section_11.1.2.3)
* [Rule 11-9, *Code simple type corresponds to a code list*](#rule_11-9): [Section 11.1.2.3, *Code simple types*](#section_11.1.2.3)
* [Rule 11-10, *Attribute of code simple type has code representation term*](#rule_11-10): [Section 11.1.2.3, *Code simple types*](#section_11.1.2.3)
* [Rule 11-11, *Complex type with simple content has structures:SimpleObjectAttributeGroup*](#rule_11-11): [Section 11.1.3, *Complex type definition*](#section_11.1.3)
* [Rule 11-12, *Element type does not have a simple type name*](#rule_11-12): [Section 11.2.1, *Element declaration*](#section_11.2.1)
* [Rule 11-13, *Element type is from conformant namespace*](#rule_11-13): [Section 11.2.1, *Element declaration*](#section_11.2.1)
* [Rule 11-14, *Name of element that ends in “Abstract” is abstract*](#rule_11-14): [Section 11.2.1, *Element declaration*](#section_11.2.1)
* [Rule 11-15, *Name of element declaration with simple content has representation term*](#rule_11-15): [Section 11.2.1.1, *Object element declarations*](#section_11.2.1.1)
* [Rule 11-16, *Name of element declaration with simple content has representation term*](#rule_11-16): [Section 11.2.1.1, *Object element declarations*](#section_11.2.1.1)
* [Rule 11-17, *Element substitution group defined by conformant schema*](#rule_11-17): [Section 11.2.2, *Element substitution group*](#section_11.2.2)
* [Rule 11-18, *Attribute type defined by conformant schema*](#rule_11-18): [Section 11.2.3, *Attribute declaration*](#section_11.2.3)
* [Rule 11-19, *Attribute name uses representation term*](#rule_11-19): [Section 11.2.3, *Attribute declaration*](#section_11.2.3)
* [Rule 11-20, *Element or attribute declaration introduced only once into a type*](#rule_11-20): [Section 11.3.2.1, *Element use*](#section_11.3.2.1)
* [Rule 11-21, *Element reference defined by conformant schema*](#rule_11-21): [Section 11.3.2.1, *Element use*](#section_11.3.2.1)
* [Rule 11-22, *Referenced attribute defined by conformant schemas*](#rule_11-22): [Section 11.3.3, *Attribute use*](#section_11.3.3)
* [Rule 11-23, *Schema uses only known attribute groups*](#rule_11-23): [Section 11.3.3.1, *Attribute group use*](#section_11.3.3.1)
* [Rule 11-24, *Data definition does not introduce ambiguity*](#rule_11-24): [Section 11.6.1, *Human-readable documentation*](#section_11.6.1)
* [Rule 11-25, *Object class has only one meaning*](#rule_11-25): [Section 11.6.1, *Human-readable documentation*](#section_11.6.1)
* [Rule 11-26, *Data definition of a part does not redefine the whole*](#rule_11-26): [Section 11.6.1, *Human-readable documentation*](#section_11.6.1)
* [Rule 11-27, *Do not leak representation into data definition*](#rule_11-27): [Section 11.6.1, *Human-readable documentation*](#section_11.6.1)
* [Rule 11-28, *Data definition follows 11179-4 requirements*](#rule_11-28): [Section 11.6.1, *Human-readable documentation*](#section_11.6.1)
* [Rule 11-29, *Data definition follows 11179-4 recommendations*](#rule_11-29): [Section 11.6.1, *Human-readable documentation*](#section_11.6.1)
* [Rule 11-30, *xs:documentation has xml:lang*](#rule_11-30): [Section 11.6.1, *Human-readable documentation*](#section_11.6.1)
