NIEM GitHub

The National Information Exchange Model (NIEM) is a community-driven, common vocabulary that enables efficient information exchange across diverse public and private organizations. NIEM can save time and money by providing consistent, reusable data terms and definitions, and repeatable processes. This site provides information and resources to help the community use NIEM.

PDF generated from niem.github.io on July 25, 2021

- Training (http://localhost:4010/training/)
- Reference (http://localhost:4010/reference/)
- JSON (http://localhost:4010/json/)
- Community Activities (http://localhost:4010/community/)
- Geospatial Integration (http://localhost:4010/geospatial/)

BESbswyBEbbffjfjyBESbswyBEbbffjfjyBESbswyBEbbffjfjyBESbswyBEbbffjfjyBESbswyBEbbffjfjyBESbswyBEbbff

Training

Welcome to the new home of NIEM training

We recently modernized our training to include user-driven tutorials and reference content on GitHub. It's an evolution based on how people learn these days.

The content addresses the two types of training audiences we historically see:

1. Someone who is a newcomer to NIEM and needs to get up to speed on how to do things. This could include managing domain model content, developing an exchange, implementing an exchange, publishing a data set, or reusing the vocabulary. If this is you, please see the training tracks below.
2. Someone who is already familiar with NIEM and is looking for quick reference material to determine or verify how to do something. If this is you, please visit the Reference section.

This section represents our *work in progress*. As we knock out content, we push it out to the user community for feedback and use.

- Just getting started? Head over to the main site to learn About NIEM, How to Get Started, and much more!
- Looking for information about a specific topic? See the Reference section for general documentation.

Training tracks

Choose from below for the kind of materials that will best fit your role in using NIEM:

- Domain Modeler
  A domain modeler creates and manages the content in a NIEM domain.
  Begin training
- IEPD Developer
  An Information Exchange Package Documentation (IEPD) Developer designs, builds, and validates the components (artifacts) of an Information Exchange Package (IEP).
  Begin training
- IEPD Implementer
  An IEPD implementer exchanges information based on the format, rules, and guidance provided by an existing IEPD specification.
  Begin training

Domain Modeler

A domain modeler creates and manages the content in a NIEM domain.

- General information
  - Other sections on this site
  - External resources

General information

There are a large number of tools and reference materials available to help you build and manage a domain. You can see a large list of information resources at the Reference page. Some of the sections you may find most immediately useful are the ones highlighted below:

Other sections on this site

The following is a list of other sections on niem.github.io that may be useful for domain modelers:

- Artifacts

  Learn about the kinds of files that make up NIEM releases and IEPDs.

- Content

  A high-level perspective on the kinds of content that appear in Core and the domains.

- Concepts for data modeling

  Learn about properties, types, augmentations, and other building blocks used to construct the model.

- Releases

  A NIEM release is a versioned set of schemas and supporting artifacts published by the NIEM Management Office. Release schemas contain reusable properties and types meant for use in information exchanges. These schemas include NIEM Core, domains, and code tables.

- Specifications

  NIEM specifications provide rules and guidance in order to design consistent and well-defined information exchanges. Specifications are managed by the NIEM Technical Architecture Committee (NTAC).

- Tools

NIEM tools provide support for searching and exploring NIEM content and developing NIEM domains and exchanges.

- NIEM Domain Modeling Guide

  How to design a Community of Interest

External resources

- Domain Governance

  Learn about governance committees and domain content management on niem.gov.

- About the NIEM Model

  Review the NIEM model architecture and model management on niem.gov.

IEPD Developer

An Information Exchange Package Documentation (IEPD) Developer designs, builds, and validates the components (artifacts) of an Information Exchange Package (IEP).

The IEPD allows senders and receivers to exchange information in a manner that both understand. The exchange is typically in the form of XML schema documents that define XML instance documents, and other documents that describe, for example, the version history and contents of the documents.

You are here to learn the steps for developing an IEPD according to the IEPD Lifecycle.

IEPD Lifecycle

You develop the IEPD according to a six-phase lifecycle, as shown graphically and listed in the following sections. The phases link to pages with detailed, introductory information about that phase. More lifecycle phase information can be found in the IEPD lifecycle section.

Start with Scenario Planning and work your way through the list in order. See the tutorial for a step-by-step guide to create a simple IEPD set.

1. **Scenario Planning**: Review background information related to your information exchange, assess resource impact, understand business context, and identify information exchange business scenarios.

2. **Analyze Requirements**: Further elaborate on the information exchange scenario to understand and document the business context and data requirements.

3. **Map and Model**: Compare the data requirements for the exchange gathered in the last step to what is available in NIEM. Record results in a mapping spreadsheet and plan how requirements that aren't available in NIEM should be modeled and added as IEPD extensions.

4. **Build and Validate**: Create subset and extension schemas for the IEPD based on the mapping and modeling decisions made in the previous step. Check that schemas are valid and NIEM-conformant. Build and validate sample instances for additional testing.

5. **Assemble and Document**: Finalize the documentation and other IEPD artifacts and then package all related files for the IEPD into a single, self-contained, self-documented, portable archive file.

6. **Publish and Implement**: Publish the IEPD for search, discovery, and reuse, and implement the IEPD for production.

IEPD Tutorial

The Simple IEPD tutorial takes you step-by-step through the creation process starting with the Scenario Planning phase. Future tutorials will reference and build from it.

- Simple IEPD - This tutorial shows you how to create a very simple IEPD set. Once you are comfortable with the process and output, you can move on to more complex IEPDs.

IEPD Developer Resources

There are a large number of tools and reference materials available to help you in your development effort. You can see a comprehensive list of information resources at the **Reference** page. Some of the sections you may find most immediately useful are the ones highlighted below:

- Artifacts

  Learn about the kinds of files that make up NIEM releases and IEPDs.

- Content

  A high-level perspective on the kinds of content that appear in Core and the domains.

- Concepts for data modeling

  Learn about properties, types, augmentations, and other building blocks used to

construct the model.

- IEPDs

  An Information Exchange Package Documentation (IEPD) is a collection of NIEM artifacts. They define and describe the context, content, semantics, and structure of one or more implementable information exchanges.

- Releases

  A NIEM release is a versioned set of schemas and supporting artifacts published by the NIEM Management Office. Release schemas contain reusable properties and types meant for use in information exchanges. These schemas include NIEM Core, domains, and code tables.

- Specifications

  NIEM specifications provide rules and guidance in order to design consistent and well-defined information exchanges. Specifications are managed by the NIEM Technical Architecture Committee (NTAC).

- Tools

  NIEM tools provide support for searching and exploring NIEM content and developing NIEM domains and exchanges.

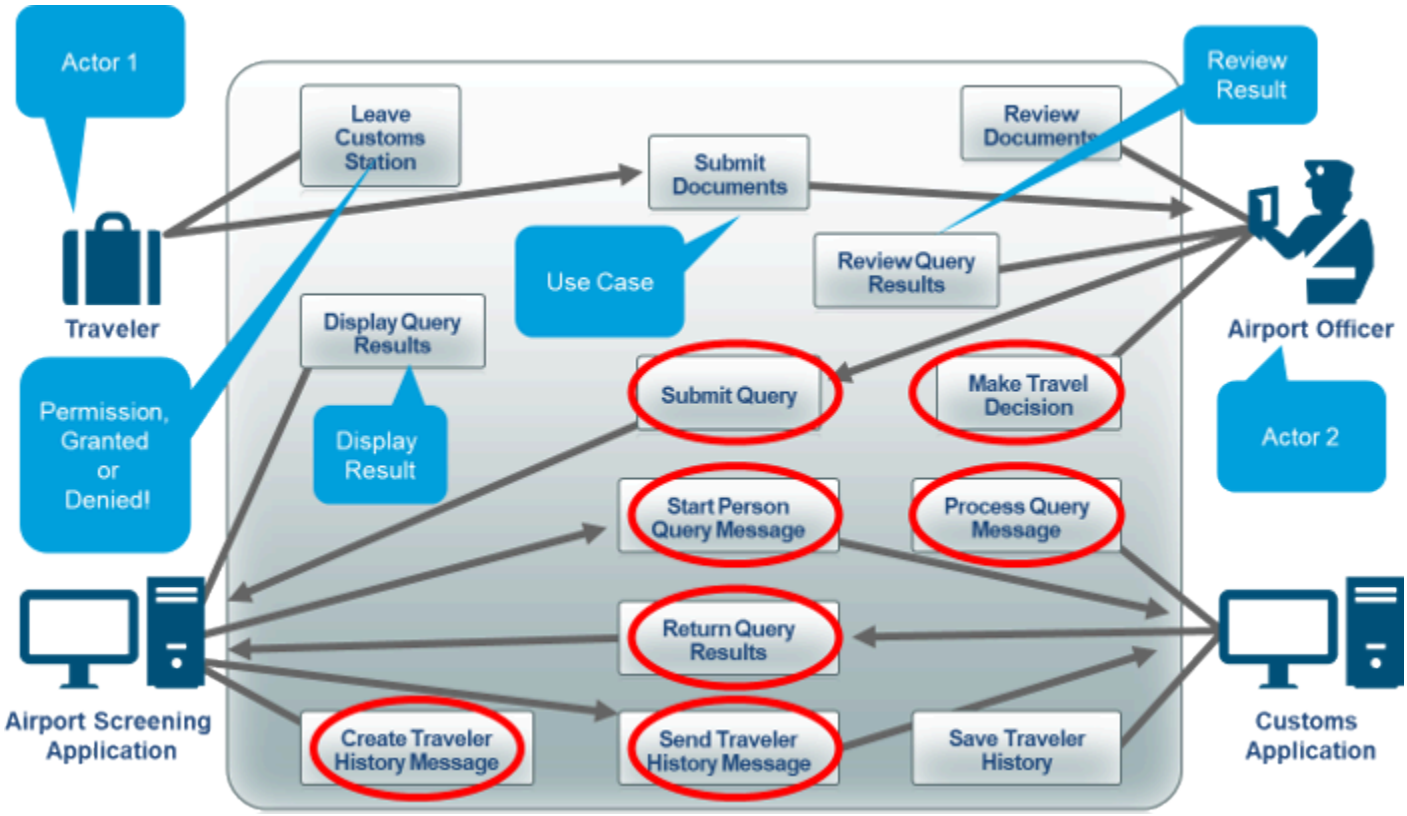
Scenario Planning

The scenario planning phase is the first step you take in IEPD development.

In scenario planning, the IEPD stakeholders decided and agreed on what will go into the package, and developed one or more diagrams from the following types:

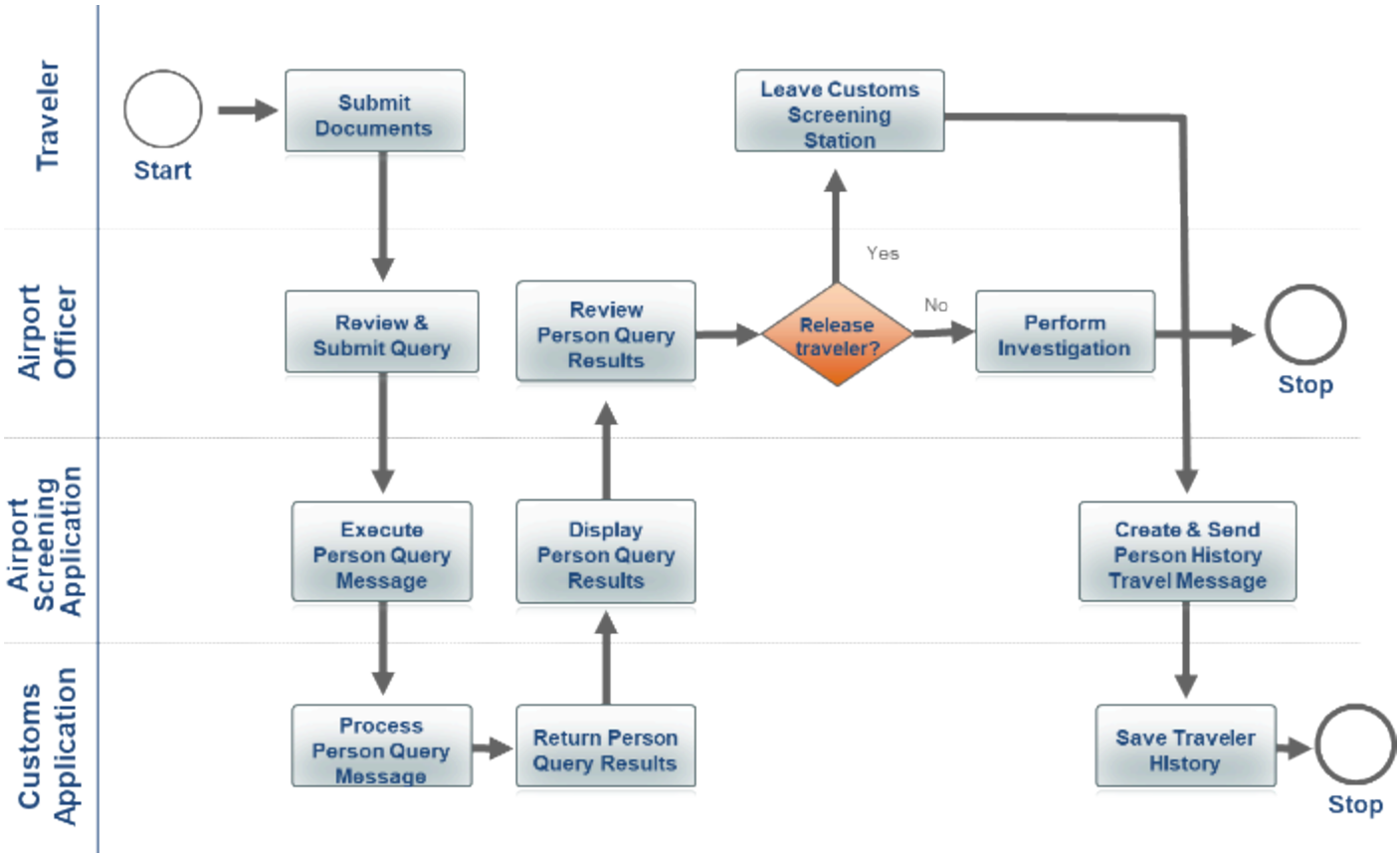
Use Case Diagram Example

A use case diagram describes functionality.



Business Process Diagram Example

A business process diagram shows the graphical and sequential activities involved in an exchange.

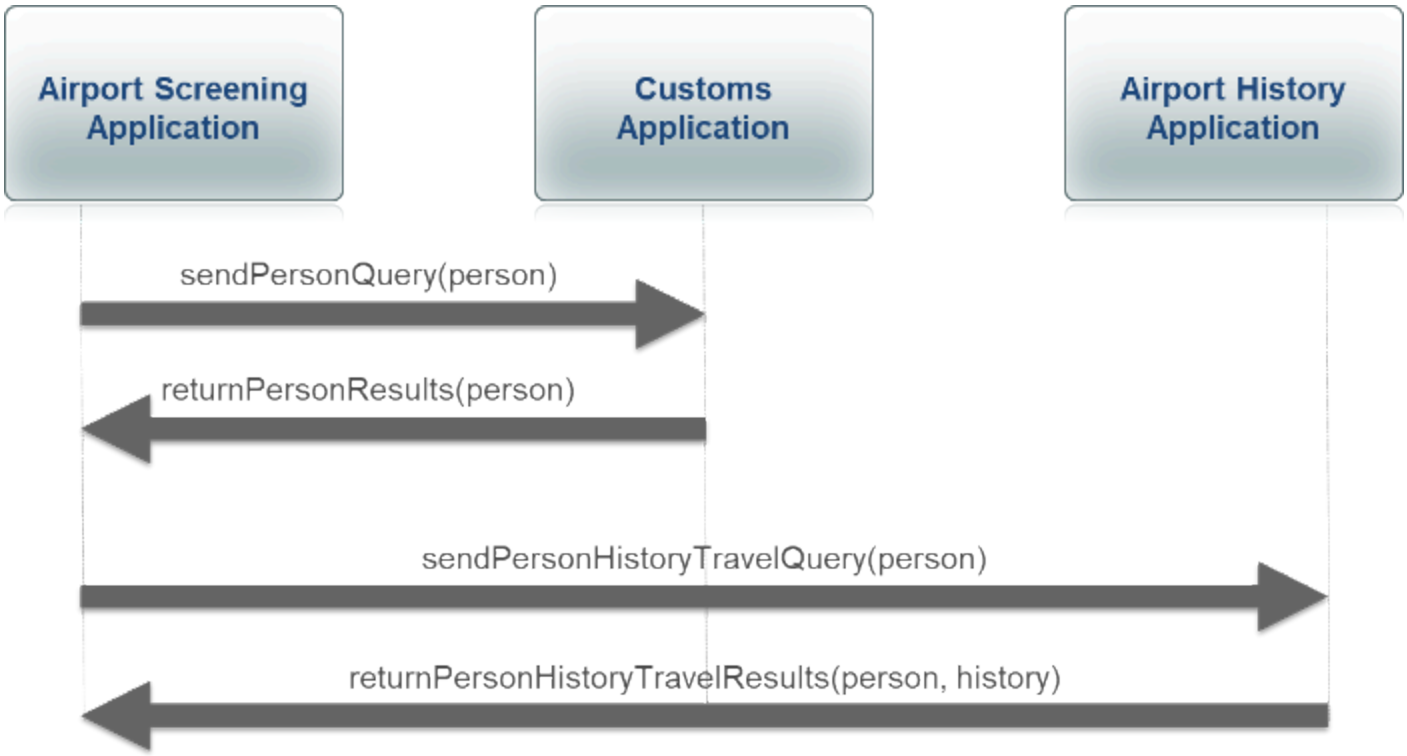


Sequence Diagram Example

A sequence diagram shows how applications or systems operate with one another. It displays the sequential order of operational processes or messages between applications as horizontal arrows between the parallel, vertical lines that are used for applications.

This sub-process of the traveler processing system has three interacting applications:

- Airport Screening Application
- Customs Application
- Airport History Application



When you have answers to the questions and complications discussed earlier, and one or more detailed diagrams, proceed to the next phase: Analyze Requirements.

Analyze Requirements

The analyze requirements phase is the next step you take in IEPD development.

You have graphically defined your scenario, and now you need to define your business rules and requirements, which define the exchange as well as the expectations of the exchange.

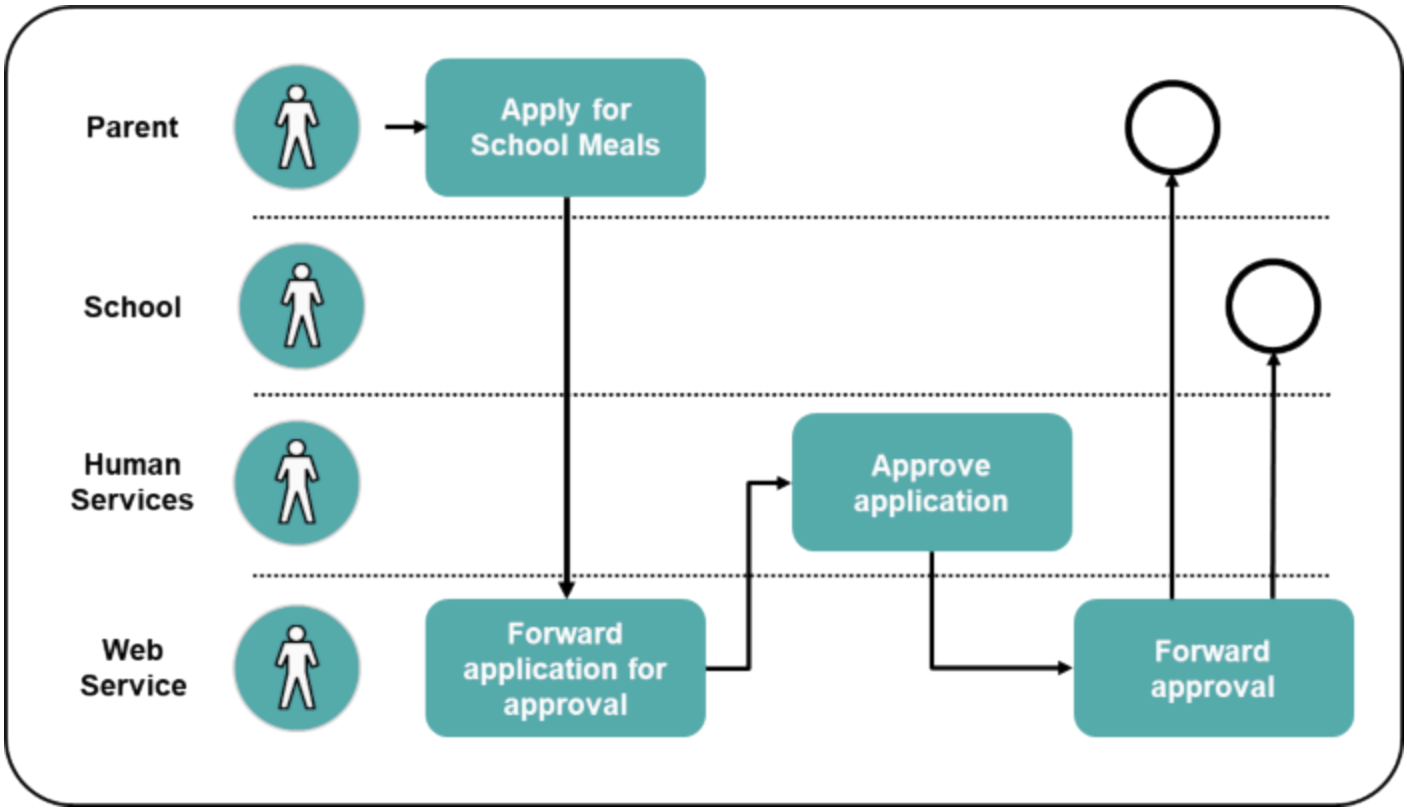
Business Case Example

Currently, the only means a city has to process school meal program applications is through paper-based applications at the city Human Services Agency office. Once approved, the Human Services Agency faxes approved applications to the particular school where the child attends. It is a primarily manual process that inhibits timely delivery of citizen services.

The city government identifies the need to create a series of information exchanges:

- Exchanges that allow resident parents to apply online for their children's school meals program.
- Exchanges that automate the processing of the parent's request between the Human Services Agency and the school.
- Exchanges that implement a web-based service that, upon submission, automatically checks the city's Human Services Agency database to ensure the applicant is from a low-income family and qualifies for the school meals program. If qualified, the web-based service notifies the respective school to add the child to the meal program.

Your scenario diagram may look like this:



Create Effective Requirements

Results can vary between individuals and groups. Examples of good requirements could be the following:

Candidate Message Exchange Packages

- Request from Parent to meal program eligibility system
- Eligibility reply from Human Services Agency to eligibility system
- Notification to add child to meal program from eligibility system to school
- Notification to add child to meal program from eligibility system to parent

Performance Requirements

- The web-based service notifies parents via email whether or not their child qualifies for the school meals program ten minutes after receiving their application.
- A school receives notification within 24 hours of a new qualified child within their school.

Reporting Requirements

Each school in the city receives a monthly report of every child in their school who is currently enrolled in the school meals program.

Data Quality Requirements

Data Quality Requirements should be taken into consideration once above requirements are discussed.

Map and Model

During the map and model phase, you compare your information exchange requirements to what's in NIEM, and then define a mapping between your requirements and NIEM content. You use a mapping document, which may be a spreadsheet or similar tabular form.

The mapping process consists of the creation of a mapping document and then searching for NIEM elements and mapping them in the document.

Your entries in the document identify the exchange requirements that are well-represented by NIEM, and what needs to be defined as an extension to NIEM.

In the following sections, we provide more detailed information on how you determine what you need to enter.

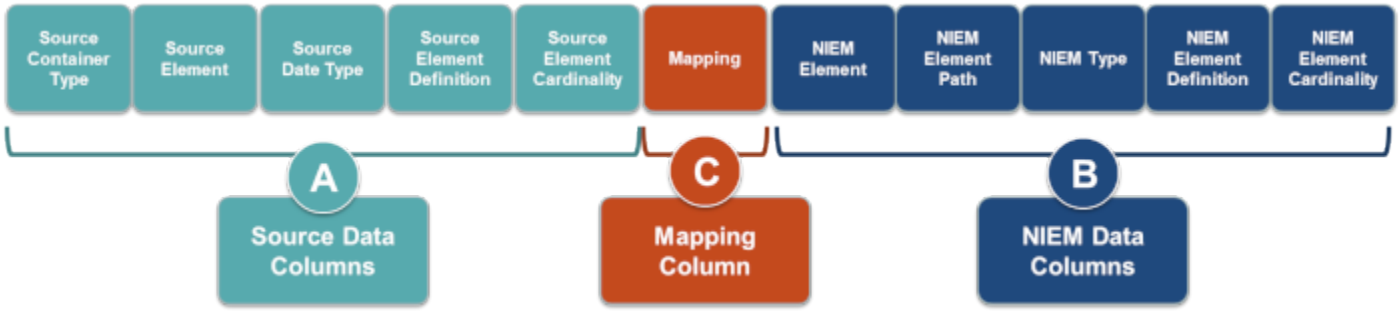
> This section will be undergoing changes to update the examples and screenshots to use the latest version of the mapping spreadsheet.

What is a Mapping Document

The Mapping Document has the following characteristics:

- Demonstrates where the existing exchange data (or the Exchange Content Model) objects map to NIEM data objects by recording the degree of similarity.
- Helps to identify data objects that are not currently within NIEM as candidates for an Extension Schema Document.
- Helps to communicate the level of reuse of NIEM to business users and owners.

The following illustration shows the recommended structure of a mapping document.



Mapping Document Setup

See the **Mapping Spreadsheet** page in the IEPD Artifacts section for more information about the spreadsheet and to download the template and examples.

In the following steps, identify (fill in) the **Source Data Columns**:

Source Data Columns (A)

1. Identify a data object within the existing exchange content model. This is the **Source Container Type**, which is the data source high-level object, class, or context of a set of data elements (e.g., person, vehicle, arrest) to which they belong.

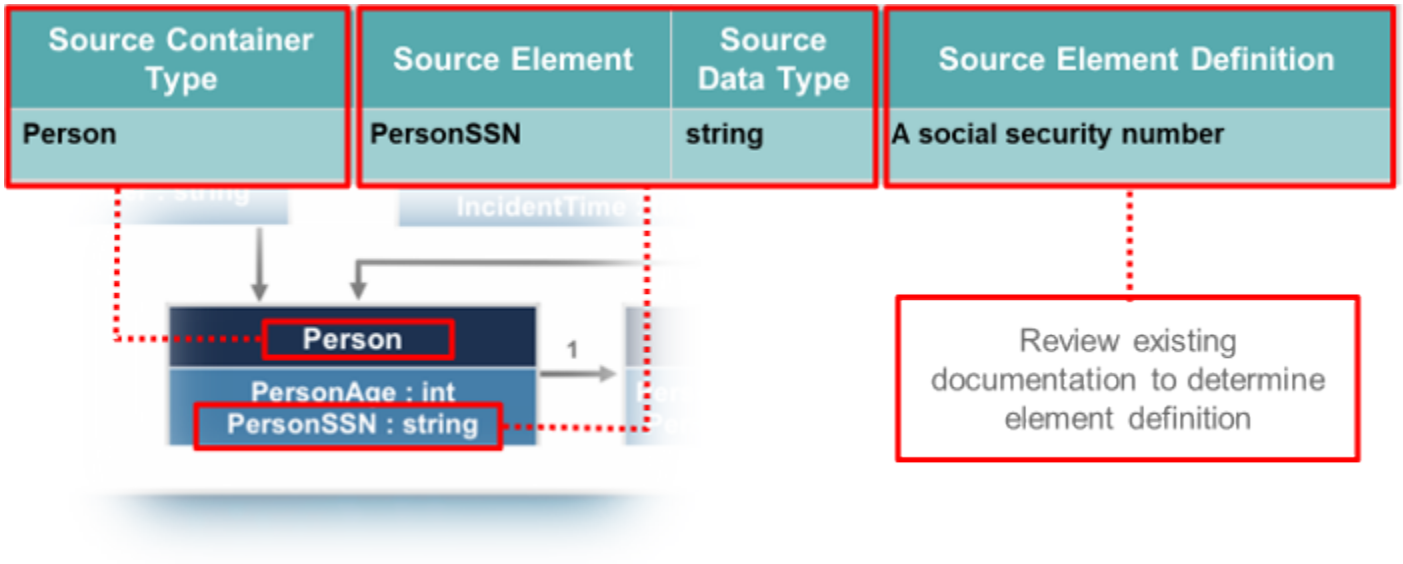
2. Identify an element within that object. This is the **Source Element**, which is a specific data element that is associated with the Source Container Type.

3. Identify the data type for the element. This is the **Source Data Type**, which is the data type (e.g., date, string) of the source element.

4. Provide a definition for the element. This is the **Source Element Definition**, which can be as descriptive as you want it to be.

5. Repeat the previous steps for each object in the Exchange Content Model.

Four columns should be filled in for every element within the exchange content model.



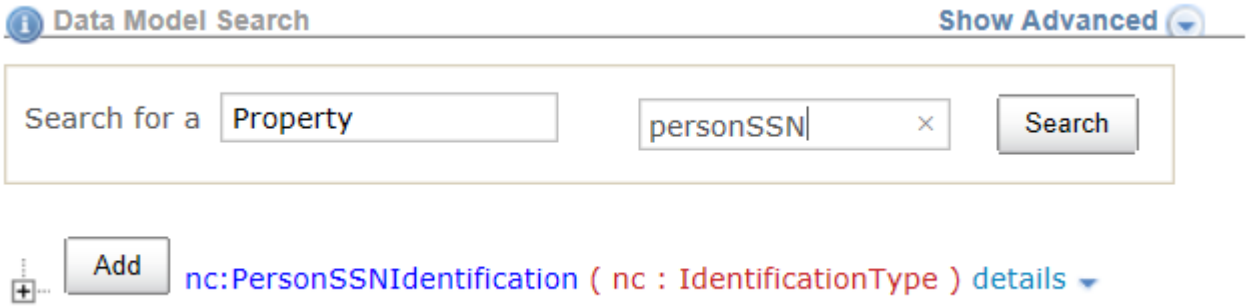
NIEM Data Columns (B) and Mapping Column (C)

After you complete the entries in the Source Data Columns, identify the NIEM elements that may have the same logical definition, semantics, and structure as the source elements.

1. Use a NIEM tool (e.g., Schema Subset Generation Tool (SSGT)) to find an element from a NIEM reference schema that "maps" to the exchange data element. In this example, look for a "Property" with "personSSN" as the search string. In general, a "Property" search is a good place to start.

2. If nothing is found that looks like a match, try searching by name, for synonyms, conceptual

meaning, or by type. In this example, you found a possible match.



3. Enter the match in the mapping document under "NIEM Element." Map objects that are conceptually and semantically equivalent. If the alignment is not easily understood, it is better to extend, which is explained in the schema build and validate process.

NIEM Element
nc:PersonSSNIdentification

4. Note the "type" associated with the NIEM element match and enter it under "NIEM Type."

NIEM Type
nc:IdentificationType

5. Open "details" next to the element for the "NIEM Element Definition." This is the standard definition for the NIEM element found in the element's metadata. Copy that information to the column.

NIEM Element Definition
A unique identification reference to a living person; assigned by the United States Social Security Administration.

6. Determine the **Mapping** value, which is the degree to which a source element maps to a NIEM element. This can be a personal judgment. The possible values are the following:

Value	What It Means
Equivalent	Semantics and structure map exactly or almost exactly. The NIEM data object's name, definition, and containing objects should have exactly the same conceptual meaning and logical alignment as the exchange data object.

Value	What It Means
Partial	For example, `Person` (exchange data) matches to `nc:Person` (NIEM data object) `Person First Name` (exchange data) matches to `nc:Person/nc:PersonName/nc:PersonGivenName` (NIEM data object) A partial match occurs when you can take some things from NIEM and extend the rest for a given Type. In other words, the semantics or structure of exchange and NIEM data objects do not exactly fit. Partial matches can usually be categorized into three conflict categories. Mitigations exist for each of these categories that maintain the integrity of the NIEM data model. **Semantic** - occurs when there is a small discrepancy in definition or name (e.g., `nc:Date` does not align exactly with `local-ns:EntryDate`). To mitigate, reuse the NIEM data type and create a local element. **Structural** - occurs when the NIEM data object has constraints that disallow a direct mapping of the exchange data object (e.g., data type mismatches; code lists (enumeration facets) do not have all the necessary entries). To mitigate, reuse the NIEM element, and either create a new data type or extend a NIEM base type. **Container** - occurs when an exchange data object hierarchy does not match the hierarchy of a desired NIEM data object (e.g., data object contains, or is contained within, elements or types that do not make logical sense or align well). To mitigate, reuse the NIEM element or type but create local elements or types to contain them.
No Match	No exchange element or type maps to NIEM. Typically this is due to a "new" government or private sector issue or data object that has not made it into the model, or you have uncommon data, such as something very specific to a particular organization. The result is an extension to NIEM through an IEPD extension schema.

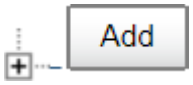
No Match - "Cargo Ship" has found a match that would otherwise be equivalent; except, it has a container or structural conflict. It has this conflict because "Hazmat Indicator" is not found under `nc:Vessel` in the NIEM data model. Therefore, "Hazmat Indicator" has no match, and "Cargo Ship" is only a partial match.

Exchange Data	Mapping	NIEM Data Objects
Cargo Ship	Partial Match	`nc:Vessel`
Hazmat Indicator	No Match	`nc:Vessel/local-ns:HazmatIndicator`

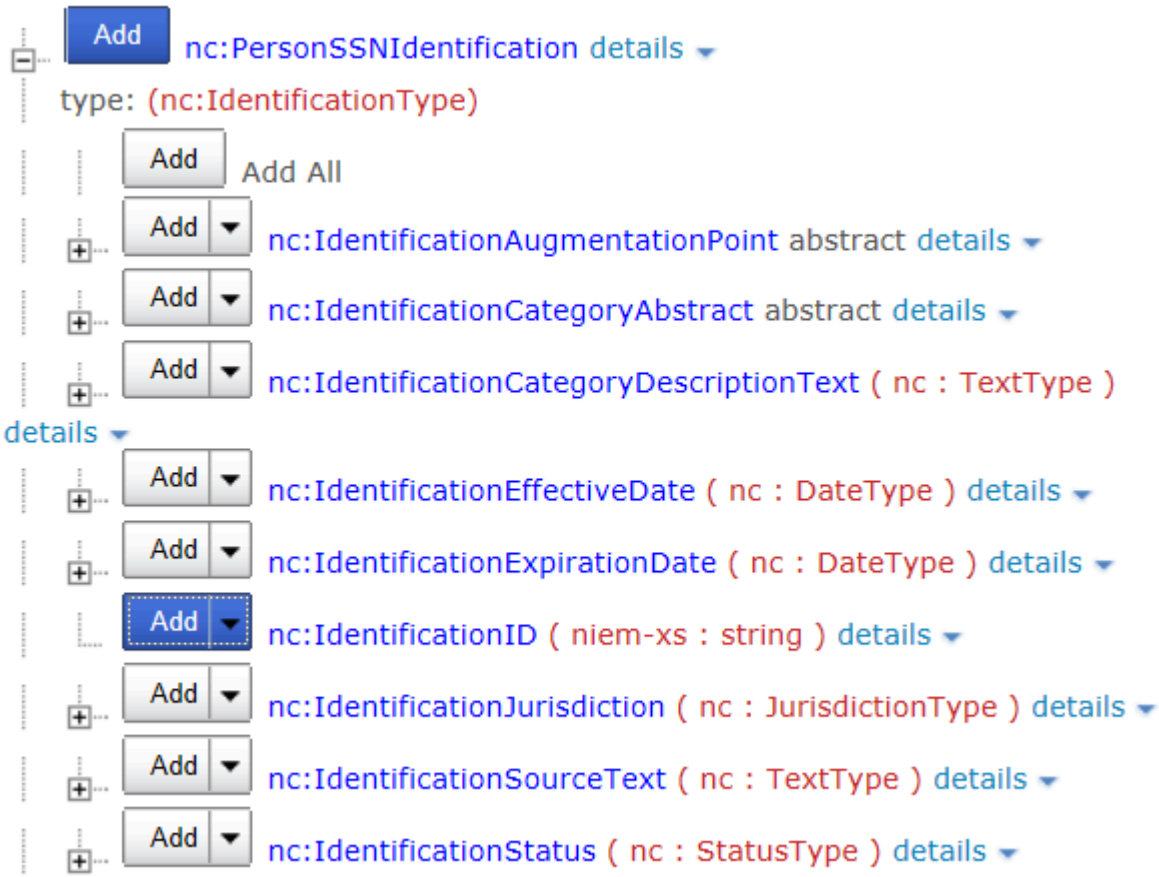
In this example, the match is equivalent (the best type of match). Enter "Equivalent" under "Mapping."

Mapping
Equivalent

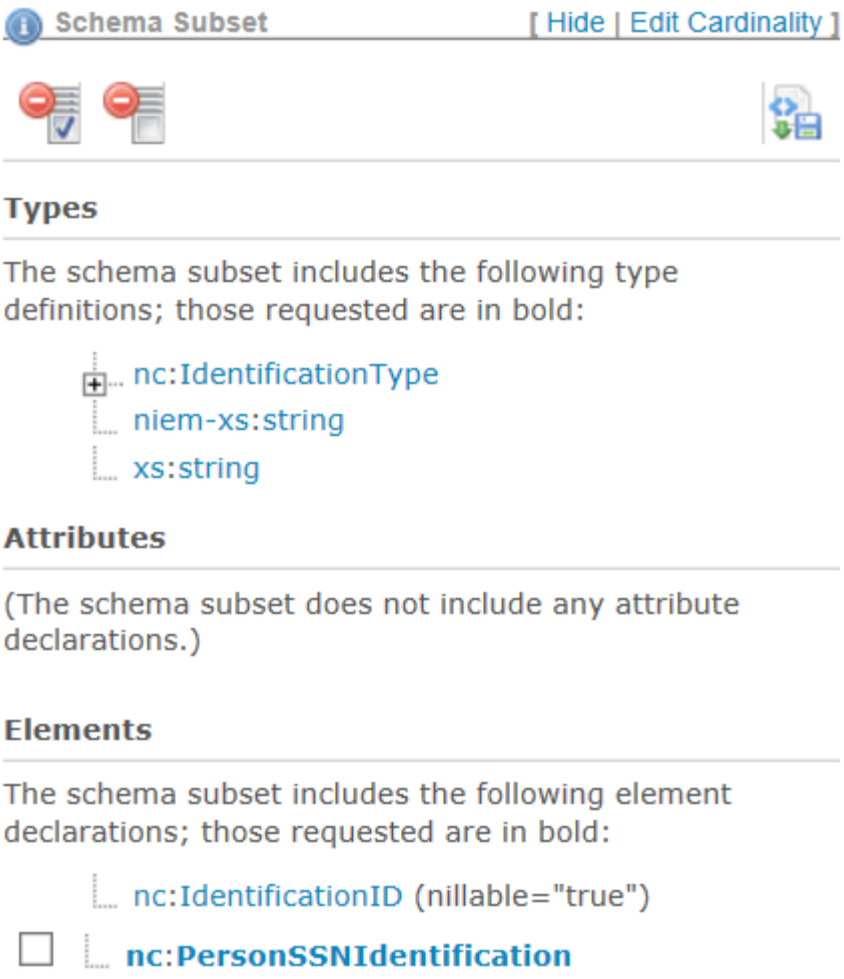
7. Click the expand button (next to **Add**) to open likely relationship trees until the terminal node is revealed.



8. Click the **Add** button next to the likely elements to begin the schema-creation process and to identify an element path.



9. Note the **Types** and **Elements** areas are populated when the **Add** button is clicked.



10. The **NIEM Element Path** is the path of the NIEM element within the NIEM model. Join the tree elements together to make the path and enter that in the column.

NIEM Element Path
nc:PersonSSNIdentification/ nc:IdentificationID

11. With the information you have now, you can begin the schema build and validate process.

Model Searching Tips

Searching the NIEM model can be difficult, but these methods can help. Note that the tactics that work using web search engines (like Google or Yahoo) also work with the search tools for the NIEM model.

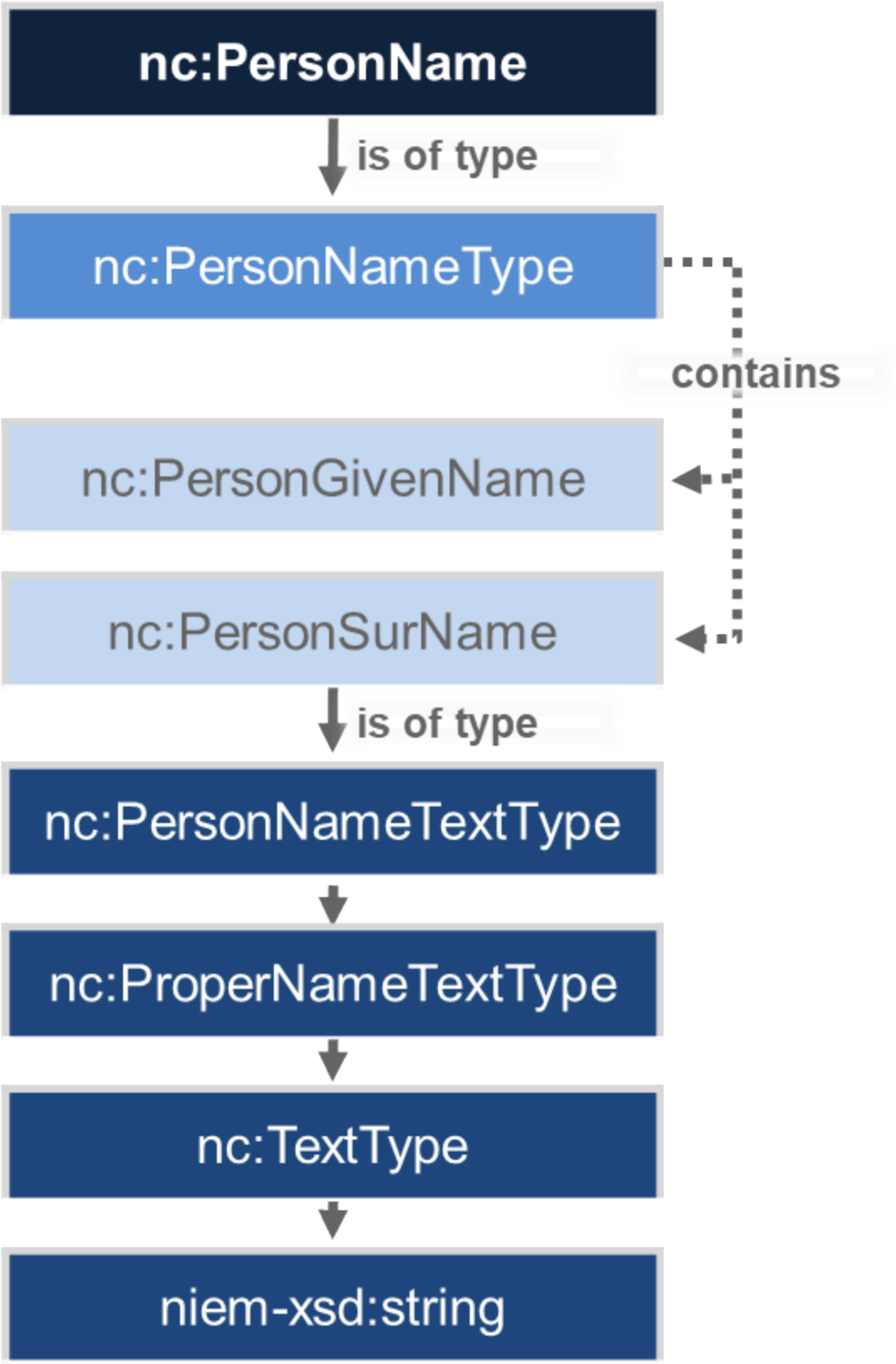
Search Method	Explanation	Example
Name	Searching for variations of a name can	Search for **Officer** instead of **Official** to yield

Search Method	Explanation	Example
Variations	often yield results.	`j:EnforcementOfficial`.
Conceptual Meaning	Search for words that could be in the definition of the data object.	Search for **Modified Charge** to yield `j:AmendedCharge`.
Synonyms	Synonyms of the data object to find exactly what you seek.	Search for **Facility** instead of **Building** to get different results.
Containers	Use a more abstract term to find containers of the data objects.	Search for **Person** instead of **Person Arrest** to get the maximum number of results.

NIEM Data Model Reuse Considerations

The hierarchical implications of the model must be considered when reusing NIEM elements. NIEM's hierarchal approach to modeling allows for flexibility and reuse, while still maintaining context.

- Elements and types are defined globally and referenced when used.
- Type hierarchy provides a contextual representation of the element.
- Attributes are assigned to some types to provide context on their use.



Additional Considerations

You should include additional notes in the mapping document to provide reasoning and context. When

mapping, be conservative in extending NIEM data objects to maintain the integrity of the NIEM data model.

Build and Validate

During the build and validate phase, you create XML schemas and artifacts and verify they meet NIEM standards.

Data objects identified in the mapping document appear in either the exchange, extension, or subset schemas. Associations and cardinality from the exchange content model are reflected in XML schemas.

NIEM-Conformant XML Schema Generation

XML schemas that are typically created for an IEPD include subset, exchange, extension, constraint, and reference. A NIEM-conformant IEPD is required to constrain at least one schema that is either a NIEM reference schema or subset schema. XML schemas for NIEM-conformant exchanges can be created in two ways:

Method	Process
Generation Through Tools	Automatically generate schemas based on an exchange content model, mapping document, or other inputs.
Coded by Hand	Start with existing schemas or NIEM schema templates, which can be derived from NIEM reference schemas.

It is recommended that you start with tools and modify as needed.

The NIEM Tools Catalog

The NIEM Tools catalog is a collection of different tools that aid in schema generation. For example, the Subset Schema Generation Tool (SSGT) is often used to start schema development because it is easy to use and produces subset schemas and wantlists for use within an exchange. The SSGT can also be used for searching and browsing the NIEM data model and generating Wantlists.

Look for an appropriate tool at the NIEM Tools Catalog.

Code List Artifact

A code list is a file that consists of comma-separated value (csv), tabular data, which makes the list suitable for an IEPD schema:

- Each row is a “distinct entry” or set of code values.
- Each cell in the row has a value identified by its column header.

The following table is an example of a code list for vehicle makes and models:

ExampleTemplateCode	
Example Code List Definition	
Code	Definition
FORD	Ford

ExampleTemplateCode	
HOND	Honda
TOYT	Toyota

- The first row is the type name (ExampleTemplateCode).
- The second row is the code set definition (Example Code List Definition).
- The third row is the column labels (Code, Definition).
- Subsequent rows are the codes and definitions.

Validate XML Schemas

XML Schema Conformance

Schemas must conform to predefined rules and guidelines as defined in the NIEM Naming and Design Rules (NDR) and Information Exchange Package Documentation (IEPD) Specification to promote increased levels of interoperability and reuse within exchanges. The NDR provides other general conformance guidelines and principles. XML schemas need to be checked for conformance both manually and through the use of tools because no tool currently exists that checks all of the NDR rules.

XML Instance Validation

Schemas must be validated to prove conformance. Validation is the process of testing XML to ensure well-formed and NIEM-conformant schemas. You can leverage NIEM tools to make the validation process easier. Each XML element must validate against the schema definition of that element. XML instances may be validated through multiple schema validation passes, using multiple schemas for a single namespace.

Resolve errors with tools as far as they are capable; the remainder must be resolved manually.

NIEM Conformance Tool

The NIEM Conformance Tool, based on the Conformance Testing Assistant (ConTesA), is an available option to validate schema. This tool allows you to upload an IEPD, a set of schemas in a zip file, or an individual schema, and provides a summary report that lists the NDR rules that were checked with those that passed and those that failed. The NIEM Conformance Tool does not check all NDR rules as some rules are subject to interpretation and then must be manually checked.

> The NIEM Conformance Tool is available for evaluation and feedback purposes only; it is not the measure for NIEM conformance. It assists developers by identifying potential locations of non-conformance within IEPD artifacts (e.g., schemas, catalog, xml) using the latest published NIEM specifications.

Resolve Validation Errors

The NIEM Conformance Tool provides a summary report that lists the location of schema failures by line number and by XML object name. You should then do the following:

1. Resolve each validation error by reviewing the location of the error within the schema and the NDR rule(s) that was broken for that error.

2. Make modifications to the appropriate schema to address each validation error.

3. Run each schema through the Conformance Validation Tool to check for NIEM conformance.

4. Repeat the previous steps until all conformance errors have been resolved.

Contact the NIEM Help Desk if you are unable to resolve conformance errors or need additional assistance.

Manual Review Process

Do a manual review of each schema to address the NDR rules not checked within the Conformance Validation Tool. Manual review of schemas for NDR conformance is always necessary to verify conformance. Use the following steps:

1. Review the summary report from the Conformance Validation Tool to identify the NDR rules not checked through the tool.

2. Analyze each schema for adherence to each of the NDR rules that were not checked within the Conformance Validation Tool.

3. Make the appropriate changes to the schema(s) to resolve each issue per mistakes made in the manual review.

4. Validate the schemas again in the Conformance Validation Tool.

Assemble and Document

Assemble and Document is the next phase in IEPD development after the Build and Validate phase.

You have all of your mandatory and recommended artifacts ready. Now you need to gather all of it in one electronic package.

Mandatory Artifacts Examples

Readme Artifact

As a starting point, logically organize the Readme document in chronological order of the development artifacts. Existing IEPD master documents can be used as a general guide. The Readme document can be a simple-text file and should not include XML code.

The following is a recommended, general-purpose table of contents.

1. Executive Summary
2. [Name] Information Exchange
    - Overview of the [Name] Exchange
    - Exchange Partner Interaction
3. Business Models
4. Business Rules and Requirement
5. Exchange Content Model
6. Development Information
    - IEPD Definition
    - Tools and Methodologies
    - Testing and Conformance
7. Appendices
    - List of IEPD Artifacts
    - IEPD Catalog

Change Log Artifact

The Change Log can be a simple-text file.

Version	Date	Description	Author
1.0	2/2/03	Original Version	Bob
1.1	4/16/03	Added new elements to subset schema.	Bill
2.0	5/9/18	Updated requirements and constraint schema.	Sara

Assemble the IEPD

Once all the artifacts have been compiled, the next step is to properly name, file, and archive the final package.

A standardized file structure promotes consistency and creates logical navigation through a large number of IEPD artifacts. Moreover, this consistency enables a greater degree of discovery and IEPD reuse because you can easily access information in a structured and uniform manner. Note that the catalog enables IEPD developers to locate artifacts in a directory and label them as specific IEPD artifacts no matter where they exist in the package.

1. **Name** each artifact for your exchange with a consistent naming convention.
2. **File** each artifact in a logical, consistent folder structure.
3. **Archive** (e.g., zip) the file structure, including artifacts, into the final package.

The following example shows a recommended IEPD folder structure and file location.

The documentation produced by the SSGT creates some of the structure for you.

The root directory name should be meaningful and include the NIEM version and a revision number.

my-iepd-4.0-rev-04

- base-xsd
  - extension
    - *extension.xsd
  - niem
    - niem-core/4.0
    - proxy/xsd/4.0
    - utility
      - appinfo/4.0
      - conformanceTargets/3.0
      - structures/4.0
    - wantlist.xml
    - xml-catalog.xml
  - xml-catalog.xml (references extension uri and nextCatalog niem/xml-catalog.xml)
- documentation (miscellaneous, binaries)
- iep-sample (contains sample xml instances; must be in root directory)
- iepd-catalog.xml (must be in root directory)
- changelog.txt (.md, .htm, .pdf; must be in root directory)
- readme.txt (.md, .htm, .pdf; must be in root directory)
- conformance-assertion.txt (.md, .htm, .pdf; should be in root directory)
- schematron (if Schematron is used; should be in root directory)

Review the IEPD

You should perform a peer review of your IEPD. This has several benefits:

- Maintains consistent quality of artifact content and presentation.
- Decreases the occurrence of published IEPDs that are difficult to reuse.
- Peer participation in a review also increases individual knowledge of information exchanges and promotes accountability in verifying that the exchange meets established requirements.

A good way to review an IEPD is to compare it to a checklist. The checklist should include the following:

- The readme document contains documentation necessary to effectively describe the information exchange.
- The exchange content model file is included and accurately represents the XML code in the schemas.
- The XML catalog validates with the NIEM Information Exchange Package Documentation (IEPD) catalog schema (XSD) and resides in the root directory of the IEPD bearing the file name “iepd-catalog.xml.”
- The change log is in the root directory of the IEPD, begins with the substring “changelog,” and records changes to previous IEPD schemas that are represented.

Publish and Implement

The publish and implement phase is the last phase of IEPD development.

The search, discovery, and reuse of IEPDs are necessary for effective information exchange across organizations. IEPD publication allows other organizations to search for needed information without having to recreate it.

IEPD Repository

Publish your IEPD to an appropriate NIEM Community repository.

IEPD Implementation

Implement a push or pull service for other interested parties to obtain your exchange.

Security and Privacy Considerations

NIEM does not dictate how you handle security and privacy issues. You must use your organization's standards.

You can tag data with security and privacy, however, other technologies are required upon exchange implementation to enforce security and privacy rules.

You also may use metadata to describe specific requirements in regard to information security and the handling of sensitive, privacy-protected information. The metadata allows systems that implement NIEM to automatically enforce rules that govern the use, protection, dissemination, and access controls for data being shared. An example of this is used by the Intelligence community, which established the Intelligence Community Information Security Marking (IC-ISM) as a standard for classified information. The NIEM technical architecture provides integration capabilities to leverage IC-ISM, as well as other IC security markups such as Need to Know (NTK).

Download and refer to the SuperHero ISM IEPD for an example on how to apply the IC-ISM markings.

Simple IEPD Tutorial

This tutorial shows you how to create a very simple IEPD. Once you are comfortable with the process and output, you can move on to more complex IEPDs.

In this tutorial we are going to walk through the creation of a simple IEPD using the IEPD Lifecycle. These steps include:

1. Scenario Planning - decide what information to include in your exchange
2. Analyze Requirements - break down business context and data requirements
3. Map and Model Requirements - begin mapping your data to NIEM components
4. Build and Validate - create a set of exchange-specific, NIEM-conformant XML schemas
5. Assemble and Document - package all files into a self-contained archive file
6. Publish and Implement - publish the IEPD for search, discovery, and reuse

---

Scenario Planning

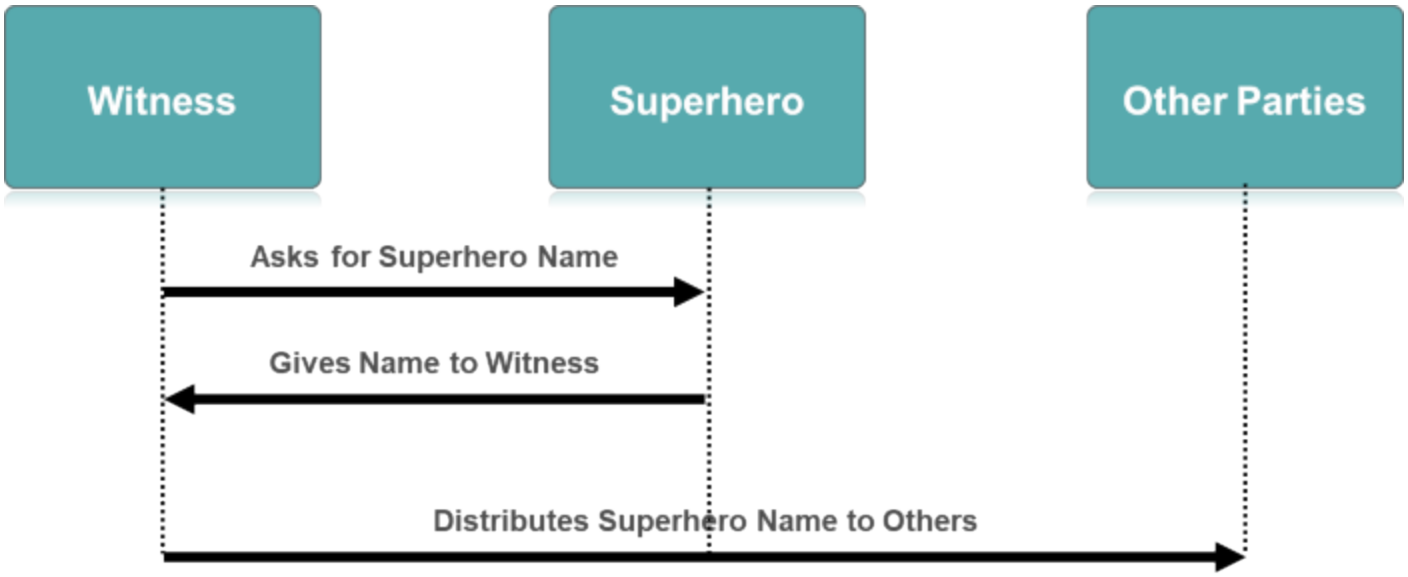
Scenario Planning is the first step in the IEPD development process. In this step you develop a high level scope and goal of the intended information exchange. Decide what information needs to be included.

You observe a person who displays superhuman powers (a SuperHero). As a government official tasked with tracking super powered individuals, you want to inform others of this person's presence and unique characteristics. You talk to personal and professional associates and decide the information should be distributed so that others may learn, or be able to add to the group's knowledge about the SuperHero.

Determine the information that would be useful to gather and the resources needed to obtain and distribute the information. To help visualize the information flow and content, you can develop a use case, business process, or sequence diagram to graphically show the actors and information flow for your scenario. A visual representation can be useful especially if the scenario is complex.

You decide that acquiring and distributing the super powered person's name is enough.

> The NIEM data model is very large. It is highly likely that the type and format of the data you want to acquire and distribute is already defined. If not, then one of the several NIEM Communities may have tackled a similar problem and developed an information exchange package (IEP) that comes close to meeting your needs. A search through their published resources can help.
>
> Scenario Information Acquisition and Flow
>
> 1. Witness sees a SuperHero display powers.
> 2. Witness obtains the SuperHero's name.
> 3. Witness distributes information to other parties using an IEPD.
>
> The information flow is illustrated in the following diagram:



Answer a few questions in a short quiz.

Analyze Requirements

After Scenario Planning, you want to analyze the scenario's requirements for the next step in the IEPD development process.

The information exchange scenario now needs to be broken down into finer detail to understand and document the business context and data requirements. There is no prescribed way to to do this, nor is the knowledge of NIEM or XML Schema required. The most important idea to keep in mind is that the subject matter experts capture the requirements and analysis with thorough detail.

The first item you need to determine is the single, main, focal point or purpose for the exchange. This is technically known as the root element. For this simple IEPD, that is the SuperHero.

> root element = `Person`. Assume the SuperHero is a human person.

The content portion of the requirements analysis can be done however you prefer: document, spreadsheet, or model diagram. The kinds of information to capture include the following:

- Data elements - These are the data fields for the message. For each field, the following information may or will be needed:
    - Element name - This is the name of the field or tag. Examples include First Name, Last Name, Street Address, and Phone Number.
    - Data type - The structural representation or format of the element. Examples include string, date, integer, decimal, Boolean, state code set, person data type, location data type.
    - Definition - Descriptive definitions will be required for all components in the exchange. At this stage, the semantics need to be captured, but the wording does not have to be finalized or perfected. If you determine later that an element maps to NIEM, then the

NIEM element will come with its own definition. The definition captured here should be sufficient enough so that the meaning is clear.

- Occurrence constraints - The minimum and maximum number of times an element may appear in the instance. For example, Last Name may occur once only; Social Security Number (SSN) may occur 0 to 1 times; Phone Number may occur 0 to many times.