* [Rule 11-31, *Standard opening phrase for augmentation point element data definition*](#rule_11-31): [Section 11.6.1.1.1, *Element and attribute opening phrases*](#section_11.6.1.1.1)
* [Rule 11-32, *Standard opening phrase for augmentation element data definition*](#rule_11-32): [Section 11.6.1.1.1, *Element and attribute opening phrases*](#section_11.6.1.1.1)
* [Rule 11-33, *Standard opening phrase for metadata element data definition*](#rule_11-33): [Section 11.6.1.1.1, *Element and attribute opening phrases*](#section_11.6.1.1.1)
* [Rule 11-34, *Standard opening phrase for association element data definition*](#rule_11-34): [Section 11.6.1.1.1, *Element and attribute opening phrases*](#section_11.6.1.1.1)
* [Rule 11-35, *Standard opening phrase for abstract element data definition*](#rule_11-35): [Section 11.6.1.1.1, *Element and attribute opening phrases*](#section_11.6.1.1.1)
* [Rule 11-36, *Standard opening phrase for date element or attribute data definition*](#rule_11-36): [Section 11.6.1.1.1, *Element and attribute opening phrases*](#section_11.6.1.1.1)
* [Rule 11-37, *Standard opening phrase for quantity element or attribute data definition*](#rule_11-37): [Section 11.6.1.1.1, *Element and attribute opening phrases*](#section_11.6.1.1.1)
* [Rule 11-38, *Standard opening phrase for picture element or attribute data definition*](#rule_11-38): [Section 11.6.1.1.1, *Element and attribute opening phrases*](#section_11.6.1.1.1)
* [Rule 11-39, *Standard opening phrase for indicator element or attribute data definition*](#rule_11-39): [Section 11.6.1.1.1, *Element and attribute opening phrases*](#section_11.6.1.1.1)
* [Rule 11-40, *Standard opening phrase for identification element or attribute data definition*](#rule_11-40): [Section 11.6.1.1.1, *Element and attribute opening phrases*](#section_11.6.1.1.1)
* [Rule 11-41, *Standard opening phrase for name element or attribute data definition*](#rule_11-41): [Section 11.6.1.1.1, *Element and attribute opening phrases*](#section_11.6.1.1.1)
* [Rule 11-42, *Standard opening phrase for element or attribute data definition*](#rule_11-42): [Section 11.6.1.1.1, *Element and attribute opening phrases*](#section_11.6.1.1.1)
* [Rule 11-43, *Standard opening phrase for association type data definition*](#rule_11-43): [Section 11.6.1.1.2, *Complex type opening phrases*](#section_11.6.1.1.2)
* [Rule 11-44, *Standard opening phrase for augmentation type data definition*](#rule_11-44): [Section 11.6.1.1.2, *Complex type opening phrases*](#section_11.6.1.1.2)
* [Rule 11-45, *Standard opening phrase for metadata type data definition*](#rule_11-45): [Section 11.6.1.1.2, *Complex type opening phrases*](#section_11.6.1.1.2)
* [Rule 11-46, *Standard opening phrase for complex type data definition*](#rule_11-46): [Section 11.6.1.1.2, *Complex type opening phrases*](#section_11.6.1.1.2)
* [Rule 11-47, *Standard opening phrase for simple type data definition*](#rule_11-47): [Section 11.6.1.1, *Data definition opening phrases*](#section_11.6.1.1)
* [Rule 11-48, *Same namespace means same components*](#rule_11-48): [Section 11.7.1, *xs:schema document element restrictions*](#section_11.7.1)
* [Rule 11-49, *Different version means different view*](#rule_11-49): [Section 11.7.1, *xs:schema document element restrictions*](#section_11.7.1)