  Default occurrence constraints in NIEM are 0 to unbounded (with a few exceptions); the default in XML Schema is once only. Without explicitly setting these occurrence constraints, the defaults mean that components reused from NIEM will be optional and the ones added locally will be required.

- Source information - Optional. It may be useful or necessary in some cases to record the source of the requirement. Traceability information may seem very apparent at the time but can be difficult to reconstruct at a later date if needed.

If a corresponding NIEM element is already known at this stage, then the NIEM values may be used in place of local ones for element name, data type, and definition.

- Objects and classes - These are the complex, reusable data structures where related elements are grouped together. Examples include Person (with elements First Name, Last Name, SSN, and Phone) and Location (with elements Street Address, City, State, and Zip Code).
- Code sets - This is a list of allowable values, such as a state code list or an eye color list. Code sets may come from a standard or may be custom-defined.
- Conditions and business rules - There may be certain restrictions on the content that need to be represented. Examples include restrictions on individual values (such as a number that cannot be less than zero or a string that must have a certain number of characters) and conditional restraints (such as if the SSN is not provided, then Last Name and Birth Date are required), etc.

The other requirements of the exchange (e.g., technical, security and privacy, performance, reporting) should be described in this step as well.

Answer a few questions in a short quiz.

Map and Model Requirements

After you have analyzed and determined your exchange requirements, you proceed to map and model them for the next step in the IEPD development process.

A common way begin the creation of IEPD components for your exchange is to create a mapping document. This is typically a spreadsheet, which maps your local exchange data elements to the NIEM data model.

See the **Mapping Spreadsheet** page in the IEPD Artifacts section for more information about the spreadsheet and to download the template and examples.

The Schema Subset Generation Tool (SSGT) is a good tool to use to map your exchange to NIEM. If you are unfamiliar with the SSGT, refer to Map and Model Training, “What is a Mapping Document.”

The SSGT's advantage lies in that you can extract just what you need from NIEM, i.e., create a subset.

> To find matches for your local components, use common words or acronyms in the search box. Local names will be much less likely to return results. A search term like "FirstName" or "First_Name" will return empty because these exact terms do not appear in any NIEM names or definitions; a search for "First Name" will return the matching component, `nc:PersonGivenName`, based on a match in the definition.
>
> SSGT search results are sorted by namespace (e.g., **hs:**, **j:**, **nc:**, and so on). The results list can be very long. Take your time looking through it.
>
> 1. A search of "Property" for "Person" shows `nc:Person` which is of `nc:PersonType`.
> 2. Browse through `nc:PersonType`, and you see it contains `nc:PersonName` which is of `nc:PersonNameTextType`.
> 3. Browse through `nc:PersonNameTextType`, and you see it contains `nc:PersonGivenName` and `nc:PersonSurName`. These look like they should fit with our model and complete our search for the time being.

We have enough information from the preceding example to fill in a mapping document. Make certain you have your SSGT searches handy so you can fill in the spreadsheet.

Source Data Columns

- Source Container Type - Person
- Source Element - SuperHero; Given Name; Surname
- Source Data Type - string
- Source Element Definition - a superhero who is also a human being; a superhero's name (first name); a superhero's name (family name)
- Source Element Cardinality - one person; one name; one name

NIEM Data Columns

- NIEM Element - `nc:Person`; `nc:PersonGivenName`; `nc:PersonSurName`
- NIEM Element Path - `nc:Person`; `nc:PersonGivenName`; `nc:PersonSurName`
- NIEM Type - `nc:PersonType`; `nc:PersonNameTextType`; `nc:PersonNameTextType`
- NIEM Element Definition - A human being.; A first name of a person.; A last name or family name of a person.
- NIEM Element Cardinality - 1..1 (at least one, but no more than one)

Mapping Column

- Mapping - Equivalent

Simple Mapping Example

Your mapping document should look like this.

Source Container Type	Source Element	Source Data Type	Source Element Definition	Source Element Cardinality	Mapping	NIEM Element
Person	SuperHero	string	a superhero who is also a human being	one person	Equivalent	nc:Person
Person	Given Name	string	a superhero's name (first name)	one name	Equivalent	nc:PersonGivenNan
Person	Surname	string	a superhero's name (family name)	one name	Equivalent	nc:PersonSurName

Generate Subset Schema

For the final step in this phase, use the SSGT to generate your new subset schema documentation:

1. Click "Generate Documents."
2. Under "Generate Subset Schema," click "Save Subset Schema to a file."
3. Under "Generate Wantlist," click "Save current wantlist to a file."

The Subset Schema and the Wantlist will form a substantial part of your IEPD. We'll be using these files in the next step, so remember where you save them.

Answer a few questions in a short quiz.

Build and Validate

You create and validate a set of exchange-specific, NIEM-conformant XML schemas that implement the exchange content model created for the exchange and validate them. Components in this phase also include other XML documents generated from NIEM tools (e.g., Wantlist).

Obtain the documentation the SSGT created for you in Map and Model. This comprises much of your IEPD for the simple exchange, in particular the schema subset.

Create a root directory folder for your IEPD. The name should be meaningful and include the NIEM version and a revision number. For this tutorial we'll name our root directory folder **superhero-**

iepd-4.0-rev-01. Create a subdirectory under your root directory called **base-xsd**. Place the subset schema and wantlist files in the base-xsd folder like in the following example.

superhero-iepd-4.0-rev-01

- base-xsd
  - niem
    - niem-core/4.0
    - proxy/xsd/4.0
    - utility
      - appinfo/4.0
      - conformanceTargets/3.0
      - structures/4.0
    - wantlist.xml
    - xml-catalog.xml

For the purpose of this tutorial, the local exchange components all map to NIEM. In a more complex IEP, there will likely be components that do not map. These would become part of an extension schema that is included in an IEPD, and is discussed in another tutorial.

Schema Example

There is no automated way to generate the instance schema document. You can copy an existing example and alter it to suit your exchange.

With the mapping components handy, create an instance schema XML document so that it looks like the following (with values added to `nc:PersonGivenName` and `nc:PersonSurName`). Place this file in a subdirectory folder called **iep-sample** under the root directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<nc:PersonName xmlns:nc="http://release.niem.gov/niem/niem-core/4.0/">
  <nc:PersonGivenName>Bruce</nc:PersonGivenName>
  <nc:PersonSurName>Wayne</nc:PersonSurName>
</nc:PersonName>
```

Answer a few questions in a short quiz.

Assemble and Document

You prepare and package all related files for the IEPD into a single, self-contained, self-documented, portable archive file (e.g., zip) according to the recommended file-and-folder structure. The following example shows where to place relevant artifacts under the root directory.

superhero-iepd-4.0-rev-01

- base-xsd/
  - niem/

        - xsd/
            - adapters/
            - codes/
            - domains/
            - utility/
            - niem-core.xsd
            - wantlist.xml
            - xml-catalog.xml
    - iepd-catalog.xml (must be in root directory)
    - changelog.txt (.md, .htm, .pdf; must be in root directory)
    - readme.txt (.md, .htm, .pdf; must be in root directory)
    - iep-sample (contains sample xml instances; must be in root directory)
    - conformance-assertion.txt (.md, .htm, .pdf; should be in root directory)
    - documentation (miscellaneous, binaries)
    - schematron (optional; should be in root directory)

Additional artifacts are required in an IEPD. A iepd-catalog is required and must conform to the IEPD Specification. Other artifacts such as documentation, sample instances, and schematron rules will not be covered in this tutorial. Schematron rules are not required but can be used to ensure the IEPD is following required business rules.

IEPD Artifacts:

- **NIEM schema subset** (required) - The output from the SSGT from the Map and Model step.

- **ied-catalog.xml** (required) - A specially-formatted XML document that contains metadata about the IEPD.
    - Catalog metadata includes:
        - IEPD unique identification
        - Conformance targets
        - Basic information about the IEPD
        - Key artifacts and directory structure
        - Relationships to other IEPDs and their artifacts

- **changelog** (required) - An artifact that describes the changes applied to the IEPD since its previous version. You may choose your own format for the changelog which can simply be the release date.

- **readme** (required) - An informal documentation artifact that includes an initial description or instructional information. This artifact should describe the IEPD purpose, scope, business value, exchange information, typical senders/receivers, interactions, and references to other documentation.

- **iep-sample** (required) - A sample xml instance that serves as a test for the IEPD schemas. The sample instance should contain realistic data and use as many data components and validity constraints as possible.

- **conformance-assertion** - An artifact that provides a declaration that an IEPD conforms to relevant NIEM specifications and associated rules, including NIEM Conformance 5.0, NIEM

Naming and Design Rules 5.0, NIEM Conformance Targets Attribute Specification 3.0, and NIEM IEPD Specification 5.0.

- **xml-catalog** - An xml instance that describes mappings between external schema references and locally-cached equivalents. A basic xml-catalog will be generated with the subset and can be modified as necessary.

- **wantlist** - An xml file that contains the elements and types from NIEM that will be included within the subset schema for the exchange. In other words, it describes what an exchange "wants" from the NIEM data model. Output from the SSGT during the Map and Model step.

Once you have assembled all required artifacts, compress the root directory folder **superhero-iepd-4.0-rev-01** to a self-contained archive (e.g., zip) file. You should then perform a peer review to ensure artifact consistency within the IEPD and with other IEPDs.

Publish and Implement

Once all artifacts have been assembled and you have compressed your IEPD to a single archive file, you can implement the IEPD into production and publish the IEPD for search, discovery, and reuse.

Publish an IEPD to all repositories that are relevant to the exchange. Many of the NIEM Communities have IEPD repositories where you can search for a package that is suitable or adaptable to your exchange. You may be able to find a home for your exchange with one of them.

You may also wish to publish your IEPD on a service such as github for other interested parties to obtain your exchange.

Additional Resources

- NIEM Conformance 5.0
- Information Exchange Package Documentation Specification 5.0
- NIEM Naming and Design Rules 5.0
- Optional IEPD Artifacts

IEPD Implementer

An IEPD implementer exchanges information based on the format, rules, and guidance provided by an existing IEPD specification.

- In this section
- Artifacts for IEPD Implementers
- IEPD Implementer Tools and Specifications
- Other resources

In this section

Below is a list of general resources for IEPD implementers.

- Implementation Cookbook

  A catalog of techniques and methods that help to implement NIEM.

- IEPD Java Bindings

  This is a Java project on GitHub that uses Maven and JAXB to generate a jar of java class files based on an IEPD that can be used to read and generate instances.

Artifacts for IEPD Implementers

Artifact	IEPD Implementer Uses
Code Lists	Interpret static or run-time code list bindings. Specify run-time bindings.

IEPD Implementer Tools and Specifications

The goal of NIEM is to provide a common data model, architecture, and framework to support more consistent, reusable, and scalable information exchanges. By the time a conformant IEPD is published, it should already:

1. Have all of the NIEM-specific rules and guidances about developing the data model and IEPDs built in, and
2. Capture and fully represent all of the local exchange requirements.

The IEPD should include such things as schemas to enable message validation, well-named and well-defined components, and sample instances and other documentation that demonstrate and describe the format and requirements of the exchange. This bundle of artifacts does not require any NIEM-specific tools or specifications to implement - NIEM is independent of any particular software or environment.

See the NIEM Tools Catalog for information about community and commercially-available tools.

Other resources

Please see other sections on this site for additional information:

- IEPD Starter Kit

  Learn about NIEM IEPD conformance, view sample IEPDs, and more.

- Artifacts

  Learn about the kinds of files that make up NIEM releases and IEPDs.

Implementation Cookbook

A catalog of techniques and methods that help to implement NIEM.

When you're creating an information exchange, you rarely ever use NIEM by itself. Depending on business requirements, there are other aspects of information exchange implementations that are required, including access controls, policy automation, transmission protocol, and others.

Based on our NIEM implementation experience, we've identified some common patterns (or "recipes," so to speak) to implement NIEM with other pieces of the information exchange puzzle.

The implementation cookbook serves as a single location for the community to learn and share reusable solutions for implementing NIEM.

With your help, our goal is for the cookbook to define the many ways NIEM-based exchanges can be used within specific environments, with various technologies and standards. This includes programming languages, data formats, query languages, and messaging frameworks.

Remember, NIEM and other data standards are independent of any particular software. However, ensuring NIEM is expressed, defined, and used in various software platforms better enables reusability, portability, and interoperability—which is what this effort is working toward.

- Avoid Replicating Data in a NIEM XML Document

  There are several NIEM-specific strategies to avoid replicating data within a NIEM XML document.

- Browsing NIEM Content

  There are several online resources that may help you find out what is in NIEM.

- Efficient XML Interchange

  Efficient XML Interchange (EXI) uses information in the XML Schema for an XML document to optimize compression of XML documents.

- Enterprise Integration Patterns

  Enterprise Integration Patterns (EIP) provide a way of thinking about and expressing integration between and across systems, languages, APIs, and databases.

- Fast Infoset

  Fast Infoset (FI) is a specification which provides for compression of XML documents.

- Server Output Filter

A server output filter can compress XML data being transmitted.

- Size of NIEM XML Documents

  Strategies to reduce or compress the size of NIEM documents.

IEPD Java Bindings

This is a Java project on GitHub that uses Maven and JAXB to generate a jar of java class files based on an IEPD that can be used to read and generate instances.

This project is meant to be an example of how one may go about creating a JAXB-based project for using Java to manipulate a NIEM based exchange. While we have attempted to accommodate all sorts of NIEM-based schema idioms, please understand that not all schema models will translate into perfect Java beans.

IEPD Java Bindings Project

Reference

NIEM is a data model that defines many reusable components and rules for building local extensions. The purpose for this is to be able to create information exchanges with clear semantics and a higher degree of consistency across the community.

This section represents our *work in progress* and is part of our training modernization. As we knock out content, we push it out to the user community for feedback and use.

- Artifacts
  Learn about the kinds of files that make up NIEM releases and IEPDs.
  Learn more
- Concepts for data modeling
  Learn about properties, types, augmentations, and other building blocks used to construct the model.
  Learn more
- Content
  A high-level perspective on the kinds of content that appear in Core and the domains.
  Learn more
- IEPDs
  An Information Exchange Package Documentation (IEPD) is a collection of NIEM artifacts. They define and describe the context, content, semantics, and structure of one or more implementable information exchanges.
  Learn more
- Releases
  A NIEM release is a versioned set of schemas and supporting artifacts published by the NIEM Management Office. Release schemas contain reusable properties and types meant for use in information exchanges. These schemas include NIEM Core, domains, and code tables.
  Learn more
- Specifications
  NIEM specifications provide rules and guidance in order to design consistent and well-defined information exchanges. Specifications are managed by the NIEM Technical Architecture Committee (NTAC).
  Learn more
- Tools
  NIEM tools provide support for searching and exploring NIEM content and developing NIEM domains and exchanges.
  Learn more
- NIEM Domain Modeling Guide
  How to design a Community of Interest
  Learn more

Artifacts

NIEM artifacts are individual files or packages relating to NIEM releases or information exchanges.

Information about release-specific and IEPD-specific artifacts has moved to the **Release Artifacts** and **IEPD Artifacts** pages.

- Code Lists

  NIEM Code Lists are special files that support the use of enhanced code lists in NIEM, beyond the basic capabilities provided by XML and JSON schema enumerations.

- JSON-LD Context

  A JSON-LD context file is a file that can be used to associate namespace prefixes with the full URIs that they represent.

- XML Catalog

  An XML catalog is a XML document that assigns locations to files. This can be used to override the file locations assigned by NIEM XML schema import statements without having to modify the original schema itself.

Code Lists

Code List Artifacts (Code Lists) provide a method to specify categorical descriptors for data used within exchanges and IEPs. Code Lists in NIEM use one of two formats: Comma-Separated Values (CSV), or Genericode.

- Overview
  - Quick Example
- Structure
  - Terminology
  - Comma-Separated Values (CSV) Files
- Specifying a Code List
- Binding in XML
  - Run-Time (Dynamic) Binding
  - Schema-Time (Static) Binding

- simple machine-readable formats (CSV or Genericode)
- ability to specify code list references at Schema-Time (static) or Run-Time (dynamic)
- XML catalog support for URIs to actual code list files
- rules for matching abbreviated values to corresponding codes specified within a code list file
- serves as the basis for IEPDs and Extension Schema Documents

Code Lists, while functionally identical to schema enumerations, do not replace the established convention or deprecate the use of schema enumeration.

Overview

In general, data exchanged between senders and receivers tend to be categorical or have distinct value ranges. One method of specifying that data within an exchange is by codifying them into NIEM Enumeration Facets. However, it might be easier to create a self-contained artifact to hold the associations. We call this artifact a Code List.

"Codes" is the general term NIEM uses to identify associative identifiers with more verbose data or data ranges (e.g., MON for Monday; 2000-3000 for sets of numbers).

Code Lists provide a tabular format for specifying *codes* and their associated data. Code Lists are most often used to limit possible values for an element. If Code Lists are reusable by many exchanges, they have the potential to be integrated into NIEM Core or a NIEM domain. An example of a code list in NIEM core is `nc:LocationCountryFIPS10-4Code`, which is used in the example below. The possible values for that particular element are country codes.

Quick Example

We can summarize Section 2.9, *Code Lists* as follows:

- A code list is a set of distinct entries with a corresponding set of columns.
- Think of it as a table: rows are distinct entries, columns are code list columns, and individual

cells are code values.

In this example of a code list for vehicle makes and models, note the following:

- The code list is the table
- The distinct entries are the rows (excluding the header)
- The code values are the entries of the individual table cells (e.g., **FORD**, **Toyota**, **Camry**, etc.)

Make code	Make description	Model code	Model description	Class
FORD	Ford	FUS	Fusion	Auto
FORD	Ford	F15	F-150	Pickup
HOND	Honda	CIV	Civic	Auto
HOND	Honda	CRV	CRV	SUV
HOND	Honda	ACC	Accord	Auto
TOYT	Toyota	COA	Corolla	Auto
TOYT	Toyota	CAM	Camry	Auto

Structure

The following sections provide details on the parts that make up a code list.

Terminology

binding
The correspondence between a set of data values, such as data within an XML document, and a set of columns within a code list

code list identifier
An absolute URI that identifies a code list

column identifier
An absolute URI in a Genericode code list document (GC-CLD) that identifies a column as a well-known column

column name
A string in a CSV file code list document (CSV-CLD) file that identifies a column as a well-known column

well-known column
A column in a CSV-CLD or GC-CLD that has well-understood semantics, i.e., the semantics are generally understood

#code
A binding that uses the column name “code” to refer to a well-known column

#range
A binding that uses the column name “range” to refer to range columns (columns that describe

ranges of values)

Comma-Separated Values (CSV) Files

A CSV code list document MUST be a CSV file with the following features:

- Has a CSV header.
- No column names are empty.

A CSV file may act as a CSV code list document when it has the following characteristics:

- **Code list identifiers** - the CSV file does not specify its code list identifiers. Each CSV file contains a single code list. The CSV file MAY be resolved to any code list identifier.
- **Distinct entries** - each CSV record of the CSV file constitutes a distinct entry.
- **Code values** - each CSV field corresponds to a code value.
- **Column names** - the column name of a code value is the CSV column name corresponding by position within the CSV header to the position of the CSV field within its CSV record.

A column in a CSV file is a well-known column when its CSV column name is the same as the column name of a well-known column.

A reference to column #code is matched, in order, to the following:

- a column with CSV column name of code, or
- the first column of the CSV file.

A code value in a CSV file has no type; its data type is empty.

Specifying a Code List

A code list document is a file or resource that contains one or more code lists. It is a conformance target of the Code Lists specification. A code list document MUST conform to all rules of this specification that apply to this conformance target. An XML document with an effective conformance target identifier of #CodeListDocument MUST be a code list document.

The specification for an implementation of a code list document by a resource class (such as with Genericode or CSV) needs to specify the following characteristics:

- Candidate code list identifiers. Some code list documents specify what URIs may be used to reference the code list. Some resources may feature multiple code lists within a single resource, and the specification must describe how a specific code list is identified within the resource.
- the mapping between features of the resource and distinct entries
- the mapping between features of the resource and code values
- the mapping between features of the resource and column names
- the method of identifying a column as being a well-known column (described in Section 7, Well-known columns and references)
- the method of identifying well-known column reference #code (described in Section 7, Well-known columns and references)

- the method of identifying the data type of a code value, which can be a data type defined by an XML Schema, or empty

These characteristics are described for Genericode files in Section 6, Genericode code lists. They are described for CSV files in Section 5, Comma-separated values (CSV) code lists.

A CSV code list document is a code list document. It is a conformance target of this specification. A CSV code list document MUST conform to all rules of this specification that apply to this conformance target. The conformance target identifier for this conformance target is #CSVCodeListDocument.

Binding in XML

XML content in a message may be identified as corresponding to content of a code list. This correspondence is referred to as a code list binding, connecting the XML content to the code list.

A code list binding is an assigned correspondence between a set of data values, such as data within an XML document, and a set of columns within a code list, identified via a code list identifier and a set of column names.

A code list binding has the following properties:

- a code list identifier
- a set of column/value pairs, each having
    - a column reference: either a column name or a well-known column reference, and
    - a data value
- a boolean value (constraining)

Run-Time (Dynamic) Binding

The example below shows an XML schema snippet that creates two code elements of type `nc:CodeType`. This type enables code list information to be provided in instance messages.

```
<xs:element name="VehicleMakeCode" type="nc:CodeType"
            substitutionGroup="nc:VehicleMakeAbstract">
  <xs:annotation>
    <xs:documentation>A code for a manufacturer of a vehicle.</xs:documentat
  </xs:annotation>
</xs:element>

<xs:element name="VehicleModelCode" type="nc:CodeType"
            substitutionGroup="nc:VehicleModelAbstract">
  <xs:annotation>
    <xs:documentation>A code for a model of a vehicle.</xs:documentation>
  </xs:annotation>
</xs:element>
```

The example below is an XML instance based on the schema snippet above:

Because the schema does not identify a URI or column name of a code set, this information

must be provided by the instance.

```
<?xml version="1.0" encoding="US-ASCII" standalone="yes"?>
<nc:Vehicle
    xmlns:cli="http://reference.niem.gov/niem/specification/code-lists/4.0/coc
    xmlns:ext="http://example.org/namespace/extension-run-time"
    xmlns:nc="http://release.niem.gov/niem/niem-core/4.0/">
  <ext:VehicleMakeCode
      cli:codeListURI="http://example.org/code-list/vehicle-make-model"
      cli:codeListColumnName="make"
    >DODG</ext:VehicleMakeCode>
  <ext:VehicleModelCode
      cli:codeListURI="http://example.org/code-list/vehicle-make-model"
      cli:codeListColumnName="model"
    >R15</ext:VehicleModelCode>
</nc:Vehicle>
```

Schema-Time (Static) Binding

The XML schema example below creates static, schema-time bindings between the elements and and CSVs. URIs and column names are used to indicate the particular set of valid code values.

```
  <xs:element name="Vehicle" type="nc:VehicleType"
              substitutionGroup="nc:Vehicle">
    <xs:annotation>
      <xs:appinfo>
        <clsa:ComplexCodeListBinding
          codeListURI="http://example.org/code-list/vehicle-make-model">
          <clsa:ElementCodeListBinding
              elementName="ext:VehicleMakeCode"
              columnName="make"/>
          <clsa:ElementCodeListBinding
              elementName="ext:VehicleModelCode"
              columnName="model"/>
        </clsa:ComplexCodeListBinding>
      </xs:appinfo>
    </xs:annotation>
  </xs:element>

  <xs:element name="VehicleMakeCode" type="niem-xs:token"
              substitutionGroup="nc:VehicleMakeAbstract">
    <xs:annotation>
      <xs:documentation>A code for a manufacturer of a vehicle.</xs:documentat
      <xs:appinfo>
        <clsa:SimpleCodeListBinding
            codeListURI="http://example.org/code-list/vehicle-make-model"
            columnName="make"/>
      </xs:appinfo>
    </xs:annotation>
  </xs:element>
```

```
  <xs:element name="VehicleModelCode" type="niem-xs:token"
              substitutionGroup="nc:VehicleModelAbstract">
    <xs:annotation>
      <xs:documentation>A code for a model of a vehicle.</xs:documentation>
      <xs:appinfo>
        <clsa:SimpleCodeListBinding
            codeListURI="http://example.org/code-list/vehicle-make-model"
            columnName="model"/>
      </xs:appinfo>
    </xs:annotation>
  </xs:element>
```

The example below is an XML instance based on the schema snippet above:

Because bindings to CSVs have been made in the schema (URI and column name), they do not appear in the instance.

```
<?xml version="1.0" encoding="US-ASCII"?>
<ext:Vehicle xmlns:ext="http://example.org/namespace/extension-schema-time">
  <ext:VehicleMakeCode>DODG</ext:VehicleMakeCode>
  <ext:VehicleModelCode>R15</ext:VehicleModelCode>
</ext:Vehicle>
```

JSON-LD Context

A JSON-LD context file is a file that can be used to associate namespace prefixes with the full URIs that they represent.

This is a required artifact when using NIEM JSON.

This is the same kind of information captured in XML via namespace prefix declarations. Without explicitly associating namespace prefixes to URIs, it would be left to future users to have to guess or try to reverse engineer which versions of NIEM schemas were used. Interpreting the meaning of NIEM messages should not involve guesswork.

- Reference property `nc:Location` in a JSON extension schema.
- Without assigning the prefix `nc` to the URI that identifies it, it could be from any version of NIEM Core - or from a different, non-NIEM namespace entirely.

- Reference property `nc:Location` in a JSON extension schema.
- Declare the namespace prefix in the JSON-LD context file:
    - `"nc" : "http://release.niem.gov/niem/niem-core/4.0/#"`
- This is enough information for users to be able to easily tell that property `nc:Location` is from NIEM Core version 4.0.

For a NIEM release, this file is located in the `jsonld-context` folder in the root of the package or repo.

This example is the beginning of the JSON-LD context file from the NIEM 4.0 release:

```
{
    "aamva_d20" : "http://release.niem.gov/niem/codes/aamva_d20/4.0/#",
    "ag" : "http://release.niem.gov/niem/domains/agriculture/4.0/#",
    "ag_codes" : "http://release.niem.gov/niem/codes/ag_codes/4.0/#",
    "apco" : "http://release.niem.gov/niem/codes/apco_event/4.0/#",
    "atf" : "http://release.niem.gov/niem/codes/atf/4.0/#",
    "biom" : "http://release.niem.gov/niem/domains/biometrics/4.0/#",
    "can" : "http://release.niem.gov/niem/codes/canada_post/4.0/#",
    "cbrn" : "http://release.niem.gov/niem/domains/cbrn/4.0/#",
    "cbrncl" : "http://release.niem.gov/niem/codes/cbrncl/4.0/#",
    "census" : "http://release.niem.gov/niem/codes/census_uscounty/4.0/#",
    "cjis" : "http://release.niem.gov/niem/codes/fbi_cjis/4.0/#",
```

XML Catalog

Overview

An XML catalog is a XML document that assigns locations to files. This can be used to override the file locations assigned by NIEM XML schema import statements without having to modify the original schema itself.

XML catalogs are an OASIS standard.

- Overview
- Template
- Using with NIEM CSV Code Lists
- Tool support

While XML catalogs provide a broader set of features that may be used as needed by developers and implementers for namespace resolution, the primary ways that NIEM uses XML Catalogs for releases are presented here.

A XML catalog has a root-level `catalog` element. NIEM releases use `uri` elements under the `catalog` element to associate namespaces with file locations. A subset of the XML Catalog file from the 4.0 release, xml-catalog.xml, is shown below.

- The `name` attribute holds the target namespace of the XML schema file
- The `uri` attribute holds the absolute or relative file location

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE catalog PUBLIC "-//OASIS//DTD Entity Resolution XML Catalog V1.0//EN
<catalog prefer="public" xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <uri name="http://release.niem.gov/niem/appinfo/4.0/" uri="utility/appinfo/4
  <uri name="http://release.niem.gov/niem/conformanceTargets/3.0/" uri="utilit
  <uri name="http://release.niem.gov/niem/niem-core/4.0/" uri="niem-core/4.0/n
</catalog>
```

Template

The template below may be used as a starting point for creating an XML catalog:

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE catalog PUBLIC "-//OASIS//DTD Entity Resolution XML Catalog V1.0//EN
<catalog prefer="public" xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <uri name="" uri=""/>
</catalog>
```

Using with NIEM CSV Code Lists

In addition to mapping target namespaces of XML schemas to file locations, XML catalogs can also be used to map a CSV Code List file to a source identifier.

The example catalog below maps:

- the target namespace of the NIEM Core 4.0 schema to a relative file location.
- the source identifier for GENC 3-7 3-character country codes to a relative file location for the CSV file.

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE catalog PUBLIC "-//OASIS//DTD Entity Resolution XML Catalog V1.0//EN
<catalog prefer="public" xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <uri name="http://release.niem.gov/niem/niem-core/4.0/" uri="niem-core/4.0/n
  <uri name="http://api.nsgreg.nga.mil/geo-political/GENC/3/3-7" uri="codes/ge
</catalog>
```

Tool support

XML Catalogs are supported by many common XML tools, such as Oxygen XML Editor, XMLSpy, and Xerces. See the following for more information about XML catalogs.

- Using XML Catalogs with Oxygen

  Use XML catalog files in Oxygen to specify or override schema file locations.

- Using XML Catalogs with XMLSPy

  Use XML catalog files in XMLSpy to specify or override schema file locations.

Concepts for data modeling

Learn about properties, types, augmentations, and other building blocks used to construct the model.

- The Basics
  - Namespaces
  - Properties
  - Types
  - Code Sets and other Facets
- NIEM-Specific Techniques

The concepts below have NIEM-specific representations in XML and JSON. Each section has information that demonstrates how to define these concepts in a schema and how sample data would appear in an instance.

> For a basic introduction to XML or JSON, there are many resources available online, including the W3 Schools XML Tutorial and XML Schema Tutorial, and the json-schema.org tutorial Understanding JSON Schema.

The Basics

Fundamentally, a NIEM release consists of a set of namespaces, each of which define properties and types (referred to more generally as components). An IEPD reuses components from a release that meet its requirements, and creates new properties and types in local namespaces as needed. These namespaces, properties, and types are basic building blocks of NIEM.

Namespaces

Learn about namespaces in NIEM and see schema examples and templates with header information at the XML and JSON links below.

Namespaces	A namespace is a collection of properties and types, managed by a common authoritative source.	XML	JSON

Properties

A property represents a concept, idea, or thing.

Properties	A property represents a concept, idea, or thing.	XML	JSON
Elements	An element represents a concept. In an instance, it acts as a container that may carry either a simple value or an object, and optionally attributes.	XML	JSON
Abstract Elements	An abstract element is an element defined as a placeholder in a schema that must be replaced with an appropriate substitution in an instance.	XML	JSON
Substitutable	A substitutable element is one that can replace another element in	XML	JSON

Elements	an instance.		
Attributes	An attribute is a property that may carry a simple value only.	XML	JSON

See the information in the **Property** page for guidance on **Naming**, **Definitions**, and **Representation terms + Definition Standard Opening Phrases**.

Types

A type represents a data structure that defines a set of allowable values.

Types	A type represents a data structure that defines a set of allowable values.	XML	JSON
Complex Types with Complex Content	A complex type with complex content (CCC) is a structure that represents an object and may contain elements and attributes.	XML	JSON
Complex Types with Simple Content	A complex type with simple content (CSC) is a structure that represents a simple value and that may optionally contain attributes.	XML	JSON
Simple Types	A simple type is a structure that represents a simple value only.	XML	JSON

Code Sets and other Facets

Facets let you constrain the allowable values for a string, number, date, or other simple type. Codes (enumerations) are the most commonly used facet in NIEM. Other facets include patterns, length, and maxExclusive.

Codes / Facets	A facet is a constraint on a simple type that limits the set of allowable values.	XML	JSON

NIEM-Specific Techniques

In addition to the basic concepts described above, NIEM defines or uses additional concepts to reflect its conceptual model, improve reusability, and support well-defined information exchanges. These topics are defined by the NIEM Naming and Design Rules (NDR).

Adapters	An adapter is a mechanism to use components from a non-conformant external standard within a NIEM-conformant namespace.	XML	JSON
Associations	An association is a complex relationship between objects, with optional additional characteristics.	XML	JSON
Augmentation Points	An augmentation point is a special substitution group to allow for later replacement by additional content from other namespaces.	XML	JSON
Augmentation Element	An augmentation element is additional content that replaces an augmentation point defined in another namespace.	XML1 XML2	JSON
Local Terminology	Local terminology is a word, phrase, abbreviation, acronym, jargon, or other string of characters specially documented in a namespace	XML	JSON

	because no definition or literal exists in a standard dictionary.		
Metadata	Metadata is a set of data that describes characteristics of other data.	XML	JSON
References	A reference is used to link to content defined elsewhere.	XML	JSON
Roles	A role is a function or responsibility of someone or something.	XML	JSON
Representations	The representation pattern is a mechanism to support multiple representations of a concept, along with additional properties.	XML	JSON

Namespaces

Namespaces act like collections that logically group related properties and types. Namespaces represent content from an authoritative source or standard, such as a domain or the NIEM Business Architecture Committee (NBAC), which collaboratively manages the Core namespace. They also have additional characteristics, like a target namespace URI (a unique ID) and a namespace prefix (used as shorthand for the URI).

- Overview
  - Categories
  - Persistence
- Modeling guidance
  - Identifier
  - Prefix
  - Version
  - Definition
  - Language - [new in 5.0]
  - Conformance Targets
- XML
  - Full Templates
  - Parts of the template
    - URI, Version, and Language
    - xsi:SchemaLocation
    - Conformance Targets
    - Namespace prefixes
    - Definition
    - Imports
  - NIEM's profile of XML Schema
- JSON
  - Schema template
  - Schema example
  - JSON-LD Context
- NDR references
  - General-purpose
  - XML-specific

Overview

Components are often referred to by their qualified names, like `nc:Person`. Using the namespace prefix in front of the component name helps to identify and distinguish it, especially in cases where the same name may appear in multiple namespaces.

- The `Human Services` domain is a namespace.
- The full identifier (`targetNamespace` attribute in XML) for this namespace in NIEM 4.0 is `http://release.niem.gov/niem/domains/humanServices/4.0/`
- In the 4.0 release, it uses the namespace prefix `hs` as shorthand for the full namespace

identifier and defines components like property `hs:RequestedService` and type `hs:PlacementType`.

NIEM primarily creates namespaces in order to group components together based on governance.

- Core contains content typically too general to belong to any one domain, so it is managed by the NIEM Business Architecture Committee (NBAC), a governance body with cross-domain and community representation.
- Domain namespaces are managed by governance bodies in specific subject areas.
- Code namespaces typically reflect standards managed outside of NIEM.

A domain or IEPD may reuse any component from any NIEM release namespace. Reuse is not limited to only Core and the most closely-related domains.

Namespaces are sometimes referred to at a high level (like "Core") when the release version is already known or doesn't matter in the given context. To be unambiguous, the version should also be included (e.g., "Core 4.0").

Categories

By convention, NIEM categorizes release namespaces. This makes schemas easier to find in the release packages. In the 4.0 release, the categories are the following:

Namespace	Count	Description
Core	1	Defines general properties and types (e.g., person, organization, activity) that do not belong to a specific subject area. Objects defined in Core leverage base XML objects defined in the Proxy and Structures schemas, and can be used as base objects for definitions in domain-specific schemas as well as extension schemas.
Domain	14	Defines properties and types specific to a given subject area.
Code	36	Provides NIEM-conformant representations of code sets that are typically defined and managed outside of NIEM.
Adapter	4	Defines adapter components (the NIEM mechanism to reuse non-conformant external standards in a conformant way).
External	5*	Defines a namespace from an external standard that is not NIEM conformant. External standards are integrated into NIEM through the schemas in this namespace.
Proxy	1	Defines complex types corresponding to standard XML Schema simple types.
Utility	4	Provides components needed to support NIEM methodologies.
Support	1	Defines the underlying standardized structure for data objects in NIEM. Each of the data objects in the other namespaces reuse the basic data objects in Support: `structures` contains NIEM base types to provide a set of generic types from which all types should derive. `appinfo` contains definitions used in high-level data modeling and also for validation of NIEM-conformant instances.

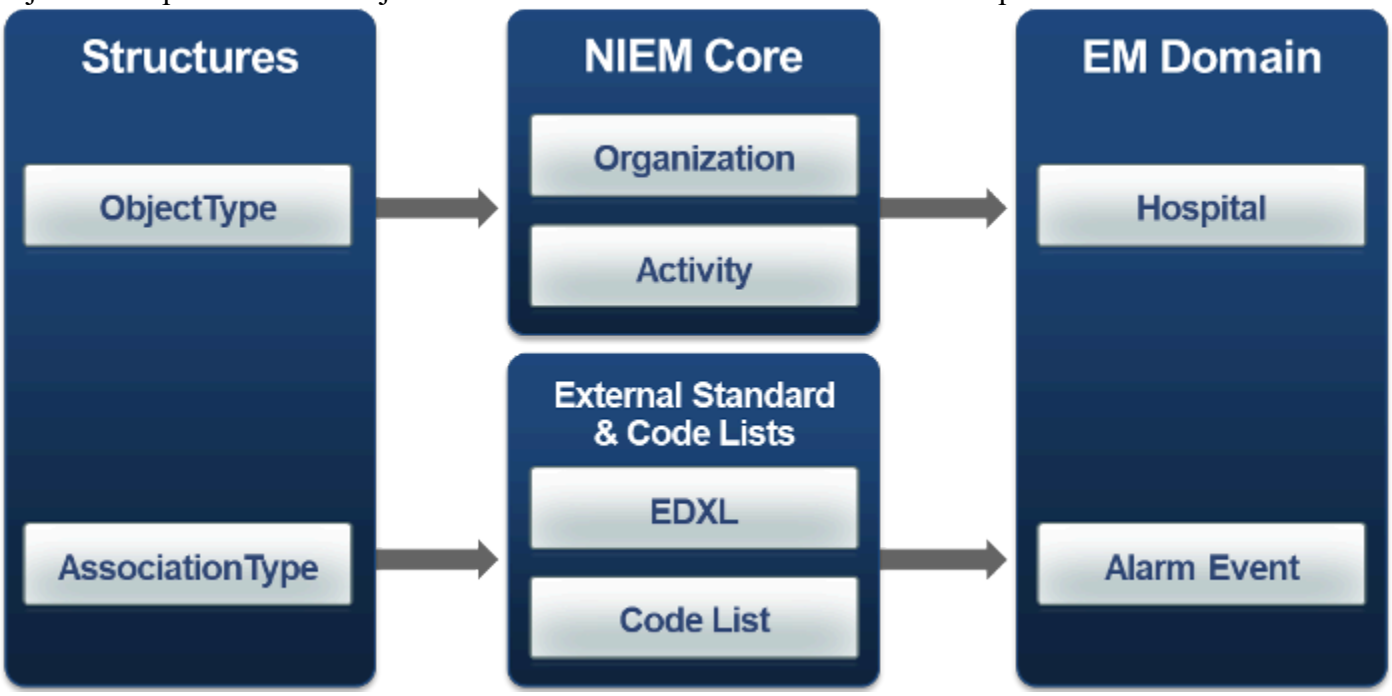
* - External standards are sometimes made up of multiple namespaces. The count above simply lists the number of external standards in NIEM, not the total number of namespaces they represent.

- See the 5.0 NIEM release on GitHub to review the full set of schemas.

- See the training content pages to get an overview of the content in these namespaces.

An IEPD is typically made up of subsetted versions of NIEM release namespaces, and one or more local namespaces that contain user-defined properties and types. These user-defined namespaces are often referred to as **extension** namespaces.

> The Domain namespace category provides a mission-based and domain-specific layer of data objects that specialize base objects from the NIEM Core and Structures namespaces.



Persistence

NIEM release namespaces are persistent. Once a release is published, its schemas will not be overwritten with future changes. Changes go into new schemas. This ensures that existing exchanges do not break when NIEM publishes a new release.

> Because release namespaces are persistent, exchanges do not have to be updated when a new release is published. Older release schemas can continue to be used indefinitely.

Modeling guidance

A namespace defines content in the form of properties and types. Details about these components are provided in their own sections.

Identifier

All NIEM-conformant namespaces are assigned an identifier, formatted as an absolute URI.

- In NIEM XML, this identifier appears in the schema's `targetNamespace` attribute.
- In NIEM JSON, this identifier appears in the JSON-LD context.

The Core 4.0 namespace identifier is https://release.niem.gov/niem/niem-core/4.0/.

Identifiers must by unique

A target namespace acts as the unique ID for a namespace. Each namespace must have its own target namespace. This makes all schemas easy to reference individually and prevents one namespace from being defined across multiple files.

Identifiers must be absolute URIs

A target namespace must be an absolute URI. Relative URIs are not allowed. An absolute URI, such as a URL or a URN, is universally identifiable.

Absolute URL example:

http://release.niem.gov/niem/niem-core/4.0/

Relative path URL:

niem/niem-core/4.0/

Relative root URL :

/niem/niem-core/4.0/

Relative protocol URL:

//niem/niem-core/3.2/

URI conventions

For NIEM release namespaces, the target namespace URI typically follows the format:

```
base / category / name / version /

http://release.niem.gov/niem/domains/immigration/4.0/
-------------base-----------/--cat--/---name----/ver/
```

The category is included only if there are multiple schemas for that kind of namespace. Since there is only one Core namespace, the URI for Core omits the category.

More examples:

Namespace	Target namespace URI
Core 3.2	http://release.niem.gov/niem/niem-core/3.2/
Core 4.0	http://release.niem.gov/niem/niem-core/4.0/
Immigration 4.0 domain	http://release.niem.gov/niem/domains/immigration/4.0/
USPS 4.0 codes	http://release.niem.gov/niem/codes/usps_states/4.0/

User-defined namespaces can follow different conventions for their URIs.

URI bases

NIEM release namespaces use identifiers that begin with the base `http://release.niem.gov/niem/` because the NIEM Management Office owns this web domain. For user-defined namespaces, like IEPD extension schemas, a different URI base should be chosen - preferably a domain that the user owns.

> Replace `http://release.niem.gov/niem/` with a different domain for namespaces that are not part of a NIEM release.

Prefix

Namespace prefixes act as abbreviations for the full namespace URI.

> The namespace prefix for Core 4.0 is `nc`.

Target namespace URIs uniquely identify a namespace, but they are lengthy and can be awkward to use when referring to components (option 1, below). Namespace prefixes allow use to use a much simpler syntax (option 2).

- option 1: `{https://release.niem.gov/niem/niem-core/4.0/}PersonType`
- option 2: `nc:PersonType`

The URIs are still necessary, however, for two reasons:

- Namespace prefixes are not unique.
- Namespace prefixes are locally assigned.

The NIEM 3.0 release uses `nc` as the namespace prefix for Core. The 4.0 release does the same. This is done on purpose since it requires less work to update IEPDs and documentation among other things. The side effect, however, is that `nc` does not refer to any one specific version of Core. Furthermore, an IEPD could choose to assign a completely different prefix to whichever version of Core it is using.

All of this leads to the following guidance:

> For formal usage, a namespace prefix must be linked to the URI that it represents.

This enables us to use the simpler syntax provided by the prefixes while maintaining the precision provided by the URIs.

Version

A namespace version field is used to distinguish different drafts or updates of a namespace. This version does not have to correspond with the release version or any versioning information in the URI.

> During the alpha 1 stage of the 4.0 development process, the Core namespace version was **alpha1**.

When the 4.0 NIEM schemas were under development, this version attribute was set to the corresponding pre-release stage - “alpha1”, “beta1”, “rc2”, etc.

In the final release, the Core 4.0 namespace has version **1**.

Version “1” represents that this is the first published version of the 4.0 Core namespace. NIEM has not published follow-up versions (e.g., “2”) to any namespace at the same URI, but it is an (unlikely) option for changes that do not affect schema validation or model semantics.

Definition

Each namespace must have a definition that describes what it is.

Language - [new in 5.0]

Each namespace must define the language in which its components are defined. The language for NIEM release schemas is `en-US`.

Conformance Targets

Each NIEM namespace must define one more conformance targets. This explicitly identifies which rule sets should be applied for conformance validation. A conformance target is identified by a URI.

For conformance to the NDR, the target will look like:

https://reference.niem.gov/niem/specification/naming-and-design-rules/VERSION/#TARGET

VERSION should be the version of the NDR:

- **3.0** - for schemas based on NDR 3.0 rules, like those in the NIEM 3.0, 3.1, and 3.2 releases.
- **4.0** - for schemas based on NDR 4.0 rules, like those in the NIEM 4.0 release.
- **5.0** - for schemas based on NDR 5.0 rules, like those in the NIEM 5.0 release.

TARGET should be a target defined by the NDR:

- **ReferenceSchemaDocument** - for schemas following the NDR’s stricter rule set, like the NIEM release schemas.
- **ExtensionSchemaDocument** - for schemas following the NDR’s less rigid rule set.

A NIEM 3.2 schema using NDR extension rules would use the conformance target: https://reference.niem.gov/niem/specification/naming-and-design-rules/**3.0/#ExtensionSchemaDocument**

A NIEM 4.0 schema using the NDR reference rules would use the conformance target: https://reference.niem.gov/niem/specification/naming-and-design-rules/**4.0/#ReferenceSchemaDocument**

Release schemas must follow the reference rules, but locally-defined namespaces (like extension schemas) may choose which rule set to target.

See the section about the Conformance Targets Attribute Specification for more information.

XML

Full Templates

Templates for a 4.0 and a 5.0 NIEM namespace are provided below.

This should help give an overview of how a NIEM namespace looks in XML Schema, but each part will be broken out and explained in the section that follows.

Placeholders appear in upper case. A few common namespace prefixes and import statements for 4.0 and 5.0 are also provided, but the relative paths in the import statements will need to be adjusted based on local directory layouts.

> In order to use extension schema rules rather than reference schema rules, like for an IEPD extension namespace, change the end of the ct:conformanceTargets value from `#ReferenceSchemaDocument` to `#ExtensionSchemaDocument`.

4.0 template

```
<?xml version="1.0" encoding="US-ASCII"?>
<xs:schema
  targetNamespace="URI" version="VERSION"

  xsi:schemaLocation="http://release.niem.gov/niem/appinfo/4.0/ ../niem/utilit
  http://release.niem.gov/niem/conformanceTargets/3.0/ ../niem/utility/conform

  ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-a

  xmlns:PREFIX="URI"
  xmlns:appinfo="http://release.niem.gov/niem/appinfo/4.0/"
  xmlns:ct="http://release.niem.gov/niem/conformanceTargets/3.0/"
  xmlns:nc="http://release.niem.gov/niem/niem-core/4.0/"
  xmlns:niem-xs="http://release.niem.gov/niem/proxy/xsd/4.0/"
  xmlns:structures="http://release.niem.gov/niem/structures/4.0/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <xs:annotation>
    <xs:documentation>NAMESPACE_DEFINITION</xs:documentation>
    <xs:appinfo>
      <appinfo:LocalTerm term="TERM1" literal="LITERAL1"/>
      <appinfo:LocalTerm term="TERM2" definition="DEFINITION2"/>
    </xs:appinfo>
  </xs:annotation>

  <xs:import schemaLocation="IMPORT_PATH" namespace="IMPORT_URI"/>
  <xs:import schemaLocation="../niem/niem-core/4.0/niem-core.xsd"
```

```
            namespace="http://release.niem.gov/niem/niem-core/4.0/"/>
  <xs:import schemaLocation="../niem/utility/structures/4.0/structures.xsd"
            namespace="http://release.niem.gov/niem/structures/4.0/"/>

  <!-- element, attribute, and type declarations -->

</xs:schema>
```

5.0 template

```
<?xml version="1.0" encoding="US-ASCII"?>
<xs:schema
  targetNamespace="URI" version="VERSION" xml:lang="en-US"

  xsi:schemaLocation="http://release.niem.gov/niem/appinfo/5.0/ ../utility/app
  http://release.niem.gov/niem/conformanceTargets/3.0/ ../utility/conformanceT

  ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-a

  xmlns:PREFIX="URI"
  xmlns:appinfo="http://release.niem.gov/niem/appinfo/5.0/"
  xmlns:ct="http://release.niem.gov/niem/conformanceTargets/3.0/"
  xmlns:nc="http://release.niem.gov/niem/niem-core/5.0/"
  xmlns:niem-xs="http://release.niem.gov/niem/proxy/xsd/5.0/"
  xmlns:structures="http://release.niem.gov/niem/structures/5.0/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <xs:annotation>
    <xs:documentation>NAMESPACE_DEFINITION</xs:documentation>
    <xs:appinfo>
      <appinfo:LocalTerm term="TERM1" literal="LITERAL1"/>
      <appinfo:LocalTerm term="TERM2" definition="DEFINITION2"/>
    </xs:appinfo>
  </xs:annotation>

  <xs:import namespace="IMPORT_URI" schemaLocation="IMPORT_PATH"/>
  <xs:import namespace="http://release.niem.gov/niem/niem-core/5.0/" schemaLoc
  <xs:import namespace="http://release.niem.gov/niem/structures/5.0/" schemaLo

  <!-- element, attribute, and type declarations -->

</xs:schema>
```

Local terminology, elements, attributes, and types, which appear in the template above, are described in their own sections.

Parts of the template

The following goes into more detail about parts of the templates given above.

URI, Version, and Language

The target namespace URI, version, and language are declared as schema attributes:

`xml:lang` is a newly-added requirement to NDR 5.0.

```
targetNamespace="URI"

version="VERSION"

xml:lang="LANG"
```

Core 5.0 example:

```
<xs:schema targetNamespace="http://release.niem.gov/niem/niem-core/5.0/" versi
  ...
</xs:schema>
```

xsi:SchemaLocation

`xsi:schemaLocation` is used to bind one or more URIs to their locations. NIEM uses this in places where a namespace is referenced but does not have to be imported (metadata on the schema itself).

The value of the attribute is a string of one or more "URI LOCATION" pairs.

```
xsi:schemaLocation="URI1 LOC1 URI2 LOC2 ..."
```

In 4.0, `xsi:schemaLocation` is used for the conformance targets namespace and the appinfo namespace.

```
<xs:schema
  xsi:schemaLocation="http://release.niem.gov/niem/appinfo/4.0/ ../niem/utilit
  http://release.niem.gov/niem/conformanceTargets/3.0/ ../niem/utility/conform
  ...
</xs:schema>
```

Conformance Targets

Conformance targets are URIs that identify rules to apply to check conformance.

```
ct:conformanceTargets="URI"
```

If there are multiple conformance targets, separate each URI with a space.

```
ct:conformanceTargets="URI1 URI2 ..."
```

The namespace prefix `ct` corresponds to URI `http://release.niem.gov/niem/conformanceTargets/3.0/`

The conformance targets namespace did not change in the NIEM 4.0 release, so it remains at version 3.0.

NIEM 4.0-based example of a namespace targeting the NDR 4.0 reference rules:

```
<xs:schema
  ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-a
  >
  ...
</xs:schema>
```

NIEM 4.0-based example of a namespace (like an extension schema) targeting the NDR 4.0 extension rules:

```
<xs:schema
  ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-a
  >
  ...
</xs:schema>
```

See the section about the Conformance Targets Attribute Specification for more information.

Namespace prefixes

Prefixes should be assigned for each namespace that is referenced, and also for the current namespace.

```
xmlns:PREFIX="URI"
```

This statement binds the prefix to the given URI, so the prefix may be used in the schema as an abbreviation for the full URI.

Core 4.0 namespace prefix example:

```
<xs:schema
  xmlns:nc="http://release.niem.gov/niem/niem-core/4.0/">
  ...
</xs:schema>
```

Prefixes should be assigned for:

- the current namespace
- imported namespaces
- xsi:schemaLocation namespaces
- the XML Schema namespace (xs)
- the XML Schema Instance namespace (xsi)

NIEM namespaces that are not required but that are commonly used include:

- Core

- niem-xs
- structures

The target and imported namespaces will vary based on the given schema, but the remaining ones from the lists above appear as the following in NIEM 4.0:

```
<xs:schema
  xmlns:appinfo="http://release.niem.gov/niem/appinfo/4.0/"
  xmlns:ct="http://release.niem.gov/niem/conformanceTargets/3.0/"
  xmlns:nc="http://release.niem.gov/niem/niem-core/4.0/"
  xmlns:niem-xs="http://release.niem.gov/niem/proxy/xsd/4.0/"
  xmlns:structures="http://release.niem.gov/niem/structures/4.0/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
</xs:schema>
```

Definition

A definition for the namespace is required.

```
<xs:annotation>
  <xs:documentation>DEFINITION</xs:documentation>
</xs:annotation>
```

The definition is represented as annotation documentation on the xs:schema root node.

CBRN 4.0 domain example:

```
<xs:schema>
  <xs:annotation>
    <xs:documentation>Chemical, Biological, Radiological, and Nuclear Domain</
  </xs:annotation>
  ...
</xs:schema>
```

Imports

Import statements (or catalog files) are required in order to reuse content from other namespaces.

An import statement lists the URI and either the absolute or the relative path of the namespace to be imported.

```
<xs:import schemaLocation="IMPORT_PATH" namespace="IMPORT_URI"/>
```

Example import of Core 4.0:

```
<xs:import schemaLocation="../niem/niem-core/4.0/niem-core.xsd"
           namespace="http://release.niem.gov/niem/structures/4.0/"/>
```

NIEM's profile of XML Schema

To promote consistency across a broad community, NIEM limits some of the features available in XML Schema that provide flexibility but may hurt interoperability or reusability.

A few of the key restrictions on the usage of XML Schema are listed below. See the NDR for more.

No local elements

Local elements and attributes cannot be reused outside of the type in which they are defined, which conflicts with the NIEM principle of maximizing reusability.

No anonymous types

Types defined anonymously can only be used by the elements that define them, which conflicts with the NIEM principle of maximizing reusability.

No xs:include

Include statements are declared to combine multiple schema files into a single logical namespace with one target namespace. Import statements are declared to reuse content from a schema with a target namespace.

NIEM requires that each XML Schema must have its own target namespace to provide unambiguous identification of content, so include statements may not be used.

No xs:choice (ref only)

The use of `xs:choice` can lead to ambiguity in some circumstances and are thus not allowed in NIEM reference schemas. Element substitution is a common alternative.

Extension schemas are allowed to use `xs:choice`.

JSON

Schema template

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "properties": {
    ...
  },
  "definitions": {
    ...
  }
}
```

`$schema` defines which draft of the JSON Schema specification that this schema should conform to.

`additionalProperties` specifies whether or not properties in addition to the ones defined in this schema are allowed in instances.

`properties` contains the NIEM property declarations. See more in the Properties section.

`definitions` contains the NIEM type declarations. See more in the Types section.

Schema example

The example below shows an example JSON schema, based on current guidance.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "properties": {
    "nc:Vehicle": {
      "description": "A conveyance designed to carry an operator, passengers a
      "oneOf": [
        {
          "$ref": "#/definitions/nc:VehicleType"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/nc:VehicleType"
          }
        }
      ]
    },
    "nc:VehicleAxleQuantity": {
      "description": "A count of common axles of rotation of one or more wheel
      "oneOf": [
        {
          "$ref": "#/definitions/niem-xs:nonNegativeInteger"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/niem-xs:nonNegativeInteger"
          }
        }
      ]
    },
    "nc:VehicleMSRPAmount": {
      "description": "A manufacturer's suggested retail price of a vehicle; a
      "oneOf": [
        {
```

```
      "$ref": "#/definitions/nc:AmountType"
    },
    {
      "type": "array",
      "items": {
        "$ref": "#/definitions/nc:AmountType"
      }
    }
  ]
},
"nc:Amount": {
  "description": "An amount of money.",
  "oneOf": [
    {
      "$ref": "#/definitions/niem-xs:decimal"
    },
    {
      "type": "array",
      "items": {
        "$ref": "#/definitions/niem-xs:decimal"
      }
    }
  ]
},
"nc:Currency": {
  "description": "A data concept for a unit of money or exchange.",
  "oneOf": [
    {
      "anyOf": [
        {
          "$ref": "#/properties/nc:CurrencyCode"
        }
      ]
    },
    {
      "type": "array",
      "items": {
        "anyOf": [
          {
            "$ref": "#/properties/nc:CurrencyCode"
          }
        ]
      }
    }
  ]
},
"nc:CurrencyCode": {
  "description": "A unit of money or exchange.",
  "oneOf": [
    {
```

```
          "$ref": "#/definitions/iso_4217:CurrencyCodeType"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/iso_4217:CurrencyCodeType"
          }
        }
      ]
    },
    "nc:VehicleIdentification": {
      "description": "A unique identification for a specific vehicle.",
      "oneOf": [
        {
          "$ref": "#/definitions/nc:IdentificationType"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/nc:IdentificationType"
          }
        }
      ]
    },
    "nc:IdentificationID": {
      "description": "An identifier.",
      "oneOf": [
        {
          "$ref": "#/definitions/niem-xs:string"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/niem-xs:string"
          }
        }
      ]
    }
  },
  "definitions": {
    "nc:VehicleType": {
      "description": "A data type for a conveyance designed to carry an operat
      "allOf": [
        {
          "$ref": "#/definitions/nc:ConveyanceType"
        },
        {
          "type": "object",
          "properties": {
            "nc:VehicleAxleQuantity": {
```

```
          "$ref": "#/properties/nc:VehicleAxleQuantity"
        },
        "nc:VehicleIdentification": {
          "$ref": "#/properties/nc:VehicleIdentification"
        },
        "nc:VehicleMSRPAmount": {
          "$ref": "#/properties/nc:VehicleMSRPAmount"
        }
      }
    }
  ]
},
"nc:ConveyanceType": {
  "description": "A data type for a means of transport from place to place
  "allOf": [
    {
      "$ref": "#/definitions/_base"
    },
    {
      "$ref": "#/definitions/nc:ItemType"
    },
    {
      "type": "object",
      "properties": {}
    }
  ]
},
"nc:ItemType": {
  "description": "A data type for an article or thing.",
  "allOf": [
    {
      "$ref": "#/definitions/_base"
    },
    {
      "type": "object",
      "properties": {}
    }
  ]
},
"nc:AmountType": {
  "description": "A data type for an amount of money.",
  "type": "object",
  "properties": {
    "nc:Amount": {
      "$ref": "#/properties/nc:Amount"
    },
    "nc:Currency": {
      "$ref": "#/properties/nc:Currency"
    }
  }
```

```
  },
  "iso_4217:CurrencyCodeType": {
    "description": "A data type for a currency that qualifies a monetary amo
    "oneOf": [
      {
        "$ref": "#/definitions/iso_4217:CurrencyCodeSimpleType"
      },
      {
        "type": "object",
        "properties": {
          "rdf:value": {
            "$ref": "#/definitions/iso_4217:CurrencyCodeSimpleType"
          }
        }
      }
    ]
  },
  "iso_4217:CurrencyCodeSimpleType": {
    "type": "string",
    "description": "A data type for a currency that qualifies a monetary amo
    "oneOf": [
      {
        "enum": [
          "EUR"
        ],
        "description": "Euro"
      },
      {
        "enum": [
          "GBP"
        ],
        "description": "Pound Sterling"
      },
      {
        "enum": [
          "USD"
        ],
        "description": "US Dollar"
      }
    ]
  },
  "nc:IdentificationType": {
    "description": "A data type for a representation of an identity.",
    "type": "object",
    "properties": {
      "nc:IdentificationID": {
        "$ref": "#/properties/nc:IdentificationID"
      }
    }
  },
```

```
    "niem-xs:decimal": {
      "description": "A data type for arbitrary precision decimal numbers.",
      "type": "number"
    },
    "niem-xs:nonNegativeInteger": {
      "description": "A data type for an integer with a minimum value of 0.",
      "type": "number"
    },
    "niem-xs:string": {
      "description": "A data type for character strings in XML.",
      "type": "string"
    },
    "_base": {
      "type": "object",
      "patternProperties": {
        "^ism:.*": {
          "type": "string"
        },
        "^ntk:.*": {
          "type": "string"
        }
      },
      "properties": {
        "@id": {
          "format": "uriref"
        },
        "@base": {
          "format": "uriref"
        }
      }
    }
  }
}
```

JSON-LD Context

Additional context information is needed to associate the namespace prefixes used by NIEM with their URIs.