* [Rule 11-50, *Reference schema document imports reference schema document*](#rule_11-50): [Section 11.8, *Schema assembly*](#section_11.8)
* [Rule 11-51, *Extension schema document imports reference or extension schema document*](#rule_11-51): [Section 11.8, *Schema assembly*](#section_11.8)
* [Rule 11-52, *Structures imported as conformant*](#rule_11-52): [Section 11.8.1, *Supporting namespaces are imported as conformant*](#section_11.8.1)
* [Rule 11-53, *XML namespace imported as conformant*](#rule_11-53): [Section 11.8.1, *Supporting namespaces are imported as conformant*](#section_11.8.1)
* [Rule 11-54, *Each namespace may have only a single root schema in a schema set*](#rule_11-54): [Section 11.8, *Schema assembly*](#section_11.8)
* [Rule 11-55, *Consistently marked namespace imports*](#rule_11-55): [Section 11.8, *Schema assembly*](#section_11.8)
* [Rule 12-1, *Instance must be schema-valid*](#rule_12-1): [Section 12, *XML instance document rules*](#section_12)
* [Rule 12-2, *Empty content has no meaning*](#rule_12-2): [Section 12.1, *The meaning of NIEM data*](#section_12.1)
* [Rule 12-3, *Element has only one resource identifying attribute*](#rule_12-3): [Section 12.2, *Identifiers and references*](#section_12.2)
* [Rule 12-4, *Attribute structures:ref must reference structures:id*](#rule_12-4): [Section 12.2.1, *Local identifiers and references*](#section_12.2.1)
* [Rule 12-5, *Linked elements have same validation root*](#rule_12-5): [Section 12.2.1, *Local identifiers and references*](#section_12.2.1)
* [Rule 12-6, *Attribute structures:ref references element of correct type*](#rule_12-6): [Section 12.2.1, *Local identifiers and references*](#section_12.2.1)
* [Rule 12-7, *structures:uri denotes resource identifier*](#rule_12-7): [Section 12.2.2, *Uniform resource identifiers in NIEM data*](#section_12.2.2)
* [Rule 12-8, *structures:id and structures:ref denote resource identifier*](#rule_12-8): [Section 12.2.2, *Uniform resource identifiers in NIEM data*](#section_12.2.2)
* [Rule 12-9, *Nested elements and references have the same meaning.*](#rule_12-9): [Section 12.2.4, *Reference and content elements have same meaning*](#section_12.2.4)
* [Rule 12-10, *Order of properties is expressed via structures:sequenceID*](#rule_12-10): [Section 12.3, *Property order and sequence identifiers*](#section_12.3)
* [Rule 12-11, *Metadata applies to referring entity*](#rule_12-11): [Section 12.4, *Instance metadata*](#section_12.4)
* [Rule 12-12, *Referent of structures:relationshipMetadata annotates relationship*](#rule_12-12): [Section 12.4, *Instance metadata*](#section_12.4)
* [Rule 12-13, *Values of structures:metadata refer to values of structures:id*](#rule_12-13): [Section 12.4, *Instance metadata*](#section_12.4)
* [Rule 12-14, *Values of structures:relationshipMetadata refer to values of structures:id*](#rule_12-14): [Section 12.4, *Instance metadata*](#section_12.4)
* [Rule 12-15, *structures:metadata and structures:relationshipMetadata refer to metadata elements*](#rule_12-15): [Section 12.4, *Instance metadata*](#section_12.4)