The NIEM 4.0 release provides this context association in the file niem-4.0-context.jsonld, a subset of which is provided below:

```
{
    "aamva_d20": "http://release.niem.gov/niem/codes/aamva_d20/4.0/#",
    "ag": "http://release.niem.gov/niem/domains/agriculture/4.0/#",
    "nc": "http://release.niem.gov/niem/niem-core/4.0/#",
    "usps": "http://release.niem.gov/niem/codes/usps_states/4.0/#"
}
```

This is a simple JSON file assigning target namespaces, followed by the “#” symbol, to each namespace prefix used in the release.

The hash symbol is used as part of JSON pointer notation.

NDR references

The following provides information about namespaces in the Naming and Design Rules (NDR) Specification.

General-purpose

NDR 5.0

Rule	Applicability	Title
NDR 9-82	REF, EXT	Schema has data definition
NDR 9-83	REF, EXT	Schema document defines target namespace
NDR 9-84	REF, EXT	Target namespace is absolute URI
NDR 9-85	REF, EXT	Schema has version
NDR 10-68	REF, EXT	Component marked as deprecated is deprecated component
NDR 10-69	REF, EXT	Deprecated annotates schema component
NDR 10-70	REF, EXT	External import indicator annotates import
NDR 10-78	REF, EXT, INS, SET	Use structures consistent with specification
NDR 11-48	REF, EXT	Same namespace means same components
NDR 11-49	REF, EXT	Different version means different view
NDR 11-50	SET	Reference schema document imports reference schema document
NDR 11-51	SET	Extension schema document imports reference or extension schema document
NDR 11-52	REF, EXT	Structures imported as conformant
NDR 11-53	REF, EXT	XML namespace imported as conformant
NDR 11-54	SET	Each namespace may have only a single root schema in a schema set
NDR 11-55	REF, EXT	Consistently marked namespace imports
NDR 12-1	INS	Instance must be schema-valid
NDR 12-2	INS	Empty content has no meaning

NDR 4.0

Rule	Applicability	Title
NDR 9-82	REF, EXT	Schema has data definition
NDR 9-83	REF, EXT	Schema document defines target namespace
NDR 9-84	REF, EXT	Target namespace is absolute URI
NDR 9-85	REF, EXT	Schema has version
NDR 10-67	REF, EXT	Component marked as deprecated is deprecated component
NDR 10-68	REF, EXT	Deprecated annotates schema component
NDR 10-69	REF, EXT	External import indicator annotates import
NDR 10-77	REF, EXT, INS, SET	Use structures consistent with specification
NDR 11-46	REF, EXT	Same namespace means same components
NDR 11-47	REF, EXT	Different version means different view
NDR 11-48	SET	Reference schema document imports reference schema document
NDR 11-49	SET	Extension schema document imports reference or extension schema document
NDR 11-50	REF, EXT	Structures imported as conformant
NDR 11-51	REF, EXT	XML namespace imported as conformant
NDR 11-52	SET	Each namespace may have only a single root schema in a schema set
NDR 11-53	REF, EXT	Consistently marked namespace imports
NDR 12-1	INS	Instance must be schema-valid
NDR 12-2	INS	Empty content has no meaning

XML-specific

NDR 5.0

Rule	Applicability	Title
NDR 4-1	SET	Schema marked as reference schema document must conform
NDR 4-2	SET	Schema marked as extension schema document must conform
NDR 4-3	REF, EXT	Schema is CTAS-conformant
NDR 4-4	REF, EXT	Document element has attribute ct:conformanceTargets
NDR 4-5	REF	Schema claims reference schema conformance target
NDR 4-6	EXT	Schema claims extension conformance target
NDR 7-1	REF, EXT, INS	Document is an XML document
NDR 7-2	REF, EXT, INS	Document uses XML namespaces properly

Rule	Applicability	Title
NDR 7-3	REF, EXT	Document is a schema document
NDR 7-4	REF, EXT	Document element is xs:schema
NDR 9-75	REF, EXT	No use of xs:group
NDR 9-76	REF, EXT	No definition of attribute groups
NDR 9-77	REF, EXT	Comment is not recommended
NDR 9-78	REF, EXT	Documentation element has no element children
NDR 9-79	REF, EXT	xs:appinfo children are comments, elements, or whitespace
NDR 9-80	REF, EXT	Appinfo child elements have namespaces
NDR 9-81	REF, EXT	Appinfo descendants are not XML Schema elements
NDR 9-86	REF	No disallowed substitutions
NDR 9-87	REF	No disallowed derivations
NDR 9-88	REF, EXT	No use of xs:redefine
NDR 9-89	REF, EXT	No use of xs:include
NDR 9-90	REF, EXT	xs:import must have namespace
NDR 9-91	SET	XML Schema document set must be complete
NDR 9-92	REF, EXT	Namespace referenced by attribute type is imported
NDR 9-93	REF, EXT	Namespace referenced by attribute base is imported
NDR 9-94	REF, EXT	Namespace referenced by attribute itemType is imported
NDR 9-95	REF, EXT	Namespaces referenced by attribute memberTypes is imported
NDR 9-96	REF, EXT	Namespace referenced by attribute ref is imported
NDR 9-97	REF, EXT	Namespace referenced by attribute substitutionGroup is imported
NDR 10-45	REF, EXT	Schema component name has xml:lang
NDR 11-30	REF, EXT	xs:documentation has xml:lang

NDR 4.0

Rule	Applicability	Title
NDR 4-1	SET	Schema marked as reference schema document must conform
NDR 4-2	SET	Schema marked as extension schema document must conform
NDR 4-3	REF, EXT	Schema is CTAS-conformant
NDR 4-4	REF, EXT	Document element has attribute ct:conformanceTargets
NDR 4-5	REF	Schema claims reference schema conformance target
NDR 4-6	EXT	Schema claims extension conformance target
NDR 7-1	REF, EXT, INS	Document is an XML document
NDR 7-2	REF, EXT, INS	Document uses XML namespaces properly
NDR 7-3	REF, EXT	Document is a schema document
NDR 7-4	REF, EXT	Document element is xs:schema
NDR 9-75	REF, EXT	No use of xs:group
NDR 9-76	REF, EXT	No definition of attribute groups
NDR 9-77	REF, EXT	Comment is not recommended
NDR 9-78	REF, EXT	Documentation element has no element children
NDR 9-79	REF, EXT	xs:appinfo children are comments, elements, or whitespace
NDR 9-80	REF, EXT	Appinfo child elements have namespaces
NDR 9-81	REF, EXT	Appinfo descendants are not XML Schema elements

Rule	Applicability	Title
NDR 9-86	REF	No disallowed substitutions
NDR 9-87	REF	No disallowed derivations
NDR 9-88	REF, EXT	No use of xs:redefine
NDR 9-89	REF, EXT	No use of xs:include
NDR 9-90	REF, EXT	xs:import must have namespace
NDR 9-91	SET	XML Schema document set must be complete
NDR 9-92	REF, EXT	Namespace referenced by attribute type is imported
NDR 9-93	REF, EXT	Namespace referenced by attribute base is imported
NDR 9-94	REF, EXT	Namespace referenced by attribute itemType is imported
NDR 9-95	REF, EXT	Namespaces referenced by attribute memberTypes is imported
NDR 9-96	REF, EXT	Namespace referenced by attribute ref is imported
NDR 9-97	REF, EXT	Namespace referenced by attribute substitutionGroup is imported

Properties

A property represents a concept, idea, or thing. It defines specific semantics and appears in exchanges as the tag or label for a field.

Properties may be more commonly known as as elements, attributes, fields, tags, keys, or keywords.

> One of the properties defined in Core is `PersonBirthDate`. It represents a person's birthday and will carry that value in an exchange.
>
> - The type of the property will define the value's structure (e.g., free text or a specific date format).
> - The property itself is what gives this date value its meaning and distinguishes it from other kinds of dates that might appear, such as a driver license expiration date.

- Elements vs Attributes
- Modeling Guidance
  - Global
  - Conformant type
  - Naming
    - OED Terms
    - Spelling
    - Parts of a name
    - Restricted terms
    - No double terms
    - ID vs Identification
  - Definitions
    - Required
    - Human-readable
    - Informative
    - Unique
    - No duplication
    - No type info
    - Standard opening phrases
    - Begin with "A(n)"
    - Follow ISO 11179-4
- Representation terms and definition phrases
  - Simple value properties
  - Other properties
- NDR references
  - General property info
  - Name info
  - Definition info

On its own, a property has meaning but no inherent structure.

> Without a type, any kind of value could be provided for property `PersonBirthDate` above, like the following:

- 06/30/1950
- 1950-06-30
- June 30th, 1950
- green (without a type, any value is valid)

A property must have an assigned type in order to have both clear semantics and a well-defined structure.

Elements vs Attributes

In NIEM, there are two basic kinds of properties: elements and attributes.

Attributes can only ever be used to represent simple content (a value). They do not exist independently; they must be carried by an element.

Elements can be used to represent simple content (a value) or complex content (an object). In either case, an element can also carry attributes.

> This example lists 5 elements (each beginning with `nc:Person`) and 2 attributes (`structures:id` and `nc:personNameInitialIndicator`):

```
<nc:Person>
  <nc:PersonName structures:id="a123">
    <nc:PersonGivenName>John</nc:PersonGivenName>
    <nc:PersonMiddleName nc:personNameInitialIndicator="true">Q</nc:PersonMidd
    <nc:PersonSurName>Smith</nc:PersonSurName>
  </nc:PersonName>
</nc:Person>
```

Attributes cannot be extended, substituted, augmented, or have multiple occurrences. Additionally, because NIEM uses attributes for referencing and linking data, metadata, and security markup, elements should be used over attributes whenever possible in order to support these features. Over 99% of the properties in NIEM are elements.

> The information on this page describes a general property in NIEM. Information about specific kinds of properties, their XML representations, and their JSON representations or alternatives appear as sub-pages under this section:

- Elements

  An element represents a concept. In an instance, it acts as a container that may carry either a simple value or an object, and optionally attributes.

- Abstract Elements

  An abstract element is an element defined as a placeholder in a schema that must be replaced with an appropriate substitution in an instance.

- Substitutable Elements

A substitutable element is one that can replace another element in an instance.

- Attributes

  An attribute is a property that may carry a simple value only.

Modeling Guidance

Information about property names and definitions are provided separately in the pages that follow. Other rules and guidance for modeling properties are provided below.

Global

All properties must be declared globally as top-level elements in a schema. One of the key principles of NIEM is to maximize reuse. A local property cannot be reused outside of the one type in which it is defined.

Conformant type

A property type must be NIEM-conformant.

> Only an abstract element may be defined without a type.

Naming

A single NIEM component name may consist of multiple terms. A term is a meaningful word, an abbreviation for a word, or an acronym.

The following provides rules and guidance for naming properties.

OED Terms

Besides the additional accepted terms listed below, terms should be real words as defined by the Oxford English Dictionary.

Spelling

For consistency, words should use the preferred U.S. spelling as defined in the Oxford English Dictionary.

Additional terms

Additional terms that may be used are:

- `ID`, the authorized abbreviation for `Identifier`
- `URI`, the authorized abbreviation for `Uniform Resource Identifier`
- Local terminology, defined by a NIEM namespace.

Parts of a name

In accordance with IEC/ISO 11179 Part 5, terms that make up a NIEM property name are classified into four basic parts according to their placement and function:

- Object
- Qualifier
- Property
- Representation

Object term

An object term is the real world object or concept to which this name applies. This may be omitted if the property is very generic and no specific class applies.

  - `AircraftFuselageColorCode` - a color of the main body of an aircraft.
  - `Aircraft` is the object term. In this example, that is the object being modeled.

Qualifier term

A qualifier term indicates more specific information applicable to the object, property, or representation term.

- Multiple qualifier terms may be used if needed.
- Qualifiers are optional.

  - `AircraftFuselageColorCode` - a color of the main body of an aircraft.
  - `Fuselage` is a qualifier term. In this example, it describes what kind of color is expected.
  - A related element in NIEM is `AircraftWingColorCode`. `Wing` would be the qualifier term in that example.

Property term

A property term is the characteristic, or portion of the object, to which this name applies.

  - `AircraftFuselageColorCode` - a color of the main body of an aircraft.
  - `Color` is the property term. This is the primary characteristic of the property name. The value of this property will be a color.

Representation term

A representation term indicates the kind of value a property carries.

- A representation term, if present, must appear as the final term of the name.
- Representation terms are required for properties that carry a simple value, like a string or a number.
- Representation terms are required for other properties with specific types.

See the Representation Terms and Definitions Phrases reference table on this page for the list of representation terms.

- `AircraftFuselageColorCode` - a color of the main body of an aircraft.
- `Code` is the representation term. It indicates the general kind of value to expect.

Restricted terms

Type

Avoid use of the term `Type`. `Type` is reserved as the representation term for NIEM types; it indicates a structure. Properties should use the term `Category` instead to represent type / category / kind.

NIEM does allow for exceptions when the use of the term `Category` causes confusion over very well-known concepts that use the term `type`. One such exception is for the element `j:PersonBloodTypeCode`.

Number

Avoid use of the term `Number` in data names. Its use is usually too generic to be meaningful or helpful. A number should usually be an ID (Identifier), Quantity, Numeric (or Value), Amount, Measure, or Duration (of time).

No double terms

Do not use double terms (i.e., consecutive identical or similar terms such as IDID or NumberNumeric). Double terms should be replaced with a single instance of the term if it makes sense to do so.

This can sometimes cause different parts of a name, like the property term and the representation term, to be combined into one.

ID vs Identification

In NIEM, **`ID`** is used as the abbreviation for `Identifier`. This term is used for simple properties; it represents a unique string or number.

NIEM uses **`Identification`** as the representation term for a complex representation of an ID. This term carries a unique string or number, and additional information, like expiration date and the issuer. It is typically used by properties with the type `nc:IdentificationType`.

Definitions

The following provides rules and guidance for constructing property definitions.

Required

All properties must have a definition.

Human-readable

Definitions are meant to be human-readable.

Informative

Definitions should provide more information than the terms in the property's name, when possible.

> In some cases, a property name may be so obvious (e.g., `PersonHairColorText`) that attempts to provide synonyms or alternate phrasing would be counterproductive. A definition does not have to be elaborate if the property name would be obvious to the general public.

Unique

Each data component definition must be unique from all others and distinguishable in meaning. No two definitions can be identical in wording or so close in meaning that they could refer to the same data component.

> There is an exception for a property with multiple representations, like various substitutions of an abstract property. Since each substitution represents the same concept, they may each share the same definition.

No duplication

Definitions of a part do not need to redefine the whole.

> Definitions for elements like `nc:PersonHairColorText`, `nc:PersonName`, and `nc:PersonAgeMeasure` do not need to each define what a person is. That should be defined once by element `nc:Person`.

No type info

Type information should not appear in a property definition

A property definition is intended to describe semantic meaning only and should be decoupled from specific representation or structure. The representation term of a property and the standard opening phrase of a definition provide enough context to determine if a property is a string, a date, an ID, etc.

> More specific typing information usually belongs with the types (the structures) rather than the properties (the concepts). Things like patterns, code values, and the number of characters should usually not be embedded in property definitions.

There may be a few cases where providing such information is essential to the semantics of the property. In this case, it is permitted.

Standard opening phrases

Definitions of properties with simple content or designated types should use a standard opening phrase as defined by the NDR.

See the Representation terms and definition phrases reference table on this page for more.

Begin with "A(n)"

Property definitions almost always begin with an indefinite article (i.e., "a" or "an"), never a definite article (i.e., "the"). This is based on ISO 11179 guidance.

Follow ISO 11179-4

Definitions must follow 11179-4 requirements, and should follow its recommendations.

Representation terms and definition phrases

The tables below pair property representation terms (primary and alternates) with the recommended definition standard opening phrases.

Simple value properties

Rep Terms (Primary / Alt)	Typical Uses	Typical Data Types	Definition Phrase
Amount	Money	nc:AmountType	A(n) (optional adjectives) amount ...
BinaryObject	Binary, Base64 encoding, hexadecimal encoding	nc:BinaryType	A(n) ...
- Graphic	Diagram, graph, mathematical curve, etc.	nc:ImageType	A(n) ...
- Picture	Picture, photo, image	nc:ImageType	A(n) (optional adjectives) (image/ picture/photograph) ...
- Sound	Sound	nc:BinaryType	A(n) ...
- Video	Video	nc:BinaryType	A(n) ...
Code	Code Set	custom type	A(n) ...
DateTime	Date and time	nc:DateType	A(n) (optional adjectives) date and time ...
- Date	Date	nc:DateType	A(n) (optional adjectives) date ...
- Time	Time	niem-xs:time	A(n) (optional adjectives) time ...
- Duration	Length of time	niem-xs:duration	A(n) ...
ID	Unique identifier	niem-xs:string	A(n) (optional adjectives) identifier ...
- URI	URL	niem-xs:anyURI	A(n) ...
Indicator	True or False; Yes or No	niem-xs:boolean	True if ...; false (otherwise/if) ...
Measure	Measured value with unit	nc:MeasureType	A(n) ...

Rep Terms (Primary / Alt)	Typical Uses	Typical Data Types	Definition Phrase
Numeric	Number, no unit	nc:NumericType	A(n) ...
- Value	Calculated number	nc:NumericType	A(n) ...
- Rate	Ratio comparing two numbers	nc:NumericType	A(n) ...
- Percent	Ratio representing parts per hundred	nc:PercentType	A(n) ...
Quantity	Numeric count (non-monetary)	nc:QuantityType	A(n) (optional adjectives) (count/ number) ...
Text	Character string, free text	nc:TextType	A(n) ...
- Name	Word or term used for identification	nc:TextType	A(n) (optional adjectives) name...
- CategoryText	Kind or type of something	nc:TextType	A kind of ...
- DescriptionText	Free text description	nc:TextType	A description of ...
List	Sequence of values for very large data sets	various space-delimited lists	A(n) ...

Other properties

Representation Terms	Typical Uses	Typical Data Types	Definition Standard Opening Phrase
Abstract	Substitution group head for code lists or alternative to xs:choice	none	A data concept ...
Association	A relationship between elements	...AssociationType	An (optional adjectives) (relationship/ association) ...
Augmentation	Additional information about an existing type	...AugmentationType	Additional information about ...
AugmentationPoint	A hook in a complex type to allow others to add additional information (augmentations)	none	An augmentation point ...
Identification	An identifier plus related info (e.g., expiration date, category, status)	nc:IdentificationType	A(n) (optional adjectives) identification ...
Metadata	Secondary data about information (e.g., source, last updated, info quality)	...MetadataType	Metadata about ...
All other elements			A(n) ...

NDR references

General property info

NDR 5.0

- NDR Section 9.2 - Declaration components
- NDR Section 11.2 - Declaration components

Rule	Applicability	Title
NDR 7-5	REF, EXT	Component name follows ISO 11179 Part 5 Annex A

NDR 4.0

- NDR Section 9.2 - Declaration components
- NDR Section 11.2 - Declaration components

Rule	Applicability	Title
NDR 7-5	REF, EXT	Component name follows ISO 11179 Part 5 Annex A

Name info

NDR 5.0

- NDR Section 10.8 - Naming rules

Rule	Applicability	Title
NDR 10-44	REF, EXT	Schema component name composed of English words
NDR 10-46	REF, EXT	Schema component names have only specific characters
NDR 10-47	REF, EXT	Punctuation in component name is a separator
NDR 10-49	REF, EXT	Attribute name begins with lower case letter
NDR 10-50	REF, EXT	Name of schema component other than attribute and proxy type begins with upper case letter
NDR 10-53	REF, EXT	Local terminology interpretation
NDR 10-54	REF, EXT	Singular form is preferred in name
NDR 10-55	REF, EXT	Present tense is preferred in name
NDR 10-56	REF, EXT	Name does not have nonessential words
NDR 10-57	REF, EXT	Element or attribute name follows pattern
NDR 10-58	REF, EXT	Object-class term identifies concrete category

Rule	Applicability	Title
NDR 10-59	REF, EXT	Property term describes characteristic or subpart
NDR 10-60	REF, EXT	Name may have multiple qualifier terms
NDR 10-61	REF, EXT	Name has minimum necessary number of qualifier terms
NDR 10-62	REF, EXT	Order of qualifiers is not significant
NDR 10-63	REF, EXT	Redundant term in name is omitted

NDR 4.0

- NDR Section 10.8 - Naming rules

Rule	Applicability	Title
NDR 10-44	REF, EXT	Schema component name composed of English words
NDR 10-45	REF, EXT	Schema component names have only specific characters
NDR 10-46	REF, EXT	Punctuation in component name is a separator
NDR 10-47	REF, EXT	Names use camel case
NDR 10-49	REF, EXT	Name of schema component other than attribute and proxy type begins with upper case letter
NDR 10-50	REF, EXT	Names use common abbreviations
NDR 10-53	REF, EXT	Singular form is preferred in name
NDR 10-54	REF, EXT	Present tense is preferred in name
NDR 10-55	REF, EXT	Name does not have nonessential words
NDR 10-56	REF, EXT	Element or attribute name follows pattern
NDR 10-57	REF, EXT	Object-class term identifies concrete category
NDR 10-58	REF, EXT	Property term describes characteristic or subpart
NDR 10-59	REF, EXT	Name may have multiple qualifier terms
NDR 10-60	REF, EXT	Name has minimum necessary number of qualifier terms

Rule	Applicability	Title
NDR 10-61	REF, EXT	Order of qualifiers is not significant
NDR 10-62	REF, EXT	Redundant term in name is omitted

Definition info

NDR 5.0

- NDR Section 11.6 - Annotation components

Rule	Applicability	Title
NDR 11-24	REF, EXT	Data definition does not introduce ambiguity
NDR 11-25	REF, EXT	Object class has only one meaning
NDR 11-26	REF, EXT	Data definition of a part does not redefine the whole
NDR 11-27	REF, EXT	Do not leak representation into data definition
NDR 11-28	REF, EXT	Data definition follows 11179-4 requirements
NDR 11-29	REF, EXT	Data definition follows 11179-4 recommendations
NDR 11-36	REF, EXT	Standard opening phrase for date element or attribute data definition
NDR 11-37	REF, EXT	Standard opening phrase for quantity element or attribute data definition
NDR 11-38	REF, EXT	Standard opening phrase for picture element or attribute data definition
NDR 11-39	REF, EXT	Standard opening phrase for indicator element or attribute data definition
NDR 11-40	REF, EXT	Standard opening phrase for identification element or attribute data definition
NDR 11-41	REF, EXT	Standard opening phrase for name element or attribute data definition
NDR 11-42	REF, EXT	Standard opening phrase for element or attribute data definition

NDR 4.0

- NDR Section 11.6 - Annotation components

Rule	Applicability	Title
NDR 11-23	REF, EXT	Data definition does not introduce ambiguity
NDR 11-24	REF, EXT	Object class has only one meaning
NDR 11-25	REF, EXT	Data definition of a part does not redefine the whole
NDR 11-26	REF, EXT	Do not leak representation into data definition
NDR 11-27	REF, EXT	Data definition follows 11179-4 requirements
NDR 11-28	REF, EXT	Data definition follows 11179-4 recommendations
NDR 11-34	REF, EXT	Standard opening phrase for date element or attribute data definition
NDR 11-35	REF, EXT	Standard opening phrase for quantity element or attribute data definition
NDR 11-36	REF, EXT	Standard opening phrase for picture element or attribute data definition
NDR 11-37	REF, EXT	Standard opening phrase for indicator element or attribute data definition
NDR 11-38	REF, EXT	Standard opening phrase for identification element or attribute data definition

Rule	Applicability	Title
NDR 11-39	REF, EXT	Standard opening phrase for name element or attribute data definition
NDR 11-40	REF, EXT	Standard opening phrase for element or attribute data definition

Elements

An element represents a concept. In an instance, it acts as a container that may carry either simple content (a value) or complex content (an object). In either case, an element can also carry attributes.

- Modeling guidance
  - Type must be complex
  - Nillable
- XML
  - Instance example
  - Schema example
  - Schema template
  - Schema restrictions
    - Prohibited types
    - Prohibited attributes
- JSON
  - Instance example
  - Schema example
  - Schema template
- NDR references

- Core defines element `nc:PersonGivenName`. This element will carry simple content, for example, a value like “Jane”.
- Core also defines element `nc:Person`. This element will carry complex content - a set of sub-elements representing properties like name, age, height, and employment information.

Modeling guidance

In addition to the general modeling rules and guidance provided for properties, the following applies specifically to elements:

Type must be complex

An elements with a type must have a complex type.

All complex types in NIEM eventually derive from `structures:ObjectType` or a similar type in the NIEM `structures` namespace. These types contain attributes that enable NIEM elements to reference to other elements, support linked data, define metadata, and provide security markup. As these features are not possible with simple types, an element is required to have a complex type.

Note: An abstract element may be defined without a type.

Nillable

Elements in a reference schema must allow a nil value. This is needed to support NIEM’s referencing mechanism.

XML

Instance example

```
<nc:Person>
  <nc:PersonName structures:id="a123">
    <nc:PersonGivenName>John</nc:PersonGivenName>
    <nc:PersonMiddleName nc:personNameInitialIndicator="true">Q</nc:PersonMido
    <nc:PersonSurName>Smith</nc:PersonSurName>
  </nc:PersonName>
</nc:Person>
```

- `nc:Person` is an element that carries complex content.
- `nc:PersonName` is an element that carries complex content and an attribute.
- `nc:PersonGivenName` and `nc:PersonSurName` are elements that carry simple content (values).
- `nc:PersonMiddleName` is an element that carries simple content (a value) and an attribute.

Schema example

Despite the variations, each of the elements above is defined the same way in a schema. The different representations are defined by the element's type, not by the element itself.

```
<!-- Element with complex content -->
<xs:element name="Person" type="nc:PersonType" nillable="true">
  <xs:annotation>
    <xs:documentation>A human being.</xs:documentation>
  </xs:annotation>
</xs:element>

<!-- Element with simple content -->
<xs:element name="PersonMiddleName" type="nc:PersonNameTextType" nillable="tru
  <xs:annotation>
    <xs:documentation>A middle name of a person.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Schema template

```
<xs:element name="NAME" type="TYPE" nillable="true">
  <xs:annotation>
    <xs:documentation>DEFINITION</xs:documentation>
  </xs:annotation>
</xs:element>
```

Schema restrictions

The following is a list of restrictions that the NIEM NDR defines relating to XML element declarations.

Prohibited types

- xs:anySimpleType
- xs:anyType

These types are not allowed to prevent elements acting as wildcards. Substitutions and augmentations should be used instead.

Prohibited attributes

The following attributes may not be used in the declaration of an element in an XML Schema:

- block
- final
- default
- fixed

NIEM is designed to be extensible and customizable, in order to support a wide-ranging variety of requirements. The `block` and `final` attributes above prevent further customization by subsequent schemas.

The `default` and `fixed` attributes can result in unexpected behavior. A validating parser may construct values for an element assigned a default or fixed value in its schema; a non-validating parser will not. Preventing these attributes ensures that the information in an instance is not altered, depending on which parser is used to process the data.

JSON

Instance example

```
{
  "nc:Person": {
    "nc:PersonName" : {
      "nc:PersonGivenName": "John",
      "nc:PersonSurName": "Smith"
    }
  }
}
```

- `nc:Person` and `nc:PersonName` are elements that carry complex content.
- `nc:PersonGivenName` and `nc:PersonSurName` are elements that carry simple content (values).

Schema example

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
```

```
  "properties": {
    "nc:Person": {
      "description": "A human being.",
      "oneOf": [
        {
          "$ref": "#/definitions/nc:PersonType"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/nc:PersonType"
          }
        }
      ]
    }
  }
}
```

Note that this is different from the corresponding XML Schema syntax. An XML type defines if a property that it contains appears once or multiple times. If a person has two middle names, then the element `nc:PersonMiddleName` may appear in the instance multiple times.

In NIEM JSON, the property declares itself to have either the expected type, or an array of objects of the expected type. Properties are keys in a JSON object - they can only appear once.

This syntax supports the simple case, where a property is defined to have only one of the given value (`"nc:PersonMiddleName"`). It also supports the case where a property has multiple values, and must be accessed as an array, like `"nc:PersonMiddleName"[0]`, `"nc:PersonMiddleName"[1]`, etc.

Exchange schemas may subset this to choose the actual representation that is needed, but reference schemas should make both options available.

Schema template

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "properties": {
    "NAME": {
      "description": "DEFINITION",
      "oneOf": [
        {
          "$ref": "#/definitions/TYPE"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/TYPE"
```

```
                }
            }
        ]
    }
  }
}
```

NDR references

NDR 5.0

Rule	Applicability	Title
NDR 9-36	REF, EXT	Element declaration is top-level
NDR 9-37	REF, EXT	Element declaration has data definition
NDR 9-38	REF, EXT	Untyped element is abstract
NDR 9-39	REF, EXT	Element of type xs:anySimpleType is abstract
NDR 9-40	REF, EXT	Element type not in the XML Schema namespace
NDR 9-41	REF, EXT	Element type not in the XML namespace
NDR 9-42	REF, EXT	Element type is not simple type
NDR 9-43	REF	No element disallowed substitutions
NDR 9-44	REF	No element disallowed derivation
NDR 9-45	REF, EXT	No element default value
NDR 9-46	REF, EXT	No element fixed value
NDR 9-47	REF	Element declaration is nillable
NDR 9-72	REF, EXT	No use of xs:unique
NDR 9-73	REF, EXT	No use of xs:key
NDR 9-74	REF, EXT	No use of xs:keyref
NDR 10-19	REF, EXT	Element of code type has code representation term
NDR 10-44	REF, EXT	Schema component name composed of English words
NDR	REF, EXT	Schema component name has xml:lang

Rule	Applicability	Title
10-45		
NDR 10-46	REF, EXT	Schema component names have only specific characters
NDR 10-47	REF, EXT	Punctuation in component name is a separator
NDR 10-48	REF, EXT	Names use camel case
NDR 10-50	REF, EXT	Name of schema component other than attribute and proxy type begins with upper case letter
NDR 10-51	REF, EXT	Names use common abbreviations
NDR 10-54	REF, EXT	Singular form is preferred in name
NDR 10-55	REF, EXT	Present tense is preferred in name
NDR 10-56	REF, EXT	Name does not have nonessential words
NDR 10-57	REF, EXT	Element or attribute name follows pattern
NDR 10-58	REF, EXT	Object-class term identifies concrete category
NDR 10-59	REF, EXT	Property term describes characteristic or subpart
NDR 10-60	REF, EXT	Name may have multiple qualifier terms
NDR 10-61	REF, EXT	Name has minimum necessary number of qualifier terms
NDR 10-62	REF, EXT	Order of qualifiers is not significant
NDR 10-63	REF, EXT	Redundant term in name is omitted
NDR 10-64	REF, EXT	Element with simple content has representation term
NDR 10-65	REF, EXT	Element with complex content has representation term when appropriate
NDR 10-66	REF, EXT	Element with complex content has representation term only when appropriate
NDR 11-12	REF, EXT	Element type does not have a simple type name
NDR 11-13	REF, EXT	Element type is from conformant namespace
NDR 11-14	REF, EXT	Name of element that ends in Abstract is abstract
NDR 11-15	REF, EXT	Name of element declaration with simple content has representation term

Rule	Applicability	Title
NDR 11-16	SET	Name of element declaration with simple content has representation term
NDR 11-17	REF, EXT	Element substitution group defined by conformant schema
NDR 11-30	REF, EXT	xs:documentation has xml:lang
NDR 12-10	INS	Order of properties is expressed via structures:sequenceID

NDR 4.0

Rule	Applicability	Title
NDR 9-36	REF, EXT	Element declaration is top-level
NDR 9-37	REF, EXT	Element declaration has data definition
NDR 9-38	REF, EXT	Untyped element is abstract
NDR 9-39	REF, EXT	Element of type xs:anySimpleType is abstract
NDR 9-40	REF, EXT	Element type not in the XML Schema namespace
NDR 9-41	REF, EXT	Element type not in the XML namespace
NDR 9-42	REF, EXT	Element type is not simple type
NDR 9-43	REF	No element disallowed substitutions
NDR 9-44	REF	No element disallowed derivation
NDR 9-45	REF, EXT	No element default value
NDR 9-46	REF, EXT	No element fixed value
NDR 9-47	REF	Element declaration is nillable
NDR 9-72	REF, EXT	No use of xs:unique
NDR 9-73	REF, EXT	No use of xs:key
NDR 9-74	REF, EXT	No use of xs:keyref
NDR 10-19	REF, EXT	Element of code type has code representation term
NDR 10-44	REF, EXT	Schema component name composed of English words

Rule	Applicability	Title
NDR 10-45	REF, EXT	Schema component names have only specific characters
NDR 10-46	REF, EXT	Punctuation in component name is a separator
NDR 10-47	REF, EXT	Names use camel case
NDR 10-49	REF, EXT	Name of schema component other than attribute and proxy type begins with upper case letter
NDR 10-50	REF, EXT	Names use common abbreviations
NDR 10-53	REF, EXT	Singular form is preferred in name
NDR 10-54	REF, EXT	Present tense is preferred in name
NDR 10-55	REF, EXT	Name does not have nonessential words
NDR 10-56	REF, EXT	Element or attribute name follows pattern
NDR 10-57	REF, EXT	Object-class term identifies concrete category
NDR 10-58	REF, EXT	Property term describes characteristic or subpart
NDR 10-59	REF, EXT	Name may have multiple qualifier terms
NDR 10-60	REF, EXT	Name has minimum necessary number of qualifier terms
NDR 10-61	REF, EXT	Order of qualifiers is not significant
NDR 10-62	REF, EXT	Redundant term in name is omitted
NDR 10-63	REF, EXT	Element with simple content has representation term
NDR 10-64	REF, EXT	Element with complex content has representation term when appropriate
NDR 10-65	REF, EXT	Element with complex content has representation term only when appropriate
NDR 11-11	REF, EXT	Element type does not have a simple type name
NDR 11-12	REF, EXT	Element type is from conformant namespace
NDR 11-13	REF, EXT	Name of element that ends in Abstract is abstract
NDR 11-14	REF, EXT	Name of element declaration with simple content has representation term
NDR	SET	Name of element declaration with simple content has representation term

Rule	Applicability	Title
11-15		
NDR 11-16	REF, EXT	Element substitution group defined by conformant schema

Abstract Elements

An abstract element exists solely to be replaced. Schemas may define abstract elements and valid substitutions, but only the valid substitutions may appear in an instance.

- Modeling guidance
  - Name
  - Definition
  - No type
- XML
  - Instances
  - Schema example
  - Schema template
- JSON
- NDR references

While it is best to reuse common components as much as possible, there are some things that do not have standard representations. This is especially true with code sets. There is no global and authoritative code set for occupation codes, or offense codes, or county codes. NIEM supports the need to be able to use alternate code sets or representations with abstract elements.

> NIEM defines abstract element `nc:EmployeeOccupationAbstract` and an element that may be substituted in its place, `nc:EmployeeOccupationCode`. This element has a code set established by the US Department of Labor. A domain or exchange may reuse the provided concrete element with the code set, or may add a code set of its own as an alternate substitution.

An element that is not abstract may sometimes be referred to as "concrete" in order to distinguish the two.

The counterpart to an abstract element is a substitutable element, which is described after this section.

Modeling guidance

Name

The name of an abstract element must end with **"Abstract"**.

See the Representation Pattern section for cases when the representation term "Representation" should be used instead.

Definition

An abstract element should begin with the standard opening phrase: **"A data concept for "**.

No type

In NIEM, an abstract element should most likely not have a type.

In XML Schema, abstract elements may or may not be assigned types. An abstract element that has no type can be replaced by any element. An abstract element with a type can only be replaced by an element with the same type, or with a type that is derived from that type. This prevents conflicts - an element that is specifically supposed to be a person element should not be replaced by a location element.

If you want to create an abstract element that may be replaced by divergent types (like free text and code sets types, or like PersonType and OrganizationType), the abstract element must not have a type.

XML

Instances

Abstract elements exist solely to be replaced. They do not appear in an XML instance.

See Substitutable Elements (XML) for an example of a substitution in an instance.

Schema example

```
<xs:element name="LocationCountyAbstract" abstract="true">
  <xs:annotation>
    <xs:documentation>
      A data concept for a county, parish, vicinage, or other such geopolitica
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

- The name ends with “Abstract”.
- The element does not have a type.
- The element declaration has an `abstract="true"` attribute.
- The definition begins with “A data concept for …”

Schema template

```
<xs:element name="NAMEAbstract" abstract="true">
  <xs:annotation>
    <xs:documentation>
      A data concept for ...
    </xs:documentation>
  </xs:annotation>
</xs:element>
```

JSON

The concept of an abstract element does not exist in JSON like it does in XML; however, the idea behind it can still be represented in a similar manner. In any XML instance, an abstract element should not appear. It can either be omitted or replaced by one or more assigned substitutions. This should occur in JSON as well.

See the Substitutable Elements (JSON) section for more.

NDR references

NDR 5.0

Rule	Applicability	Title
NDR 9-38	REF, EXT	Untyped element is abstract
NDR 9-39	REF, EXT	Element of type xs:anySimpleType is abstract
NDR 11-14	REF, EXT	Name of element that ends in Abstract is abstract
NDR 11-35	REF, EXT	Standard opening phrase for abstract element data definition

NDR 4.0

Rule	Applicability	Title
NDR 9-38	REF, EXT	Untyped element is abstract
NDR 9-39	REF, EXT	Element of type xs:anySimpleType is abstract
NDR 11-13	REF, EXT	Name of element that ends in Abstract is abstract
NDR 11-33	REF, EXT	Standard opening phrase for abstract element data definition

Substitutable Elements

A substitutable element is one that can replace another element in an instance.

- Modeling guidance
  - Avoid type conflicts
- XML
  - Instance example
  - Schema example
  - Schema template
- JSON
  - Instance example
  - Schema

A key point about substitutions is that they are defined by the replacement element, and not by the element that is being replaced. This means, for example, that domains and exchanges may create substitutions for elements in Core…without any modification to the Core schema itself.

> The Justice domain defines element `j:PersonHairColorCode` with a code set from the NCIC standard as a substitution for abstract element `PersonHairColorAbstract` in Core. The Core namespace is not affected by this substitution.

Modeling guidance

Avoid type conflicts

The type of a substitutable element cannot conflict with the type of the element it is meant to replace. If the element to be replaced has no type, then there is no conflict.

XML

Instance example

In the example below, element `nc:Date` appears as a replacement for element `nc:DateRepresentation`, an abstract element that cannot appear in an instance.

```
<nc:Person>
  <nc:PersonBirthDate>
    <!-- Date is substituted for DateRepresentation -->
    <nc:Date>1950-06-30</nc:Date>
  </nc:PersonBirthDate>
</nc:Person>
```

Schema example

```
<xs:element name="Date" type="niem-xs:date" substitutionGroup="nc:DateRepresen
```

```
  <xs:annotation>
    <xs:documentation>A full date.</xs:documentation>
  </xs:annotation>
</xs:element>
```

- The element declares what it may be substituted in place of in the `substitutionGroup` attribute.
- In this case, the `Date` element may replace element `nc:DateRepresentation` wherever it occurs.

Schema template

```
<xs:element name="NAME" type="TYPE" substitutionGroup="SUBSTITUTION" nillable=
  <xs:annotation>
    <xs:documentation>DEFINITION</xs:documentation>
  </xs:annotation>
</xs:element>
```

JSON

Instance example

A substitutable element can be used in place of the element defined as the head of its substitution group. This should appear in JSON similar to the way it appear in XML.

The example below shows the substitution of element `nc:Date` for element `nc:DateRepresentation`, which does not appear in the instance.

JSON example:

```
{
  "nc:Person" : {
    "nc:PersonBirthDate" : {
      "nc:Date"  : "1950-06-30"
    }
  }
}
```

Schema

Substitutions are not declared in JSON Schema.

A JSON Schema with the `additionalProperties` keyword set to true will allow for the addition of properties not defined in the schema. This is how substitutions can be permitted.

```
{
   "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": true,
```

```
  "properties": {
    ...
  },
  "definitions": {
    ...
  }
}
```

Attributes

In NIEM, there are two basic kinds of properties: elements and attributes.

Attributes can only ever be used to represent simple content (a value). They do not exist independently; they must be carried by an element.

Elements can be used to represent simple content (a value) or complex content (an object). In either case, an element can also carry attributes.

- When to use
- Modeling guidance
  - Name casing
  - Simple type
  - Prohibited types
  - No defaults values
  - No optional fixed values
- XML
  - Instance example
  - Schema example
  - Schema template
- JSON
  - Complex content
  - Simple content
- NDR references

Example:

```
<nc:Person>
  <nc:PersonName structures:id="a123">
    <nc:PersonGivenName>John</nc:PersonGivenName>
    <nc:PersonMiddleName nc:personNameInitialIndicator="true">Q</nc:PersonMido
    <nc:PersonSurName>Smith</nc:PersonSurName>
  </nc:PersonName>
</nc:Person>
```

This example lists 5 elements (each beginning with `nc:Person`) and 2 attributes (`structures:id` and `nc:personNameInitialIndicator`).

Attributes cannot be extended, substituted, augmented, or have multiple occurrences. Additionally, because NIEM uses attributes for referencing and linking data, metadata, and security markup, elements should be used over attributes whenever possible in order to support these features. Over 99% of the properties in NIEM are elements.

When to use

Attributes may be defined sparingly, when it makes sense for something to be tightly-coupled with an

element and will not need the other features that elements enable. This decision may come down to a judgment call.

In the example above, it makes sense for the property representing the name initial indicator to be an attribute. There never needs to be more than one value per name, it should not need to be locally extended, it should not need to carry metadata or references.

Modeling guidance

Attributes share the same modeling rules and guidelines as regular properties, with the following exceptions:

Name casing

NIEM attribute names must appear in lower camel case (lowerCamelCase). The first letter must be in lower case.

Simple type

Attributes must have a simple type (value only).

Prohibited types

An attribute may not be assigned any of the following types:

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

This prevents attributes from being used for referencing other data (NIEM already supports this with `structures:id`, `structures:ref`, and `structures:uri`), representing data that is not in an XML format, and wildcards.

No defaults values

An attribute may not have a default value assigned.

A validating parser may take an instance that does not have such an attribute present and construct the attribute with the default value in its place. A non-validating parser will not. This rule was established to make sure an instance represents the exact same data, no matter which tool or parser is used.

No optional fixed values

An optional attribute may not have a fixed value.

Similar to the rule about default values, a validating parser may take an instance in which the optional attribute does not appear and construct an attribute with the fixed value in its place. A non-validating parser will not. Again, an instance must represent the exact same data, not matter which tool or parser is used.

This rule does not apply to required fixed attributes. That is because a required attribute must appear in the message, and it must have the fixed value. In this case, there is nothing for a validating parser to construct.

XML

Instance example

This example shows the use of two attributes: `structures:id` and `nc:personNameInitialIndicator`.

```
<nc:Person>
  <nc:PersonName structures:id="a123">
    <nc:PersonGivenName>John</nc:PersonGivenName>
    <nc:PersonMiddleName nc:personNameInitialIndicator="true">Q</nc:PersonMido
    <nc:PersonSurName>Smith</nc:PersonSurName>
  </nc:PersonName>
</nc:Person>
```

Schema example

```
<xs:attribute name="personNameInitialIndicator" type="xs:boolean">
  <xs:annotation>
    <xs:documentation>
      True if value represents the first letter or initials of a persons name;
    </xs:documentation>
  </xs:annotation>
</xs:attribute>
```

Schema template

```
<xs:attribute name="NAME" type="TYPE">
  <xs:annotation>
    <xs:documentation>DEFINITION</xs:documentation>
  </xs:annotation>
</xs:attribute>
```

JSON

JSON does not have a similar notion to attributes as in XML. Because of this, attributes must be represented in JSON in a different manner.

Looking at the example of a person with a name from before, we can see how elements and attributes appear in XML:

```
<nc:Person>
  <nc:PersonName structures:id="a123">
    <nc:PersonGivenName>John</nc:PersonGivenName>
    <nc:PersonMiddleName nc:personNameInitialIndicator="true">Q</nc:PersonMidd
    <nc:PersonSurName>Smith</nc:PersonSurName>
  </nc:PersonName>
</nc:Person>
```

In JSON, this would appear as:

```
{
  "nc:Person": {
    "nc:PersonName" : {
      "structures:id": "a123",
      "nc:PersonGivenName": "John",
      "nc:PersonMiddleName": {
        "rdf:value": "Q",
        "nc:personNameInitialIndicator": true
      },
      "nc:PersonSurName": "Smith"
    }
  }
}
```

Here, we see that JSON properties just contain objects or values - no attributes. NIEM attributes are represented as regular properties.

Complex content

For NIEM elements that contain an attribute and complex content (child elements), the translation is straight-forward: the attribute should appear in JSON in the same manner as the other elements.

XML example:

```
<nc:Person>
  <nc:PersonName structures:id="a123">
    <nc:PersonGivenName>John</nc:PersonGivenName>
    <nc:PersonSurName>Smith</nc:PersonSurName>
  </nc:PersonName>
</nc:Person>
```

XML element `nc:PersonName` contains one attribute and two elements.

JSON example:

```
{
```

```
  "nc:Person": {
    "nc:PersonName" : {
      "structures:id": "a123",
      "nc:PersonGivenName": "John",
      "nc:PersonSurName": "Smith"
    }
  }
}
```

JSON property `nc:PersonName` contains three properties.

Simple content

A bigger change occurs for XML elements that contain attributes and simple content (a value).

XML example:

```
<nc:PersonMiddleName nc:personNameInitialIndicator="true">Q</nc:PersonMiddleNa
```

XML element `nc:PersonMiddleName` contains one attribute and one value.

JSON example:

```
{
  "nc:PersonMiddleName": {
    "rdf:value": "Q",
    "nc:personNameInitialIndicator": true
  }
}
```

- JSON property `nc:PersonMiddleName` contains two properties.
- `rdf:value` is a new property that only appears in JSON.

Because JSON does not have attributes, the NIEM attribute `nc:personNameInitialIndicator` must be treated like a regular property. This requires the addition of a new property to hold the element's original value, so a new name must be provided. NIEM uses the name `rdf:value` for this kind of case.

NDR references

NDR 5.0

- NDR Section 9.2.3 - Attribute declaration
- NDR Section 11.2.3 - Attribute declaration

Rule	Applicability	Title
NDR 9-48	REF, EXT	Attribute declaration is top-level
NDR 9-49	REF, EXT	Attribute declaration has data definition
NDR 9-50	REF, EXT	Attribute declaration has type
NDR 9-51	REF, EXT	No attribute type of xs:ID
NDR 9-52	REF, EXT	No attribute type of xs:IDREF
NDR 9-53	REF, EXT	No attribute type of xs:IDREFS
NDR 9-54	REF, EXT	No attribute type of xs:ENTITY
NDR 9-55	REF, EXT	No attribute type of xs:ENTITIES
NDR 9-56	REF, EXT	No attribute type of xs:anySimpleType
NDR 9-57	REF, EXT	No attribute default values
NDR 9-58	REF, EXT	No fixed values for optional attributes
NDR 10-49	REF, EXT	Attribute name begins with lower case letter
NDR 11-10	REF, EXT	Attribute of code simple type has code representation term
NDR 11-18	REF, EXT	Attribute type defined by conformant schema
NDR 11-19	REF, EXT	Attribute name uses representation term

NDR 4.0

- NDR Section 9.2.3 - Attribute declaration
- NDR Section 11.2.3 - Attribute declaration

Rule	Applicability	Title
NDR 9-48	REF, EXT	Attribute declaration is top-level
NDR 9-49	REF, EXT	Attribute declaration has data definition
NDR 9-50	REF, EXT	Attribute declaration has type
NDR 9-51	REF, EXT	No attribute type of xs:ID
NDR 9-52	REF, EXT	No attribute type of xs:IDREF
NDR 9-53	REF, EXT	No attribute type of xs:IDREFS
NDR 9-54	REF, EXT	No attribute type of xs:ENTITY
NDR 9-55	REF, EXT	No attribute type of xs:ENTITIES
NDR 9-56	REF, EXT	No attribute type of xs:anySimpleType
NDR 9-57	REF, EXT	No attribute default values
NDR 9-58	REF, EXT	No fixed values for optional attributes
NDR 10-48	REF, EXT	Attribute name begins with lower case letter
NDR 11-9	REF, EXT	Attribute of code simple type has code representation term
NDR 11-17	REF, EXT	Attribute type defined by conformant schema
NDR 11-18	REF, EXT	Attribute name uses representation term

Types

A type defines a structure - an allowable set of values. A type might describe a simple value (e.g., a string, a number) or a complex object (e.g., `PersonType`).

- A type for a person's last name property might be a string.
- A type for an address property might represent an object, with its own set of properties for street, city, state, and zip.

- Kinds of types
- Modeling guidance
  - Types must be global
  - No anonymous types
  - Representation term "Type"
  - Definition required
  - Standard opening phrase
- NDR references

Types tend to be less specific than properties. That is by design in order to increase reusability.

> The properties `nc:PersonBirthDate`, `nc:ActivityDate`, and `it:ItineraryDepartureDate` all have different semantic meanings but can each reuse the same `nc:DateType` type.

Kinds of types

There are three fundamental representations for a NIEM type. This list corresponds directly with XML type styles. NIEM defines corresponding representations of these types for JSON, which is structured differently.

- Complex Types with Complex Content

  A complex type with complex content (CCC) is a structure that represents an object and may contain elements and attributes.

- Complex Types with Simple Content

  A complex type with simple content (CSC) is a structure that represents a simple value and that may optionally contain attributes.

- Simple Types

  A simple type is a structure that represents a simple value only.

Modeling guidance

The following are rules and guidelines that apply to all types.

Please see additional guidance from the Property section for naming and definitions.

Types must be global

All types must be defined globally, as a top-level component of a schema. This provides for maximum reuse.

No anonymous types

All types must be assigned names. Similar to the rule about types being global, this ensures maximum reuse.

Representation term "Type"

The name of a type must end with the term "Type".

Definition required

Definitions are required for all types.

Standard opening phrase

A type definition should begin with "A data type for ".

Since it is often the case that a type and an element of that type can be defined with identical or similar words (for example, Person and PersonType), it is a NIEM best practice to begin a type definition with the phrase "A data type for …" This ensures that the definition for the element and its associated type are easily distinguishable.

NDR references

NDR 5.0

- NDR Section 9.1 - Type definition components
- NDR Section 11.1 - Type definition components

Rule	Applicability	Title
NDR 7-5	REF, EXT	Component name follows ISO 11179 Part 5 Annex A
NDR 9-1	REF, EXT	No base type in the XML namespace
NDR 9-2	REF, EXT	No base type of xs:ID
NDR 9-3	REF, EXT	No base type of xs:IDREF
NDR 9-4	REF, EXT	No base type of xs:IDREFS
NDR 9-5	REF, EXT	No base type of xs:anyType
NDR 9-6	REF, EXT	No base type of xs:anySimpleType
NDR 9-7	REF, EXT	No base type of xs:NOTATION

Rule	Applicability	Title
NDR 9-8	REF, EXT	No base type of xs:ENTITY
NDR 9-9	REF, EXT	No base type of xs:ENTITIES
NDR 10-44	REF, EXT	Schema component name composed of English words
NDR 10-45	REF, EXT	Schema component name has xml:lang
NDR 10-46	REF, EXT	Schema component names have only specific characters
NDR 10-47	REF, EXT	Punctuation in component name is a separator
NDR 10-48	REF, EXT	Names use camel case
NDR 10-50	REF, EXT	Name of schema component other than attribute and proxy type begins with upper case letter
NDR 10-51	REF, EXT	Names use common abbreviations
NDR 10-54	REF, EXT	Singular form is preferred in name
NDR 10-55	REF, EXT	Present tense is preferred in name
NDR 10-56	REF, EXT	Name does not have nonessential words
NDR 10-60	REF, EXT	Name may have multiple qualifier terms
NDR 10-61	REF, EXT	Name has minimum necessary number of qualifier terms
NDR 10-62	REF, EXT	Order of qualifiers is not significant
NDR 10-63	REF, EXT	Redundant term in name is omitted
NDR 11-1	REF, EXT	Name of type ends in Type
NDR 11-2	REF, EXT	Only types have name ending in Type or SimpleType
NDR 11-3	REF, EXT	Base type definition defined by conformant schema
NDR 11-24	REF, EXT	Data definition does not introduce ambiguity
NDR 11-25	REF, EXT	Object class has only one meaning
NDR 11-26	REF, EXT	Data definition of a part does not redefine the whole
NDR 11-27	REF, EXT	Do not leak representation into data definition
NDR	REF, EXT	Data definition follows 11179-4 requirements

Rule	Applicability	Title
11-28		
NDR 11-29	REF, EXT	Data definition follows 11179-4 recommendations
NDR 11-30	REF, EXT	xs:documentation has xml:lang

NDR 4.0

- NDR Section 9.1 - Type definition components
- NDR Section 11.1 - Type definition components

Rule	Applicability	Title
NDR 7-5	REF, EXT	Component name follows ISO 11179 Part 5 Annex A
NDR 9-1	REF, EXT	No base type in the XML namespace
NDR 9-2	REF, EXT	No base type of xs:ID
NDR 9-3	REF, EXT	No base type of xs:IDREF
NDR 9-4	REF, EXT	No base type of xs:IDREFS
NDR 9-5	REF, EXT	No base type of xs:anyType
NDR 9-6	REF, EXT	No base type of xs:anySimpleType
NDR 9-7	REF, EXT	No base type of xs:NOTATION
NDR 9-8	REF, EXT	No base type of xs:ENTITY
NDR 9-9	REF, EXT	No base type of xs:ENTITIES
NDR 10-44	REF, EXT	Schema component name composed of English words
NDR 10-45	REF, EXT	Schema component names have only specific characters
NDR 10-46	REF, EXT	Punctuation in component name is a separator
NDR 10-47	REF, EXT	Names use camel case
NDR 10-49	REF, EXT	Name of schema component other than attribute and proxy type begins with upper case letter
NDR 10-50	REF, EXT	Names use common abbreviations
NDR 10-53	REF, EXT	Singular form is preferred in name
NDR 10-54	REF, EXT	Present tense is preferred in name
NDR 10-55	REF, EXT	Name does not have nonessential words
NDR 10-59	REF, EXT	Name may have multiple qualifier terms
NDR 10-60	REF, EXT	Name has minimum necessary number of qualifier terms

Rule	Applicability	Title
NDR 10-61	REF, EXT	Order of qualifiers is not significant
NDR 10-62	REF, EXT	Redundant term in name is omitted
NDR 11-1	REF, EXT	Name of type ends in Type
NDR 11-2	REF, EXT	Base type definition defined by conformant schema
NDR 11-23	REF, EXT	Data definition does not introduce ambiguity
NDR 11-24	REF, EXT	Object class has only one meaning
NDR 11-25	REF, EXT	Data definition of a part does not redefine the whole
NDR 11-26	REF, EXT	Do not leak representation into data definition
NDR 11-27	REF, EXT	Data definition follows 11179-4 requirements
NDR 11-28	REF, EXT	Data definition follows 11179-4 recommendations

Complex Types with Complex Content

A complex type with complex content (CCC) is a structure that represents an object and may contain elements and attributes.

A CCC type represents an object with properties, rather than a simple value.

- Modeling guidance
  - Representation term “Type”
  - Parent
  - Default parent
  - Sequence
  - Default cardinality
- XML
  - Instance example
  - Schema example
  - Schema template
- JSON
  - Instance example
  - Schema example
  - Schema template
- NDR references

Examples of CCC types in NIEM include the following:

- `nc:PersonType`, which includes properties like name, birthday, and height.
- `nc:LocationType`, which includes properties like an address and lat-long coordinates.
- `nc:ContactInformationType`, which includes properties like a telephone number and an email address.

Modeling guidance

Representation term “Type”

The name of a type must end with the representation term `Type`.

Parent

CCC types may only extend other CCC types.

Default parent

For a type that is not metadata, an association, or an augmentation, the default parent type, if none other is provided, should be `structures:ObjectType`.

Sequence

In reference schemas, a CCC type's elements must be represented as a sequence, rather than a choice.

Default cardinality

The default usage of properties in a type from a NIEM release should be optional and over-inclusive. In XML, this would be represented as `minOccurs="0"` and `maxOccurs="0"`.

Individual information exchanges can and often should have very precise cardinality requirements. Because these vary across different use cases, and because cardinality restrictions can be tightened but not loosened in a subset, cardinality should be set as flexibly as possible in the release schemas.

XML

Instance example

```
<nc:PersonName>
  <nc:PersonGivenName>Jane/nc:PersonGivenName>
  <nc:PersonMiddleName>Elizabeth</nc:PersonMiddleName>
  <nc:PersonSurName>Smith</nc:PersonSurName>
</nc:PersonName>
```

Schema example

The following shows an example of a subset of type `nc:PersonNameType`. It is a complex type with complex content (CCC) that contains elements and an attribute.

```
<xs:complexType name="PersonNameType">
  <xs:annotation>
    <xs:documentation>
      A data type for a combination of names and/or titles by which a person i
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      <xs:sequence>
        <xs:element ref="nc:PersonGivenName" minOccurs="0" maxOccurs="unbounde
        <xs:element ref="nc:PersonMiddleName" minOccurs="0" maxOccurs="unbound
        <xs:element ref="nc:PersonSurName" minOccurs="0" maxOccurs="unbounded"
      </xs:sequence>
      <xs:attribute ref="nc:personNameCommentText" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Schema template

This template shows the creation of a complex type with complex content. It specifies a parent type, two elements, and an attribute.

```
<xs:complexType name="NAME">
  <xs:annotation>
    <xs:documentation>A data type for ....</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="PARENT_TYPE">
      <xs:sequence>
        <xs:element ref="PROPERTY_1" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="PROPERTY_2" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="ATTRIBUTE" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

- The cardinality on elements may be adjusted by changing the `minOccurs` and `maxOccurs` values. NIEM reference schemas tend to be very flexible, but cardinality should be adjusted in IEPDs to meet specific exchange requirements.
- The cardinality on attributes may be adjusted by changing the `use` value from `optional` to `required`.

JSON

Instance example

The example below shows element `nc:Vehicle`, which has a CCC type. It contains an object with three properties.

Property `nc:VehicleIdentification` also has a CCC type, representing an identification object with one property of its own.

```
{
  "nc:Vehicle": {
    "nc:VehicleAxleQuantity": 2,
    "nc:VehicleIdentification": {
      "nc:IdentificationID": "AX54C62"
    },
    "nc:VehicleMSRPAmount": 25000
  }
}
```

Schema example

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "definitions": {
    "nc:VehicleType": {
      "description": "A data type for a conveyance designed to carry an operat
      "allOf": [
        {
          "$ref": "#/definitions/nc:ConveyanceType"
        },
        {
          "type": "object",
          "properties": {
            "nc:VehicleAxleQuantity": {
              "$ref": "#/properties/nc:VehicleAxleQuantity"
            },
            "nc:VehicleIdentification": {
              "$ref": "#/properties/nc:VehicleIdentification"
            },
            "nc:VehicleMSRPAmount": {
              "$ref": "#/properties/nc:VehicleMSRPAmount"
            }
          }
        }
      ]
    }
}
```

In NIEM, `nc:VehicleType` extends `nc:ConveyanceType`. This is captured by the `allOf` syntax above, which creates a reference to the parent type, then defines the vehicle-specific properties below.

Schema template

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "definitions": {
    "NAME": {
      "description": "A data type for ...",
      "allOf": [
        {
          "$ref": "#/definitions/PARENT_TYPE"
        },
        {
          "type": "object",
          "properties": {
```

```
            "PROPERTY_1": {
              "$ref": "#/properties/PROPERTY_1"
            },
            "PROPERTY_2": {
              "$ref": "#/properties/PROPERTY_2"
            }
          }
        }
      ]
    }
}
```

Any attributes that a CCC type contains should be represented the same way as elements, i.e., as properties under the JSON Schema `properties` key.