* [Rule 12-16, *Attribute structures:metadata references metadata element*](#rule_12-16): [Section 12.4, *Instance metadata*](#section_12.4)
* [Rule 12-17, *Attribute structures:relationshipMetadata references metadata element*](#rule_12-17): [Section 12.4, *Instance metadata*](#section_12.4)
* [Rule 12-18, *Metadata is applicable to element*](#rule_12-18): [Section 12.4, *Instance metadata*](#section_12.4)

# Appendix F. General index

* appinfo namespace: [Section 10](#d3e9098), [Section 10.9](#d3e13441), [Section 10.9.1](#d3e13467), [Section 10.9.1 (definition)](#definition_appinfo_namespace), [Section 10.9.1](#d3e13486)
* application information: [Section 10.9 (definition)](#definition_application_information), [Section 10.9](#d3e13429), [Section 10.9](#d3e13438), [Section 10.9.2](#d3e13774), [Section 11.6](#d3e14803)
* association: [Section 10.3](#d3e10363), [Section 10.3 (definition)](#definition_association)
* association element declaration: [Section 10.3](#d3e10403), [Section 10.3](#d3e10488), [Section 10.3 (definition)](#X0800fc6c42d99fd31cd824ea485f2a4b7ee49f3)
* association type: [Section 5.6.3.2](#d3e3250), [Section 5.6.3.2](#d3e3268), [Section 5.6.3.3](#d3e3351), [Section 5.6.3.3](#d3e3369), [Section 5.6.3.6](#d3e3657), [Section 5.6.3.7](#d3e3757), [Section 5.6.3.8](#d3e3874), [Section 5.6.4](#d3e4007), [Section 5.6.5.1](#d3e4072), [Section 5.6.5.1](#d3e4096), [Section 5.6.5.2](#d3e4230), [Section 10.3](#d3e10409), [Section 10.3](#d3e10415), [Section 10.3](#d3e10491), [Section 10.3 (definition)](#definition_association_type), [Section 10.3.1](#d3e10571), [Section 10.3.2](#d3e10606), [Section 10.4](#d3e10697), [Section 10.4](#d3e11017), [Section 10.4.1](#d3e11062)
* attribute: [Section 3.3 (definition)](#definition_attribute)
* augmentable type: [Section 10.4](#d3e10803), [Section 10.4.1 (definition)](#definition_augmentable_type), [Section 10.4.2](#d3e11160)
* augmentation: [Section 6.5.5](#d3e4998), [Section 10.4](#d3e10770), [Section 10.4 (definition)](#definition_augmentation), [Section 10.4.4](#d3e11364), [Section 10.4.4](#d3e11372), [Section 10.4.4](#d3e11378)
* augmentation element declaration: [Section 10.4](#d3e10793), [Section 10.4.4](#d3e11331), [Section 10.4.4 (definition)](#Xfaa24539372ac2c23b1e89a5bfc415f590cea1e), [Section 10.4.4](#d3e11417), [Section 10.4.5](#d3e11607)
* augmentation type: [Section 5.6.3.8](#d3e3889), [Section 10.4](#d3e10773), [Section 10.4.4 (definition)](#definition_augmentation_type), [Section 10.4.4](#d3e11353), [Section 10.4.4](#d3e11479), [Section 10.4.4](#d3e11486)
* base type definition: [Section 3.4.1 (definition)](#definition_base_type_definition), [Section 9.1.1.1](#d3e5942), [Section 9.1.1.1](#d3e5964), [Section 10.2.1.1](#d3e9275), [Section 10.2.1.1](#d3e9280), [Section 10.4.4](#d3e11520)
* code simple type: [Section 11.1.2.3 (definition)](#definition_code_simple_type)
* code type: [Section 10.2.4 (definition)](#definition_code_type)
* complex type definition: [Section 3.4.1 (definition)](#definition_complex_type_definition), [Section 10.4.1](#d3e11037), [Section 10.4.4](#d3e11318)
* conformance target: [Section 2.4](#d3e494), [Section 2.4.1](#d3e521), [Section 2.4.1](#d3e569), [Section 3.6 (definition)](#definition_conformance_target), [Section 4.1.1](#d3e1691), [Section 4.1.2](#d3e1784), [Section 4.1.3](#d3e1887), [Section 4.1.4](#d3e1984)