NDR references

NDR 5.0

- NDR Section 9.1 - Type definition components
- NDR Section 11.1 - Type definition components

Rule	Applicability	Title
NDR 9-25	REF, EXT	Complex type definition is top-level
NDR 9-26	REF, EXT	Complex type has data definition
NDR 9-27	REF, EXT	No mixed content on complex type
NDR 9-28	REF, EXT	No mixed content on complex content
NDR 9-29	REF, EXT	Complex type content is explicitly simple or complex
NDR 9-30	REF	Complex content uses extension
NDR 9-31	REF, EXT	Base type of complex type with complex content must have complex content
NDR 9-32	SET	Base type of complex type with complex content must have complex content
NDR 9-34	REF	No complex type disallowed substitutions
NDR 9-35	REF	No complex type disallowed derivation
NDR 9-60	EXT	Model group does not affect meaning
NDR 9-61	REF, EXT	No xs:all
NDR 9-62	REF	xs:sequence must be child of xs:extension
NDR 9-63	EXT	xs:sequence must be child of xs:extension or xs:restriction
NDR 9-64	REF	No xs:choice
NDR 9-65	EXT	xs:choice must be child of xs:sequence
NDR 9-66	REF, EXT	Sequence has minimum cardinality 1
NDR 9-67	REF, EXT	Sequence has maximum cardinality 1
NDR 9-68	EXT	Choice has minimum cardinality 1
NDR 9-69	EXT	Choice has maximum cardinality 1
NDR 9-70	REF	No use of xs:any

Rule	Applicability	Title
NDR 9-71	REF	No use of xs:anyAttribute
NDR 10-1	REF, EXT	Complex type has a category
NDR 10-2	REF, EXT	Object type with complex content is derived from structures:ObjectType
NDR 10-23	REF	Augmentable type has augmentation point element
NDR 10-24	REF, EXT	Augmentable type has at most one augmentation point element
NDR 11-20	REF, EXT	Element or attribute declaration introduced only once into a type
NDR 11-21	REF, EXT	Element reference defined by conformant schema
NDR 11-22	REF, EXT	Referenced attribute defined by conformant schemas
NDR 11-23	REF, EXT	Schema uses only known attribute groups
NDR 11-46	REF, EXT	Standard opening phrase for complex type data definition

NDR 4.0

- NDR Section 9.1 - Type definition components
- NDR Section 11.1 - Type definition components

Rule	Applicability	Title
NDR 9-25	REF, EXT	Complex type definition is top-level
NDR 9-26	REF, EXT	Complex type has data definition
NDR 9-27	REF, EXT	No mixed content on complex type
NDR 9-28	REF, EXT	No mixed content on complex content
NDR 9-29	REF, EXT	Complex type content is explicitly simple or complex
NDR 9-30	REF	Complex content uses extension
NDR 9-31	REF, EXT	Base type of complex type with complex content must have complex content
NDR 9-32	SET	Base type of complex type with complex content must have complex content
NDR 9-34	REF	No complex type disallowed substitutions
NDR 9-35	REF	No complex type disallowed derivation
NDR 9-60	EXT	Model group does not affect meaning
NDR 9-61	REF, EXT	No xs:all
NDR 9-62	REF	xs:sequence must be child of xs:extension
NDR 9-63	EXT	xs:sequence must be child of xs:extension or xs:restriction
NDR 9-64	REF	No xs:choice
NDR 9-65	EXT	xs:choice must be child of xs:sequence
NDR 9-66	REF, EXT	Sequence has minimum cardinality 1
NDR 9-67	REF, EXT	Sequence has maximum cardinality 1
NDR 9-68	EXT	Choice has minimum cardinality 1
NDR 9-69	EXT	Choice has maximum cardinality 1
NDR 9-70	REF	No use of xs:any
NDR 9-71	REF	No use of xs:anyAttribute
NDR 10-1	REF, EXT	Complex type has a category
NDR 10-2	REF, EXT	Object type with complex content is derived from structures:ObjectType
NDR 10-23	REF	Augmentable type has augmentation point element
NDR 10-24	REF, EXT	Augmentable type has at most one augmentation point element

Rule	Applicability	Title
NDR 11-19	REF, EXT	Element or attribute declaration introduced only once into a type
NDR 11-20	REF, EXT	Element reference defined by conformant schema
NDR 11-21	REF, EXT	Referenced attribute defined by conformant schemas
NDR 11-22	REF, EXT	Schema uses only known attribute groups
NDR 11-44	REF, EXT	Standard opening phrase for complex type data definition

Complex Types with Simple Content

A complex type with simple content (CSC) is a common type style in XML, but does not exist in JSON. “Complex type” means that the type may carry attributes. “Simple content” means that the type carries a value, rather than an object.

- CSC types in NIEM
  - niem-xs namespace
  - Code sets
- Modeling guidance
  - Extension
  - Structures attribute group
  - Code types
- XML
  - Instance example
  - Schema example
  - Schema template
- JSON
  - Instance example
  - Schema example
- NDR references

Example

```
<nc:PersonMiddleName nc:personNameInitialIndicator="true">Q</nc:PersonMiddleNa
```

Element `nc:PersonMiddleName` above carries the value “Q” because the type has simple content and an attribute because the type is complex.

CSC types in NIEM

Even though NIEM defines very few data attributes, attributes are used to support key NIEM features like referencing, linked data, metadata, and security markup. This is the reason that NIEM requires element types be complex - so these features may be available for use in exchanges.

While there are few special requirements in order to use attributes along with complex content, some additional steps must be taken in NIEM to support the use of attributes with simple content.

niem-xs namespace

The `niem-xs` namespace is a proxy namespace, which provides alternate complex type representations for built-in XML Schema simple types.

XML Schema defines fundamental types like `xs:string`, `xs:integer`, `xs:date`, and `xs:boolean`, but these are simple types. NIEM creates corresponding complex types with simple content.

Type `niem-xs:boolean` is a complex type with simple content. It is based on the XML Schema simple type `xs:boolean`, which represents a true or false value. It also adds attributes from the `structures` namespace to support the NIEM infrastructure.

Code sets

Similar to the corresponding `niem-xs` proxy types (CSCs) and XML Schema simple types, code sets are created in NIEM as two corresponding types:

- A simple type is created to define the code set via enumeration facets. Since the type is simple, it can only contain a value; it cannot contain attributes.
- A second type, a CSC type, is created. Because a CSC type is complex, it can contain attributes, and because it has simple content, it can carry value. The CSC type extends the original simple type with the facets and adds attributes from the `structures` namespace to support the NIEM infrastructure.

  - Simple type `nc:AddressCategoryCodeSimpleType` defines enumerations like `business` and `residential`.
  - CSC type `nc:AddressCategoryCodeType` extends the simple type with the facets, and adds attributes from `structures`.

Because these types correspond so closely to each other, the names and definitions tend to be very similar. The key distinction is the addition of "Simple" to the simple type's name.

To assign a code set to an element, the element must have the CSC version of the type. This allows the element to use both the code set and any attributes that may be needed.

```
<!-- The element with the simple value.  No attributes are used here. -->
<nc:AddressCategoryCode>residential</nc:AddressCategoryCode>

<!-- The same element, this time with an attribute. -->
<nc:AddressCategoryCode structures:metadata="m1">residential</nc:AddressCatego
```

Attributes may not need to be defined for all elements, but NIEM cannot predetermine where they may or may not need to appear. This will be determined by the requirements of the exchanges. NIEM requires elements to use complex types with attributes from `structures` so that exchanges may decide locally when the attributes should be used.

Modeling guidance

Extension

A CSC type must extend either a simple type or another CSC type.

Structures attribute group

In NIEM XML, a CSC type that extends a simple type must also include a reference to

`structures:SimpleObjectAttributeGroup`. This is an attribute group that contains all of the attributes needed to support the NIEM infrastructure.

Code types

A CSC type that is based on a simple type with enumerations (codes) must have a name that ends with `CodeType`.

XML

Instance example

```
<nc:AddressCategoryCode structures:id="a1">residential</nc:AddressCategoryCode
```

Schema example

```
<xs:complexType name="AddressCategoryCodeType">
  <xs:annotation>
    <xs:documentation>A data type for a kind of address.</xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="nc:AddressCategoryCodeSimpleType">
      <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

- The CSC type extends a simple type.
- The CSC type references an attribute group from `structures`.

Schema template

```
<xs:complexType name="NAMEType">
  <xs:annotation>
    <xs:documentation>A data type for ...</xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="SIMPLETYPE">
      <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

JSON

JSON does not have a syntax that directly corresponds to attributes or CSC types. In this case, any attributes must be treated like regular properties, and a name must be assigned to a newly-created

property that carries the type's original simple content. NIEM uses `rdf:value` as this name.

Instance example

```
{
  "nc:AddressCategoryCode": {
    "@id": "a1",
    "rdf:value": "residential"
  }
}
```

`@id` is a reserved key in JSON-LD and should be used in JSON as the counterparts to `structures:id` and `structures:ref` in XML instances.

NIEM also allows for a simpler representation - one in which no attributes will be used. In this case, the new `rdf:value` property may be omitted and the value may be carried directly on the original property:

```
{
  "nc:AddressCategoryCode": "residential"
}
```

The specific format to be used should be determined at the exchange level.

Schema example

```
{
  "nc:AddressCategoryCodeType": {
    "description": "A data type for a kind of address.",
    "oneOf": [
      {
        "$ref": "#/definitions/nc:AddressCategoryCodeSimpleType"
      },
      {
        "type": "object",
        "properties": {
          "rdf:value": {
            "$ref": "#/definitions/nc:AddressCategoryCodeSimpleType"
          }
        }
      }
    ]
  }
}
```

Guidance on how to represent additional attributes from `structures` in JSON Schema is still under development.

NDR references

NDR 5.0

Rule	Applicability	Title
NDR 9-25	REF, EXT	Complex type definition is top-level
NDR 9-26	REF, EXT	Complex type has data definition
NDR 9-27	REF, EXT	No mixed content on complex type
NDR 9-29	REF, EXT	Complex type content is explicitly simple or complex
NDR 9-30	REF	Complex content uses extension
NDR 9-33	REF	Simple content uses extension
NDR 9-35	REF	No complex type disallowed derivation
NDR 9-71	REF	No use of xs:anyAttribute
NDR 10-20	REF, EXT	Proxy type has designated structure
NDR 11-11	REF, EXT	Complex type with simple content has structures:SimpleObjectAttributeGroup
NDR 11-20	REF, EXT	Element or attribute declaration introduced only once into a type
NDR 11-22	REF, EXT	Referenced attribute defined by conformant schemas
NDR 11-23	REF, EXT	Schema uses only known attribute groups
NDR 11-46	REF, EXT	Standard opening phrase for complex type data definition

NDR 4.0

Rule	Applicability	Title
NDR 9-25	REF, EXT	Complex type definition is top-level
NDR 9-26	REF, EXT	Complex type has data definition
NDR 9-27	REF, EXT	No mixed content on complex type
NDR 9-29	REF, EXT	Complex type content is explicitly simple or complex
NDR 9-30	REF	Complex content uses extension
NDR 9-33	REF	Simple content uses extension
NDR 9-35	REF	No complex type disallowed derivation
NDR 9-71	REF	No use of xs:anyAttribute
NDR 10-20	REF, EXT	Proxy type has designated structure
NDR 11-10	REF, EXT	Complex type with simple content has structures:SimpleObjectAttributeGroup
NDR 11-19	REF, EXT	Element or attribute declaration introduced only once into a type
NDR 11-21	REF, EXT	Referenced attribute defined by conformant schemas
NDR 11-22	REF, EXT	Schema uses only known attribute groups
NDR 11-44	REF, EXT	Standard opening phrase for complex type data definition

Simple Types

A simple type is a structure that represents a simple value only.

Simple types are data structures like strings, numbers, and booleans. They can also be customized with facets so that only a certain range of numbers, or a certain set of codes, for example, are allowed.

- Kinds of simple types
- Modeling guidance
  - Representation term "SimpleType"
- XML
  - Instance example
  - Schema example
  - Schema template
- JSON
  - Instance example
  - Schema example
  - Schema templates
- NDR references

Simple types can be used in two ways in the model:

- As the type of attributes, which by definition cannot have a complex type.
- As the base of corresponding complex types with simple content (CSCs), which can be assigned to elements.

  - Core defines attribute `personNameInitialIndicator`. Its type is a boolean - a simple type.
  - Core defines simple type `AddressCategoryCodeSimpleType`, which constrains a string down to a few possible codes, like `residential` and `business`.

Provided types

Common simple types provided by XML include `xs:string`, `xs:token`, `xs:date`, `xs:date-time`, `xs:integer`, `xs:decimal`, and `xs:boolean`.

Simple types provided by JSON are `string`, `integer`, `decimal`, and `boolean`. Dates are simply strings in JSON, but if they follow the format set by ISO 8601, they can be converted into dates in JavaScript.

Kinds of simple types

In addition to the simple types built into XML and JSON and user-defined simple types with facets (typically codes), NIEM supports two additional kinds of simple types:

- List Types

A list type is a simple type that allows for values to be repeated, separated by a space.

- Union Types

  A union type is a simple type that represents the combined set of allowable values from other simple types.

Modeling guidance

Representation term “SimpleType”

A simple type name must end with the representation term “SimpleType”.

XML

Instance example

The following is an example of attribute `nc:personNameInitialIndicator`. It has simple type `xs:boolean`.

```
<nc:PersonMiddleName nc:personNameInitialIndicator="true">Q</nc:PersonMiddleNa
```

The attribute has a simple value: the boolean “true”.

Schema example

This example shows a subset of `nc:AddressCategoryCodeSimpleType`.

```
<xs:simpleType name="AddressCategoryCodeSimpleType">
  <xs:annotation>
    <xs:documentation>A data type for a kind of address.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:token">
    <xs:enumeration value="business">
      <xs:annotation>
        <xs:documentation>business</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="residential">
      <xs:annotation>
        <xs:documentation>residential</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
  </xs:restriction>
</xs:simpleType>
```

Schema template

This template shows the creation of a simple type with an enumeration. Other kinds of facets could be used instead.

```
<xs:simpleType name="NAMESimpleType">
  <xs:annotation>
    <xs:documentation>A data type for ...</xs:documentation>
  </xs:annotation>
  <xs:restriction base="SIMPLETYPE">
    <xs:enumeration value="CODE">
      <xs:annotation>
        <xs:documentation>CODE_DEFINITION</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
  </xs:restriction>
</xs:simpleType>
```

JSON

Instance example

The following shows an example of a property with a value based on a simple type:

```
{
  "nc:AddressCategoryCode": "residential"
}
```

Schema example

```
{
  "nc:AddressCategoryCodeSimpleType": {
    "type": "string",
    "description": "A data type for a kind of address.",
    "oneOf": [
      {
        "enum": [ "business" ],
        "description": "business"
      },
      {
        "enum": [ "residential" ],
        "description": "residential"
      }
    ]
  }
}
```

See Facets (JSON) for examples of simple types with other kinds of facets.

Schema templates

The following is a template for a simple type with an enumeration:

```
{
  "NAME": {
    "type": "BASE_TYPE",
    "description": "A data type for ...",
    "oneOf": [
      {
        "enum": [ "CODE" ],
        "description": "DEFINITION"
      }
    ]
  }
}
```

The following is another template for a simple type with an enumeration:

```
{
  "definitions": {
    "SIMPLE_TYPE": {
      "description": "TYPE_DEFINITION",
      "type": "BASE_TYPE",
      "FACET_KIND": "FACET_VALUE"
    },
  }
}
```

NDR references

NDR 5.0

- NDR Section 11.1.2 - Simple type definition

Rule	Applicability	Title
NDR 9-10	REF, EXT	Simple type definition is top-level
NDR 9-11	REF	No simple type disallowed derivation
NDR 9-12	REF, EXT	Simple type has data definition
NDR 11-4	REF, EXT	Name of simple type ends in SimpleType
NDR 11-8	REF, EXT	Name of a code simple type ends in CodeSimpleType
NDR 11-9	REF, EXT	Code simple type corresponds to a code list
NDR 11-47	REF, EXT	Standard opening phrase for simple type data definition

NDR 4.0

- NDR Section 11.1.2 - Simple type definition

Rule	Applicability	Title
NDR 9-10	REF, EXT	Simple type definition is top-level
NDR 9-11	REF	No simple type disallowed derivation
NDR 9-12	REF, EXT	Simple type has data definition
NDR 11-3	REF, EXT	Name of simple type ends in SimpleType
NDR 11-7	REF, EXT	Name of a code simple type ends in CodeSimpleType
NDR 11-8	REF, EXT	Code simple type corresponds to a code list
NDR 11-45	REF, EXT	Standard opening phrase for simple type data definition

List Types

A list type is a simple type that allows for values to be repeated, separated by a space.

- Modeling
  - Representation term “ListSimpleType”
- XML
  - Instance example
  - Schema example
  - Schema template
- JSON
- NDR References

In NIEM, multiple values are typically represented by a property that occurs multiple times. Lists are provided to represent larger sets of data values, such as readings from a sensor.

Modeling

Representation term “ListSimpleType”

The name of a list type should end with the representation term “ListSimpleType”.

XML

Instance example

```
<nc:MeasureDecimalValueList>1.1 1.1 1.1 1.2 1.1 1.1 1.3 1.3 1.4 1.3 1.3 1.3 1.
```

Schema example

The following example shows the declarations of

- a simple list type,
- a complex type with simple content based on the simple list type, and
- an element with the complex type.

```
<xs:simpleType name="DecimalListSimpleType">
  <xs:annotation>
    <xs:documentation>A data type for a white space-delimited list of decimal.
  </xs:annotation>
  <xs:list itemType="xs:decimal"/>
</xs:simpleType>

<xs:complexType name="DecimalListType">
  <xs:annotation>
    <xs:documentation>A data type for a white space-delimited list of decimal.
```

```
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="nc:DecimalListSimpleType">
      <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:element name="MeasureDecimalValueList" type="nc:DecimalListType" substitut
  <xs:annotation>
    <xs:documentation>A list of decimal measurement values, all using the same
  </xs:annotation>
</xs:element>
```

Schema template

The following is a template for the declaration of a simple list type:

```
<xs:simpleType name="NAMEListSimpleType">
  <xs:annotation>
    <xs:documentation>A data type for a list of ...</xs:documentation>
  </xs:annotation>
  <xs:list itemType="BASE-TYPE"/>
</xs:simpleType>
```

JSON

JSON guidance for list types is not yet available.

NDR References

NDR 5.0

Rule	Applicability	Title
NDR 9-15	REF, EXT	No list item type of xs:ID
NDR 9-16	REF, EXT	No list item type of xs:IDREF
NDR 9-17	REF, EXT	No list item type of xs:anySimpleType
NDR 9-18	REF, EXT	No list item type of xs:ENTITY
NDR 11-5	REF, EXT	Use lists only when data is uniform
NDR 11-6	REF, EXT	List item type defined by conformant schemas

NDR 4.0

Rule	Applicability	Title
NDR 9-15	REF, EXT	No list item type of xs:ID
NDR 9-16	REF, EXT	No list item type of xs:IDREF
NDR 9-17	REF, EXT	No list item type of xs:anySimpleType
NDR 9-18	REF, EXT	No list item type of xs:ENTITY

Rule	Applicability	Title
NDR 11-4	REF, EXT	Use lists only when data is uniform
NDR 11-5	REF, EXT	List item type defined by conformant schemas

Union Types

A union type is a simple type that represents the combined set of allowable values from other simple types.

- XML
  - Schema example
  - Instance example
  - Schema template
- JSON
- NDR References

A union type defines a list of simple types as its members. A value allowed by any one of its member types will be a valid value for the union type.

The benefit that this provides is reuse. Rather than needing to duplicate values, types can be reused in combination instead.

One kind of use case for a union is for situations when an existing code set provides some, but not all, of the values that are needed. One option is to create a code set with just the missing values, create a union type that combines the original codes with the new codes, and create a new element based on the union type.

XML

Schema example

This example shows a union type and its two member types.

```
<xs:simpleType name="GradeSimpleType">
  <xs:annotation>
    <xs:documentation>A data type for a grade.</xs:documentation>
  </xs:annotation>
  <xs:union memberTypes="ext:LetterGradeSimpleType ext:NumericGradeSimpleType'
</xs:simpleType>

<xs:simpleType name="LetterGradeSimpleType">
  <xs:annotation>
    <xs:documentation>A data type for a letter grade.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:token">
    <xs:pattern value="[A-F]"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="NumericGradeSimpleType">
  <xs:annotation>
```

```
    <xs:documentation>A data type for a numeric grade.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>
```

Instance example

This example shows possible values for a “GradeValue” element based on the union type above:

```
<ext:GradeValue>A</ext:GradeValue>
<ext:GradeValue>C</ext:GradeValue>
<ext:GradeValue>85</ext:GradeValue>
<ext:GradeValue>42</ext:GradeValue>
<ext:GradeValue>B</ext:GradeValue>
```

Schema template

The following is a template for a simple union type with two members. Additional member types may be added as needed.

```
<xs:simpleType name="NAMESimpleType">
  <xs:annotation>
    <xs:documentation>A data type for a ...</xs:documentation>
  </xs:annotation>
  <xs:union memberTypes="SIMPLE-TYPE-1 SIMPLE-TYPE-2"/>
</xs:simpleType>
```

JSON

JSON guidance is not yet available for union types.

NDR References

NDR 5.0

Rule	Applicability	Title
NDR 9-19	REF, EXT	No union member types of xs:ID
NDR 9-20	REF, EXT	No union member types of xs:IDREF
NDR 9-21	REF, EXT	No union member types of xs:IDREFS
NDR 9-22	REF, EXT	No union member types of xs:anySimpleType
NDR 9-23	REF, EXT	No union member types of xs:ENTITY
NDR 9-24	REF, EXT	No union member types of xs:ENTITIES
NDR 11-7	REF, EXT	Union member types defined by conformant schemas

NDR 4.0

Rule	Applicability	Title
NDR 9-19	REF, EXT	No union member types of xs:ID
NDR 9-20	REF, EXT	No union member types of xs:IDREF
NDR 9-21	REF, EXT	No union member types of xs:IDREFS
NDR 9-22	REF, EXT	No union member types of xs:anySimpleType
NDR 9-23	REF, EXT	No union member types of xs:ENTITY
NDR 9-24	REF, EXT	No union member types of xs:ENTITIES
NDR 11-6	REF, EXT	Union member types defined by conformant schemas

Codes / Facets

Facets are additional constraints that may be defined on simple types (like strings or numbers) to limit the allowable values.

They can do such things as limit a string to a pre-defined code list, constrain a number to a given range, or create a pattern that must be followed.

- A telephone number type might define a facet that creates the pattern (ddd) ddd-dddd.
- A week-day type might define facets that constrain a string down to the codes MON, TUES, WED, etc.
- A latitude type might define facets that constrain a number to the range -90 to 90.

- Overview
  - Kinds of Facets
  - Enumerations
  - Multiple facets on a type
  - Linking properties to facets
- Modeling guidance
  - Simple type names
- XML
  - Enumeration Example
  - Numeric Range Example
  - Generic Template
- JSON
  - Kinds of JSON Facets
  - Enumerations
  - Numeric Range
  - Generic Template
- NDR references

Overview

A facet is defined by a simple type by specifying its **kind**, **value**, and **definition**.

The kind of facet must be chosen from the list provided below; the facet value and definition are provided by the data modeler.

Kinds of Facets

Facet	Description	Text	Num	Date
enumeration	a code; an allowable value	X	X	X
fractionDigits	the maximum number of decimal places allowed		X	
length	the exact number of characters for a value	X		
maxLength	the maximum number of characters for a value	X		
minLength	the minimum number of characters for a value	X		

Facet	Description	Text	Num	Date
minExclusive	the lower bound (the value must be greater than this constraint)		X	X
minInclusive	the lower bound (the value must be greater than or equal to this constraint)		X	X
maxExclusive	the upper bound (the value must be less than this constraint)		X	X
maxInclusive	the upper bound (the value must be less than or equal to this constraint)		X	X
pattern	the regular expression that the value must match	X	X	X
totalDigits	the maximum number of digits allowed		X	
whiteSpace	the way white space (spaces, tabs, line feeds, carriage returns) will be handled	X		

These kinds of facets and the term "facet" itself are taken from XML Schema: Constraining or Non-fundamental facets.

Enumerations

In NIEM, the overwhelming majority (99%+) of facets that are created are **enumerations** (codes).

Because the word "code" is more well-known, it is sometimes used in place of "facet" or "enumeration."

A type creating a code set for days of the week might be based on a string and then define seven enumerations: **MON**, **TUE**, **WED**, etc. Each enumeration constrains that string to a valid option. That one type will have seven valid options for its value.

Multiple facets on a type

Multiple facets may be defined by a single simple type:

- A simple type that defines a code set must add one enumeration facet per code value.
- A simple type that defines a numeric range must add one minInclusive (or minExclusive) facet to specify the lower bound, and one maxInclusive (or maxExclusive) facet to specify the upper bound.

Examples:

Type	Facet Kind	Facet Value	Facet Definition
MonthCodeSimpleType			
	enumeration	JAN	January
	enumeration	FEB	February
	enumeration	MAR	March
	…	…	…
LatitudeSimpleType			
	minInclusive	-90	south pole
	maxInclusive	90	north pole

Linking properties to facets

A property does not define facets for itself. A property may have a type that defines facets.

For example, property `nc:TelephoneNumberCategoryCode`

- has type `xCard:TelephoneCategoryCodeType`, and
- that type is based on a simple type that defines nine enumerations including “cell,” “home,” and “work.”

The property defined in Core is thus linked to the facets defined in the xCard code set namespace. In a message instance, only those nine enumerations would be valid values for `nc:TelephoneNumberCategoryCode`.

Modeling guidance

Facets do not exist independently. They are defined as part of the simple type that they constrain.

- Each facet will have a kind.
- Each facet will have a value.
- Enumeration facets must have a definition.

A simple type that defines month codes will declare 12 facets:

- The facet kinds will each be **enumeration**.
- The facet values will be **JAN**, **FEB**, **MAR**, etc.
- The facet definitions will be **January**, **February**, **March**, etc.

Keep in mind that overly-restrictive facets limit reusability, particularly in NIEM release schemas. For example, facets that restrict a string to the number of characters allowed in a corresponding database field may limit other users with different systems.

Simple type names

A simple type declares a facet:

- If the type contains enumerations, the name of the type must end with `CodeSimpleType`.
- All other simple types must have names that end with with `SimpleType`.

XML

Enumeration Example

This example shows a simple type with two enumerations: `MonthCodeSimpleType` with codes for **JAN** and **FEB**.

This is enough to show how to create a code set, but ten more enumeration statements would need to be

added to make this type example complete.

```
<xs:simpleType name="MonthCodeSimpleType">
   <xs:annotation>
    <xs:documentation>A data type for month codes.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:token">
    <xs:enumeration value="JAN">
     <xs:annotation>
      <xs:documentation>January</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="FEB">
     <xs:annotation>
      <xs:documentation>February</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
  </xs:restriction>
</xs:simpleType>
```

Because these facets are enumerations, the type name must end with `CodeSimpleType`.

Numeric Range Example

This example shows a numeric simple type with a range of 0 to 100, inclusive.

```
<xs:simpleType name="TestResultSimpleType">
   <xs:annotation>
    <xs:documentation>A data type for a range of valid test result values.</xs
  </xs:annotation>
  <xs:restriction base="xs:decimal">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="100"/>
  </xs:restriction>
</xs:simpleType>
```

- The `xs:restriction base` may be replaced with any numeric simple type, like `xs:integer`.
- The `minInclusive` may be replaced with `minExclusive`.
- The `maxInclusive` may be replaced with `maxExclusive`.
- Remove either the min or max facet in order to create a range that is unbounded on one side.

Generic Template

The template below defines a simple type with a facet.

```
<xs:simpleType name="NAMESimpleType">
   <xs:annotation>
    <xs:documentation>A data type for a(n) ...</xs:documentation>
```

```
  </xs:annotation>
  <xs:restriction base="BASE">
    <xs:KIND value="VALUE">
     <xs:annotation>
      <xs:documentation>DEFINITION</xs:documentation>
      </xs:annotation>
    </xs:KIND>
  </xs:restriction>
</xs:simpleType>
```

- Common base types are `xs:token` (a string that does not allow irregular whitespace), `xs:decimal`, and `xs:integer`.
- Common kinds of facets are `xs:enumeration`, `xs:pattern`, `xs:minInclusive`, `xs:minExclusive`, `xs:maxInclusive`, and `xs:maxExclusive`. See the full list.
- Names, values, and definitions must be provided by the data modeler.

JSON

Kinds of JSON Facets

XML Facet	JSON Facet	Text	Num	Date
enumeration	enum	X		
fractionDigits	n/a			
length	minLength maxLength	X		
maxLength	maxLength	X		
minLength	minLength	X		
minExclusive	minimum exclusiveMinimum: true		X	
minInclusive	minimum		X	
maxExclusive	maximum exclusiveMaximum: true		X	
maxInclusive	maximum		X	
pattern	pattern	X		
totalDigits	n/a			
whiteSpace	n/a			

Draft 6 of the JSON Schema specification changes the types of `exclusiveMinimum` and `exclusiveMaximum` from booleans to numbers so that they may be used without also requiring `minimum` and `maximum`.

Enumerations

Enumerations are defined within a simple type declaration in the `oneOf` array. Each enumeration is defined as an object with `enum` and `description` properties.