* conformance target identifier: [Section 3.6 (definition)](#definition_conformance_target_identifier), [Section 4.3](#d3e2179)
* conformant element information item: [Section 4.1.4](#d3e2036), [Section 4.1.4 (definition)](#Xb8c2328714130974c99941a242199ecbb7daea7), [Section 5.6.2](#d3e3001), [Section 5.6.3.2](#d3e3244), [Section 5.6.3.2](#d3e3262), [Section 5.6.3.3](#d3e3345), [Section 5.6.3.3](#d3e3363), [Section 5.6.3.4](#d3e3462), [Section 5.6.3.5](#d3e3549), [Section 5.6.3.6](#d3e3651), [Section 5.6.3.7](#d3e3751), [Section 5.6.3.8](#d3e3868), [Section 5.6.3.8](#d3e3883), [Section 5.6.3.8](#d3e3908), [Section 5.6.3.9](#d3e3946), [Section 5.6.4](#d3e3998), [Section 10.3](#d3e10485), [Section 10.4](#d3e10784), [Section 10.4](#d3e10800)
* conformant instance XML document: [Section 4.1.3](#d3e1884), [Section 4.1.4 (definition)](#X6a4ab7051f7203e0b47588881449d56367bed27), [Section 4.1.4](#d3e1991), [Section 4.1.4](#d3e2063), [Section 4.1.4](#d3e2074), [Section 4.2](#d3e2167), [Section 5.1](#d3e2375), [Section 5.6.2](#d3e2998), [Section 5.6.3.1](#d3e3225)
* conformant schema document set: [Section 4.1.3](#d3e1867), [Section 4.1.3 (definition)](#Xeff662d8d67f5b4e72a2955d729e0ac83cad2d6), [Section 4.1](#d3e1895), [Section 4.1](#d3e1904), [Section 4.2](#d3e2155)
* constraint rule: [Section 2.4.1](#d3e551), [Section 2.4.1 (definition)](#definition_constraint_rule)
* data definition: [Section 6.5.1](#d3e4890), [Section 6.5.1](#d3e4896), [Section 6.5.1](#d3e4904), [Section 7.4 (definition)](#definition_data_definition), [Section 7.4](#d3e5375), [Section 7.4](#d3e5415), [Section 9.1.2](#d3e6274), [Section 9.1.2](#d3e6361), [Section 9.1.3](#d3e6702), [Section 9.2.1](#d3e7081), [Section 9.2.3](#d3e7443), [Section 10.2.3.3](#d3e10106), [Section 10.2.3.4](#d3e10188), [Section 11.6.1](#d3e14840), [Section 11.6.1](#d3e14900), [Section 11.6.1](#d3e14920)
* deprecated component: [Section 10.9.1.1 (definition)](#definition_deprecated_component), [Section 10.9.1.1](#d3e13514), [Section 10.9.1.1](#d3e13542)
* documented component: [Section 7.4 (definition)](#definition_documented_component), [Section 7.6](#d3e5540), [Section 10.2.3.3](#d3e10052), [Section 10.2.3.3](#d3e10103), [Section 10.2.3.4](#d3e10185)
* effective conformance target identifier: [Section 2.4.3](#d3e819), [Section 3.6 (definition)](#X6d0b4369562c0703761463ba00cd0e2b474de22), [Section 4.3](#d3e2252), [Section 4.3](#d3e2279)
* element: [Section 3.3](#d3e1092), [Section 3.3 (definition)](#definition_element)
* element declaration: [Section 3.4.1 (definition)](#definition_element_declaration), [Section 9.2.1](#d3e7361), [Section 10.2.2](#d3e9499), [Section 10.2.2](#d3e9678), [Section 10.2.2](#d3e9695), [Section 10.3](#d3e10503), [Section 10.4.2](#d3e11157), [Section 10.4.4](#d3e11344), [Section 10.7](#d3e12161), [Section 11.2.1](#d3e14323), [Section 11.2.1](#d3e14350)
* extension schema document: [Section 2.4.1](#d3e600), [Section 4.1.2 (definition)](#definition_extension_schema_document), [Section 4.1.2](#d3e1798), [Section 4.1.2](#d3e1807), [Section 4.1.2](#d3e1839), [Section 4.1](#d3e1950), [Section 4.1.4](#d3e2013), [Section 4.1.4](#d3e2056), [Section 4.2](#d3e2144), [Section 5.6.3.6](#d3e3675), [Section 5.6.3.7](#d3e3775), [Section 5.6.5.1](#d3e4063), [Section 5.6.5.2](#d3e4131), [Section 5.6.5.3](#d3e4275), [Section 5.6.5.3](#d3e4287), [Section 8.2](#d3e5647), [Section 9.3.4](#d3e8194), [Section 10.2.2](#d3e9511), [Section 10.2.3](#d3e9781), [Section 10.3](#d3e10527), [Section 10.4.1](#d3e11049), [Section 10.5.2](#d3e11719), [Section 11.8](#d3e15467)
* external adapter type: [Section 10.2.3](#d3e9820), [Section 10.2.3.2 (definition)](#definition_external_adapter_type), [Section 10.2.3.2](#d3e9921), [Section 10.2.3.3](#d3e10076)
* external schema document: [Section 10.2.3 (definition)](#definition_external_schema_document), [Section 10.2.3](#d3e9808), [Section 10.2.3](#d3e9849)
* informative: [Section 2.4](#d3e431), [Section 2.4 (definition)](#definition_informative)
* instance document: [Section 3.4 (definition)](#definition_instance_document), [Section 4.1.4](#d3e1974)
* interpretation rule: [Section 2.4.1](#d3e558), [Section 2.4.1 (definition)](#definition_interpretation_rule)
* local term: [Section 10.8.2.1 (definition)](#definition_local_term), [Section 10.8.2.1](#d3e12639)
* machine-readable annotation: [Section 10.9](#d3e13362), [Section 10.9 (definition)](#definition_machine-readable_annotation)
* metadata element declaration: [Section 10.5.2 (definition)](#definition_metadata_element_declaration), [Section 12.4](#d3e17430)
* metadata type: [Section 5.6.3.2](#d3e3271), [Section 5.6.3.3](#d3e3372), [Section 5.6.3.6](#d3e3660), [Section 5.6.3.7](#d3e3760), [Section 10.5.1 (definition)](#definition_metadata_type), [Section 10.5.2](#d3e11725), [Section 10.5.2](#d3e11758), [Section 12.4](#d3e17123)
* normative: [Section 2.4](#d3e428), [Section 2.4 (definition)](#definition_normative), [Section 2.4](#d3e491)
* object type: [Section 5.6.3.2](#d3e3247), [Section 5.6.3.2](#d3e3265), [Section 5.6.3.3](#d3e3348), [Section 5.6.3.3](#d3e3366), [Section 5.6.3.4](#d3e3465), [Section 5.6.3.5](#d3e3552), [Section 5.6.3.6](#d3e3654), [Section 5.6.3.7](#d3e3754), [Section 5.6.3.8](#d3e3871), [Section 5.6.4](#d3e4004), [Section 5.6.5.1](#d3e4069), [Section 5.6.5.1](#d3e4093), [Section 5.6.5.2](#d3e4227), [Section 10.2.1 (definition)](#definition_object_type), [Section 10.2.2](#d3e9441), [Section 10.2.2](#d3e9664), [Section 10.4](#d3e10694), [Section 10.4.1](#d3e11069)
* reference schema document: [Section 2.4.1](#d3e596), [Section 4.1.1 (definition)](#definition_reference_schema_document), [Section 4.1.1](#d3e1708), [Section 4.1.1](#d3e1717), [Section 4.1.2](#d3e1781), [Section 4.1.2](#d3e1818), [Section 4.1.2](#d3e1836), [Section 4.1.2](#d3e1842), [Section 4.1.2](#d3e1851), [Section 4.1](#d3e1926), [Section 4.1.4](#d3e2010), [Section 4.2](#d3e2133), [Section 5.6.3.6](#d3e3672), [Section 5.6.3.7](#d3e3772), [Section 5.6.5.1](#d3e4060), [Section 5.6.5.2](#d3e4128), [Section 5.6.5.3](#d3e4272), [Section 8.2](#d3e5644), [Section 8.2](#d3e5667), [Section 10.2.2](#d3e9508), [Section 10.2.3](#d3e9774), [Section 10.2.3.3](#d3e10043), [Section 10.3](#d3e10506), [Section 10.3](#d3e10524), [Section 10.4.1](#d3e11046), [Section 10.4.5](#d3e11598), [Section 11.1.3](#d3e14209), [Section 11.8](#d3e15440), [Section 11.8](#d3e15470)
* representation element declaration: [Section 10.7 (definition)](#X1bc7717493f0a423581c67a1dc9df1fdfdc9316)
* role type: [Section 10.2.2](#d3e9320), [Section 10.2.2](#d3e9354), [Section 10.2.2](#d3e9430), [Section 10.2.2 (definition)](#definition_role_type), [Section 10.2.2](#d3e9449), [Section 10.2.2](#d3e9452), [Section 10.2.2](#d3e9463), [Section 10.2.2](#d3e9474), [Section 10.2.2](#d3e9525), [Section 10.2.2](#d3e9744)
* RoleOf element: [Section 10.2.2 (definition)](#definition_RoleOf_element), [Section 10.2.2](#d3e9532), [Section 10.2.2](#d3e9661), [Section 10.2.2](#d3e9716), [Section 10.2.2](#d3e9726), [Section 10.2.2](#d3e9729), [Section 10.2.2](#d3e9732), [Section 10.2.2](#d3e9735), [Section 10.2.2](#d3e9741)
* schema component: [Section 3.4 (definition)](#definition_schema_component), [Section 4.1.2](#d3e1778), [Section 4.1.2](#d3e1845), [Section 5.1](#d3e2305), [Section 7.4](#d3e5354), [Section 7.4](#d3e5372), [Section 9.3.1](#d3e7847), [Section 10.2.3](#d3e9805), [Section 10.4.4](#d3e11473), [Section 10.9.1.1](#d3e13533)
* schema document: [Section 1.1](#d3e228), [Section 2.4.3](#d3e721), [Section 2.4.3](#d3e737), [Section 2.4.3](#d3e752), [Section 2.4.3](#d3e758), [Section 3.4 (definition)](#definition_schema_document), [Section 3.4](#d3e1405), [Section 3.4](#d3e1421), [Section 4.1.1](#d3e1685), [Section 4.1.1](#d3e1711), [Section 4.1.2](#d3e1775), [Section 4.1](#d3e1917), [Section 4.1](#d3e1941), [Section 6.2.10](#d3e4715), [Section 7.3](#d3e5161), [Section 7.3](#d3e5164), [Section 7.3](#d3e5181), [Section 9.8.1](#d3e8919), [Section 10.2.3](#d3e9765), [Section 10.2.3](#d3e9788), [Section 10.4.2](#d3e11154), [Section 11.3.2.1](#d3e14634)
* simple type definition: [Section 3.4.1 (definition)](#definition_simple_type_definition)
* structures namespace: [Section 7.6](#d3e5523), [Section 10](#d3e9095), [Section 10.2.3](#d3e9791), [Section 10.2.3](#d3e9799), [Section 10.10 (definition)](#definition_structures_namespace), [Section 10.10](#d3e13840)
* valid: [Section 3.4 (definition)](#definition_valid), [Section 3.4](#d3e1382), [Section 3.4](#d3e1411), [Section 4.1.3](#d3e1881), [Section 4.1.4](#d3e1977)
* XML document: [Section 3.2 (definition)](#definition_XML_document), [Section 3.4](#d3e1313), [Section 3.4](#d3e1379), [Section 3.4](#d3e1414), [Section 4.1.1](#d3e1694), [Section 4.1.4](#d3e1971), [Section 4.1.4](#d3e2007), [Section 6.2.10](#d3e4644), [Section 7.1](#d3e5091), [Section 7.1](#d3e5107), [Section 7.3](#d3e5158), [Section 7.3](#d3e5217), [Section 9.8](#d3e8872)
* XML Schema: [Section 2.5](#d3e897), [Section 3.4 (definition)](#definition_XML_Schema), [Section 3.4](#d3e1376), [Section 3.4](#d3e1385), [Section 3.4](#d3e1408), [Section 3.4](#d3e1424), [Section 6.2.10](#d3e4600), [Section 9.8](#d3e8869)
* XML Schema definition language: [Section 3.4](#d3e1219), [Section 3.4 (definition)](#X7e4888920c30d9bbda28173a5b8b96fa0c14a07), [Section 3.4](#d3e1279), [Section 3.4](#d3e1316), [Section 7.3](#d3e5155), [Section 9.1.3.2.1](#d3e6833), [Section 9.4](#d3e8247), [Section 9.8.1](#d3e8893), [Section 10](#d3e9084), [Section 10.7](#d3e11974)
* XML Schema document set: [Section 3.4 (definition)](#definition_XML_Schema_document_set), [Section 9.8](#d3e8866)