Defining enumerations in an array is a JSON Schema requirement. Since NIEM requires definitions for enumerations, the object must be repeated with a single enumeration in each `enum` array and a corresponding `description`.

```
{
  "oneOf" : [
    { "enum": [ "1" ], "description": "…" },
    { "enum": [ "2" ], "description": "…" },
    { "enum": [ "3" ], "description": "…" }
  ]
}
```

This example shows the definition of `iso_4217:CurrencyCodeSimpleType` with three enumerations:

```
{
  "definitions": {
    "iso_4217:CurrencyCodeSimpleType": {
      "type": "string",
      "description": "A data type for a currency that qualifies a monetary amo
      "oneOf": [
        {
          "enum": [ "EUR" ],
          "description": "Euro"
        },
        {
          "enum": [ "GBP" ],
          "description": "Pound Sterling"
        },
        {
          "enum": [ "USD" ],
          "description": "US Dollar"
        }
      ]
    }
  }
}
```

Numeric Range

This example shows a numeric simple type with a minimum and a maximum value:

```
{
  "definitions": {
    "my:CustomRangeSimpleType": {
      "description": "A data type for ...",
      "type": "integer",
      "minimum": 0,
      "maximum": 100
```

```
    },
  }
}
```

- The base type may be set to `number` instead of `integer` to allow for decimal values.
- Add the properties `exclusiveMinimum: true` and `exclusiveMaximum: true` to make the range exclusive of the edge values.

Generic Template

This template shows the definition of a simple type with a base:

```
{
  "definitions": {
    "SIMPLE_TYPE": {
      "description": "TYPE_DEFINITION",
      "type": "BASE_TYPE",
      "FACET_KIND": "FACET_VALUE"
    },
  }
}
```

- SIMPLE_TYPE - this should be a qualified name that ends with `SimpleType`.
- BASE_TYPE - this should be either **string**, **number**, or **integer**.
- FACET_KIND - this should be one of the values defined in the JSON facets table above.

Example of a string type with a maximum length:

```
{
  "definitions": {
    "ext:NAMESimpleType": {
      "description": "A data type for...",
      "type": "string",
      "maxLength": 20
    },
  }
}
```

NDR references

NDR 5.0

- NDR Section 11.1.2.3 - Code simple types

Rule	Applicability	Title
NDR 9-14	REF, EXT	Enumeration has data definition

Rule	Applicability	Title
NDR 9-13	REF	No use of fixed on simple type facets
NDR 10-17	REF, EXT	Name of code type ends in CodeType
NDR 10-18	REF, EXT	Code type corresponds to a code list
NDR 11-8	REF, EXT	Name of a code simple type ends in CodeSimpleType
NDR 11-9	REF, EXT	Code simple type corresponds to a code list

NDR 4.0

- NDR Section 11.1.2.3 - Code simple types

Rule	Applicability	Title
NDR 9-14	REF, EXT	Enumeration has data definition
NDR 9-13	REF	No use of fixed on simple type facets
NDR 10-17	REF, EXT	Name of code type ends in CodeType
NDR 10-18	REF, EXT	Code type corresponds to a code list
NDR 11-7	REF, EXT	Name of a code simple type ends in CodeSimpleType
NDR 11-8	REF, EXT	Code simple type corresponds to a code list

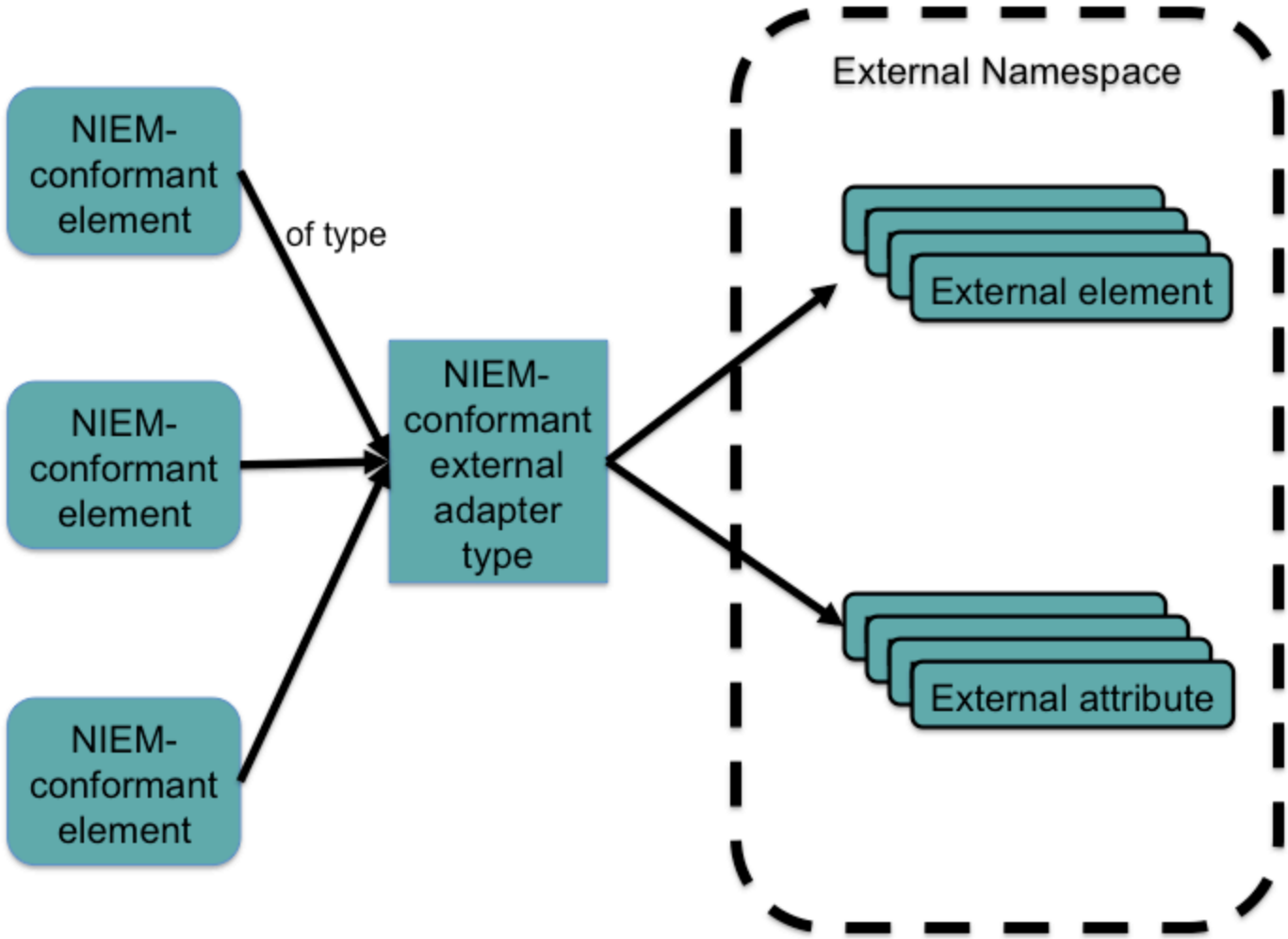
Adapters

Adapters are the mechanism NIEM uses to allow content from external standards without causing conformance validation errors.

- Modeling guidance
  - Adapter flag
  - Parent type
  - External content
  - Extra documentation
  - No extensions
  - No external ID attributes
  - IEPD exceptions
    - External attributes
- XML
  - XML example
  - XML Schema example
  - XML Schema template
- JSON
- NDR references

An adapter is a type that contains an external property. It is marked as an adapter in the schema so that NIEM conformance rules skip over its external (non-conformant) contents instead of throwing errors. Otherwise, an adapter type looks and acts like a standard NIEM type.

> `geo:SurfaceType` is an adapter type in NIEM that contains element `Surface` from GML, an external standard. It can be used normally by other NIEM components. Because it is an adapter type, NDR conformance tests will not throw errors if the element from the GML external standard does not follow NIEM rules.



Modeling guidance

Adapter flag

An adapter type must have attribute `appinfo:externalAdapterTypeIndicator` with a value of `true` in its schema declaration.

Parent type

An adapter type must extend `structures:ObjectType`.

External content

The sub-properties from an adapter type must be from external namespaces.

An adapter type simply wraps external content; it should not be used to build larger objects. Other NIEM types may be created that combine regular content with elements that have adapter types.

Extra documentation

NIEM requires that all namespaces and components be documented. Because external standards might not have this documentation, or might not have this documentation in a consistent manner as NIEM, NIEM requires all usages of external namespaces and components to be documented.

No extensions

Adapter types may not be extended by other types. Each adapter type is meant to represent a single concept from an external namespace and stand alone.

No external ID attributes

NIEM defines structures:id to enable references between components. For consistency, the use of external attributes with an ID type is not allowed so as not to bypass this standard mechanism.

IEPD exceptions

External attributes

In an extension namespace, a non-adapter type is allowed to directly contain a reference to an external attribute as long as a definition is also provided with the reference.

XML

XML example

Adapters appear in XML instances like other content.

```
<nc:LocationArea>
  <nc:LocationArea>
    <geo:Ellipse>    <!-- NIEM element with an adapter type -->
      <xls:Ellipse>  <!-- External element from GML -->
        <!-- GML content -->
      </xls:Ellipse>
    </geo:Ellipse>
  </nc:LocationArea>
</nc:Location>
```

XML Schema example

The following is a subset from the Geospatial adapter schema in the NIEM 4.0 release.

- The import statement of the GML external namespace includes a definition and attribute `appinfo:externalImportIndicator="true"`.
- The complex type declaration is an adapter type, with attribute `appinfo:externalAdapterTypeIndicator="true`.

- The reference to the external element `gml:AbstractGeometry` also includes a definition.
- Nothing special is required by the element below that uses this adapter type. The type can now be used like any other type.

```
<xs:schema xmlns:gml="http://www.opengis.net/gml/3.2" ...>
  <xs:import schemaLocation="../../../external/ogc/gml/3.2.1/gml.xsd" namespac
    <xs:annotation>
      <xs:documentation>Geography Markup Language (GML) version 3.2.1 schemas.
    </xs:annotation>
  </xs:import>

  <xs:complexType name="GeometryType" appinfo:externalAdapterTypeIndicator="tr
    <xs:annotation>
      <xs:documentation>A data type that encapsulates a GML geometry element.<
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="structures:ObjectType">
        <xs:sequence>
          <xs:element ref="gml:AbstractGeometry" minOccurs="1" maxOccurs="1">
            <xs:annotation>
              <xs:documentation>The AbstractGeometry element is the abstract h
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:element name="Geometry" type="geo:GeometryType" nillable="true">
    <xs:annotation>
      <xs:documentation>A general-purpose GML geometry adapter.</xs:documentat
    </xs:annotation>
  </xs:element>

</xs:schema>
```

XML Schema template

```
<xs:schema>
  <xs:import schemaLocation="PATH" namespace="URI" appinfo:externalImportIndic
    <xs:annotation>
      <xs:documentation>EXTERNAL NAMESPACE DEFINITION</xs:documentation>
    </xs:annotation>
  </xs:import>

  <xs:complexType name="NAMEType" appinfo:externalAdapterTypeIndicator="true">
    <xs:annotation>
      <xs:documentation>DEFINITION</xs:documentation>
    </xs:annotation>
```

```
    <xs:complexContent>
      <xs:extension base="structures:ObjectType">
        <xs:sequence>
          <xs:element ref="EXTERNAL_ELEMENT" minOccurs="1" maxOccurs="1">
            <xs:annotation>
              <xs:documentation>EXTERNAL_ELEMENT_DEFINITION</xs:documentation>
            </xs:annotation>
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

</xs:schema>
```

JSON

Guidance is not yet available for adapters and the use of external standards in JSON.

NDR references

NDR 5.0

- NDR Section 10.2.3 - External adapter types and external components

Rule	Applicability	Title
NDR 10-8	REF, EXT	External adapter type has indicator
NDR 10-9	REF, EXT	Structure of external adapter type definition follows pattern
NDR 10-10	REF, EXT	Element use from external adapter type defined by external schema documents
NDR 10-11	REF, EXT	External adapter type not a base type
NDR 10-12	SET	External adapter type not a base type
NDR 10-13	REF	External attribute use only in external adapter type
NDR 10-14	REF, EXT	External attribute use has data definition
NDR 10-15	SET	External attribute use not an ID
NDR 10-16	REF, EXT	External element use has data definition
NDR 10-71	REF, EXT	External adapter type indicator annotates complex type
NDR	REF, EXT	appinfo:appliesToTypes annotates metadata element

Rule	Applicability	Title
10-72		
NDR 10-73	SET	appinfo:appliesToTypes references types
NDR 10-74	REF, EXT	appinfo:appliesToElements annotates metadata element
NDR 10-75	SET	appinfo:appliesToElements references elements

NDR 4.0

- NDR Section 10.2.3 - External adapter types and external components

Rule	Applicability	Title
NDR 10-8	REF, EXT	External adapter type has indicator
NDR 10-9	REF, EXT	Structure of external adapter type definition follows pattern
NDR 10-10	REF, EXT	Element use from external adapter type defined by external schema documents
NDR 10-11	REF, EXT	External adapter type not a base type
NDR 10-12	SET	External adapter type not a base type
NDR 10-13	REF	External attribute use only in external adapter type
NDR 10-14	REF, EXT	External attribute use has data definition
NDR 10-15	SET	External attribute use not an ID
NDR 10-16	REF, EXT	External element use has data definition
NDR 10-70	REF, EXT	External adapter type indicator annotates complex type
NDR 10-71	REF, EXT	appinfo:appliesToTypes annotates metadata element
NDR 10-72	SET	appinfo:appliesToTypes references types
NDR 10-73	REF, EXT	appinfo:appliesToElements annotates metadata element
NDR 10-74	SET	appinfo:appliesToElements references elements

Associations

Associations are relationships between properties. An association may have additional characteristics that further describe the relationship.

- Modeling guidance
  - Association element
    - Name
    - Definition
    - Type
  - Association type
    - Name
    - Definition
    - Parent type
- XML
  - Instance example
    - Inline
    - Reference
  - Schema example
  - Schema template
- JSON
  - Instance example
    - Inline
    - Reference
  - Schema example
  - Schema template
- NDR references

A marriage can be an association between two people, with additional characteristics like marriage date.

Modeling guidance

Association element

Name

The name of an association element must end with the representation term “Association”.

Definition

The definition of an association element should begin with the phrase:

```
An (optional adjectives) (relationship|association)...
```

Type

The type of an association element must be an association type.

Association type

Name

The name of an association type must end with the representation term “AssociationType”.

Definition

The definition of an association should begin with the phrase:

Parent type

An association type must extend `structures:AssociationType` or a type derived from it.

Typically, association types extend `nc:AssociationType`, which meets this requirement plus adds properties to capture the beginning and end date of the association.

```
A data type for (a relationship|an association)...
```

XML

Instance example

- The following two examples show an association between a person and a telephone number.
- The date range and description elements are additional properties relating to this association.

Inline

The content of the elements in this association are inline.

```
<scr:PersonPhoneAssociation>

  <nc:AssociationDateRange>
    <nc:StartDate>
      <nc:Date>2015-01-01</nc:Date>
    </nc:StartDate>
    <nc:EndDate>
      <nc:Date>2016-06-30</nc:Date>
    </nc:EndDate>
  </nc:AssociationDateRange>

  <nc:AssociationDescriptionText>The person made multiple calls to the given t
```

```
  <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>
```

Reference

The person element in this association is defined as a reference to a person described elsewhere in this document, outside of the association. It can be common for one or more properties to be represented as references rather than inline.

```
<scr:PersonPhoneAssociation>

  <nc:AssociationDateRange>
    <nc:StartDate>
      <nc:Date>2015-01-01</nc:Date>
    </nc:StartDate>
    <nc:EndDate>
      <nc:Date>2016-06-30</nc:Date>
    </nc:EndDate>
  </nc:AssociationDateRange>

  <nc:AssociationDescriptionText>The person made multiple calls to the given t

  <nc:Person structures:ref="P1" xsi:nil="true"/>

  <nc:TelephoneNumber>
    <nc:FullTelephoneNumber>
      <nc:TelephoneNumberFullID>212-555-1212</nc:TelephoneNumberFullID>
    </nc:FullTelephoneNumber>
  </nc:TelephoneNumber>
</scr:PersonPhoneAssociation>

<nc:Person structures:id="P1">
  <nc:PersonName>
    <nc:PersonFullName>John Doe</nc:PersonFullName>
  </nc:PersonName>
</nc:Person>
```

Schema example

This example shows the creation of an association type and element.

For both the inline and the reference examples above, the schema will be created the same way. IDs and references are inherited by or added to all NIEM element types, so no further actions are required for their use in an instance.

- The association type…
  - name ends with “AssociationType”
  - definition begins with “A data type for a relationship”
  - extends nc:AssociationType, allowing it to inherit the association date range and description properties
- The association element…
  - name ends with “Association”
  - definition begins with “An association …”

```
<xs:complexType name="PersonPhoneAssociationType">
  <xs:annotation>
    <xs:documentation>A data type for a relationship between a PERSON and a PH
  </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="unbounde
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="PersonPhoneAssociation" type="scr:PersonPhoneAssociationType
  <xs:annotation>
    <xs:documentation>An association between a PERSON and a PHONE.</xs:documen
  </xs:annotation>
</xs:element>
```

Schema template

The template below declares an association type and element.

Multiple properties may be added to the association type to capture the primary objects of the association, and optionally any additional properties that further describe the association.

```
<xs:complexType name="NAMEAssociationType">
  <xs:annotation>
    <xs:documentation>A data type for a relationship ...</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
```

```
    <xs:extension base="nc:AssociationType">
      <xs:sequence>
        <xs:element ref="ASSOCIATION-OBJECT" minOccurs="0" maxOccurs="unbounde
        <xs:element ref="ASSOCIATION-PROPERTY" minOccurs="0" maxOccurs="unbour
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="NAMEAssociation" type="scr:NAMEAssociationType" nillable="tr
  <xs:annotation>
    <xs:documentation>An association ...</xs:documentation>
  </xs:annotation>
</xs:element>
```

JSON

Instance example

- The following two examples show an association between a person and a telephone number.
- The date range and description elements are additional properties relating to this association.

Inline

The content of the elements in this association are inline.

```
{
  "scr:PersonPhoneAssociation": {
    "nc:AssociationDateRange": {
      "nc:StartDate": {
        "nc:Date": "2015-01-01"
      },
      "nc:EndDate": {
        "nc:Date": "2016-06-30"
      }
    },

    "nc:AssociationDescriptionText": "The person made multiple calls to the gi

    "nc:Person": {
      "nc:PersonName": {
        "nc:PersonFullName": "John Doe"
      }
    },

    "nc:TelephoneNumber": {
      "nc:FullTelephoneNumber": {
        "nc:TelephoneNumberFullID": "212-555-1212"
```

```
      }
    }
  }
}
```

Reference

The person element in this association is defined as a reference to a person described elsewhere in this document, outside of the association. It can be common for one or more properties to be represented as references rather than inline.

```
{
  "scr:PersonPhoneAssociation": {
    "nc:AssociationDateRange": {
      "nc:StartDate": {
        "nc:Date": "2015-01-01"
      },
      "nc:EndDate": {
        "nc:Date": "2016-06-30"
      }
    },

    "nc:AssociationDescriptionText": "The person made multiple calls to the gi

    "nc:Person": {
      "@id": "P1"
    },

    "nc:TelephoneNumber": {
      "nc:FullTelephoneNumber": {
        "nc:TelephoneNumberFullID": "212-555-1212"
      }
    }
  },

  "nc:Person": {
    "@id": "P1"
    "nc:PersonName": {
      "nc:PersonFullName": "John Doe"
    }
  }
}
```

Schema example

This example shows the creation of an association type and element.

For both the inline and the reference examples above, the schema will be created the same way.

- The association type…
    - name ends with “AssociationType”
    - definition begins with “A data type for a relationship”
    - extends nc:AssociationType, allowing it to inherit the association date range and description properties
- The association element…
    - name ends with “Association”
    - definition begins with “An association …”

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "properties": {
    "scr:PersonPhoneAssociation": {
      "description": "An association between a PERSON and a PHONE.",
      "oneOf": [
        {
          "$ref": "#/definitions/scr:PersonPhoneAssociation"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/scr:PersonPhoneAssociation"
          }
        }
      ]
    }
  }

  "definitions": {
    "scr:PersonPhoneAssociationType": {
      "description": ">A data type for a relationship between a PERSON and a P
      "allOf": [
        {
          "$ref": "#/definitions/nc:AssociationType"
        },
        {
          "type": "object",
          "properties": {
            "nc:Person": {
              "$ref": "#/properties/nc:Person"
            },
            "nc:TelephoneNumber": {
              "$ref": "#/properties/nc:TelephoneNumber"
            }
          }
        }
      ]
    }
  }
```

```
}
```

Schema template

The template below declares an association type and element.

Multiple properties may be added to the association type to capture the primary objects of the association, and optionally any additional properties that further describe the association.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "properties": {
    "NAMEAssociation": {
      "description": "An association between ...",
      "oneOf": [
        {
          "$ref": "#/definitions/NAMEAssociation"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/NAMEAssociation"
          }
        }
      ]
    }
  }

  "definitions": {
    "NAMEAssociationType": {
      "description": ">A data type for a relationship ...",
      "allOf": [
        {
          "$ref": "#/definitions/nc:AssociationType"
        },
        {
          "type": "object",
          "properties": {
            "ASSOCIATION-OBJECT": {
              "$ref": "#/properties/ASSOCIATION-OBJECT"
            },
            "ASSOCIATION-PROPERTY": {
              "$ref": "#/properties/ASSOCIATION-PROPERTY"
            }
          }
        }
      ]
    }
  }
```

}

NDR references

NDR 5.0

- NDR Section 10.3 - Associations

Rule	Applicability	Title
NDR 10-21	REF, EXT	Association type derived from structures:AssociationType
NDR 10-22	REF, EXT	Association element type is an association type
NDR 11-34	REF, EXT	Standard opening phrase for association element data definition
NDR 11-43	REF, EXT	Standard opening phrase for association type data definition

NDR 4.0

- NDR Section 10.3 - Associations

Rule	Applicability	Title
NDR 10-21	REF, EXT	Association type derived from structures:AssociationType
NDR 10-22	REF, EXT	Association element type is an association type
NDR 11-32	REF, EXT	Standard opening phrase for association element data definition
NDR 11-41	REF, EXT	Standard opening phrase for association type data definition

Augmentations

Augmentations are NIEM's mechanism to add additional content to a type defined by another namespace.

- Type extension issues
- Basic Approach
- Example
- When to use
- IEPD exceptions
- See more

Augmentations are done via element substitution (in XML). Almost every type with complex content (sub-properties) in NIEM contains an augmentation point element. These augmentation point elements serve solely as hooks, to be later replaced by additional content defined elsewhere.

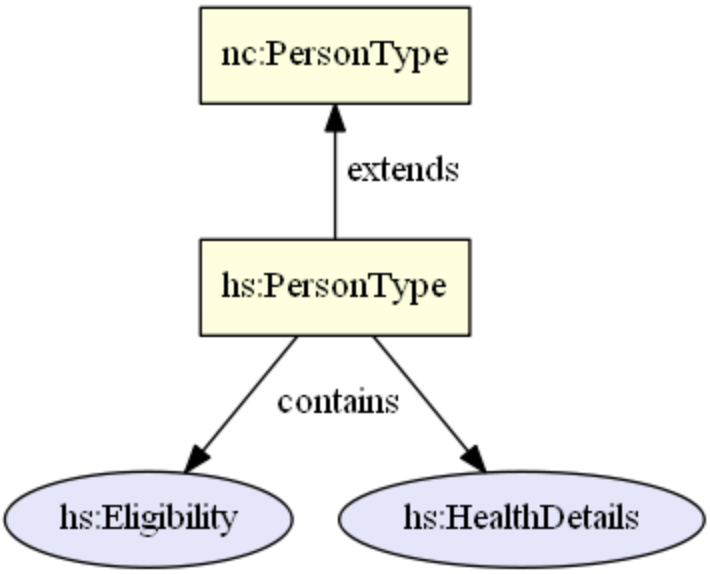
> If a domain or an IEPD wanted to define additional properties about a person, it would create a new augmentation element and substitute it for `nc:PersonAugmentationPoint` (contained by `nc:PersonType`). Any place nc:PersonType is used, the new augmentation may be used as well.

An augmentation may be either the new property to add (e.g., `my:PersonCustomCode`) or a container (e.g., `my:PersonAugmentation`) that has a set of sub-properties.

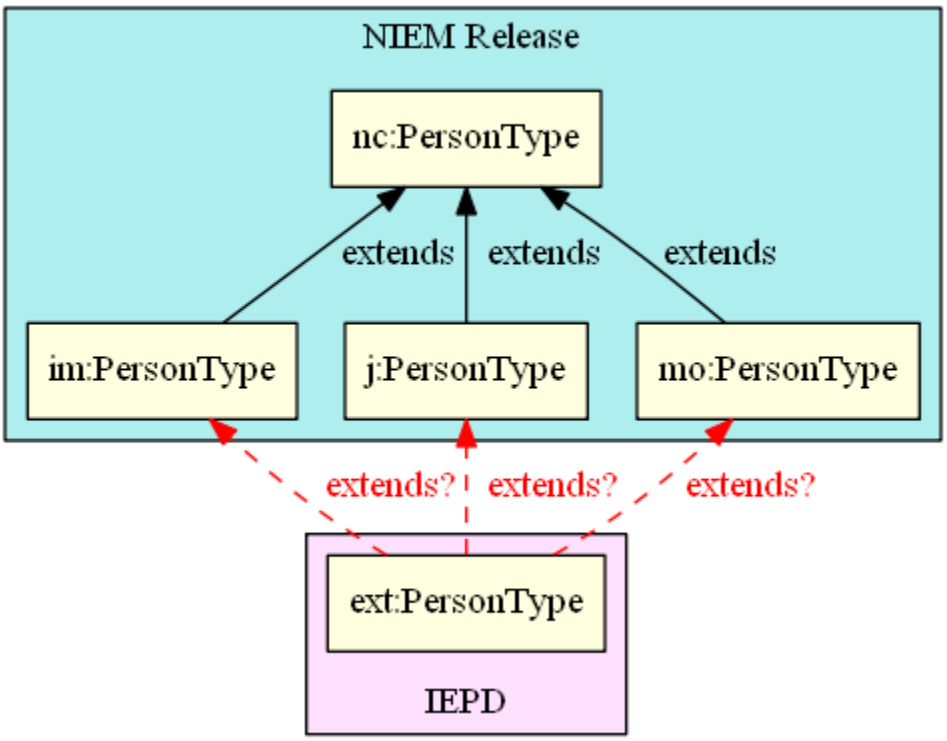
Adding additional content to a type is typically done using type extension, but NIEM abandoned that approach due to its limitations. Type extension is limited to a single parent type; augmentations from several different namespaces can be used together because element substitution allows multiple occurrences.

Type extension issues

For example, for a domain to add additional content to the Core `PersonType` using extension, a new domain type would be declared that extended `nc:PersonType`, and domain properties would be added to it.



The issue with this is that type extension does not allow for multiple inheritance. If more than one domain extends `nc:PersonType`, IEPD developers cannot use those related but separate extensions together. They must choose a single type to reuse or derive from.

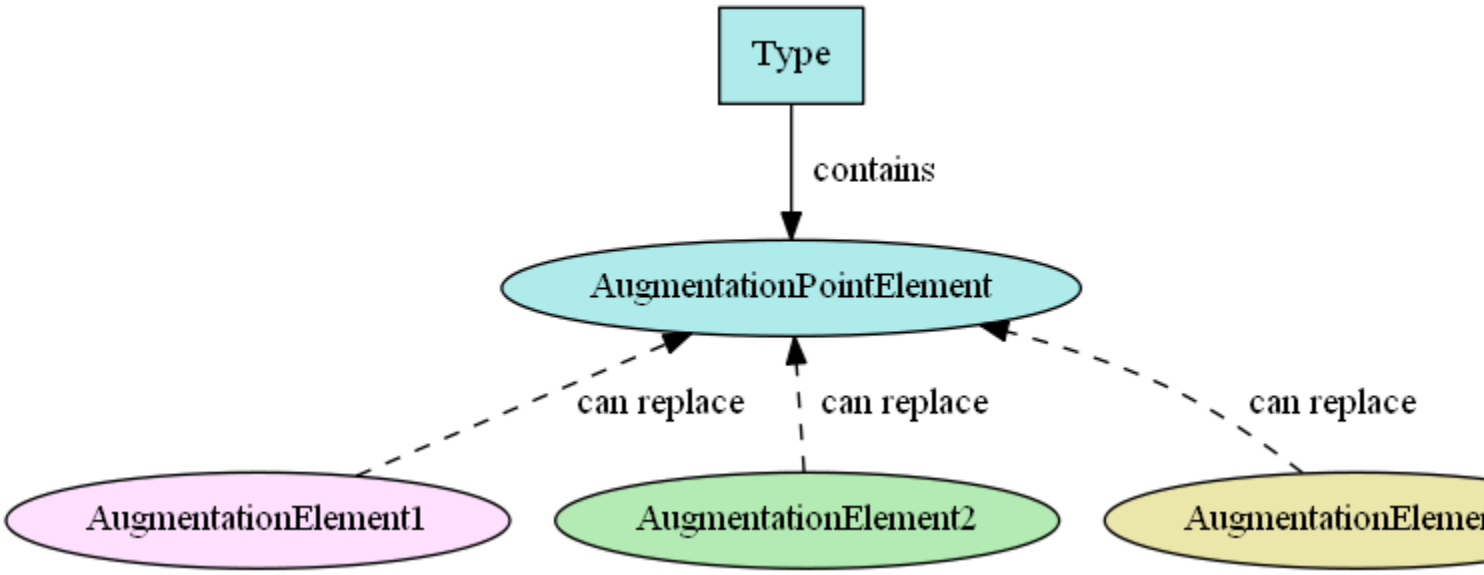


Additionally, type derivation can cause the unintended side effect of cascading extensions. If you are unable to use type substitution in your instance, then once you extend the type you want, you will also need to create a new property of that new type. If the original property is contained by another type you are using, you would then need to extend that type, and then the property of that type, and so on, all the way up the chain.

Augmentations were introduced in NIEM 2.0 to address these issues raised by the NIEM community, and revised in NIEM 3.0 to make them easier to use.

Basic Approach

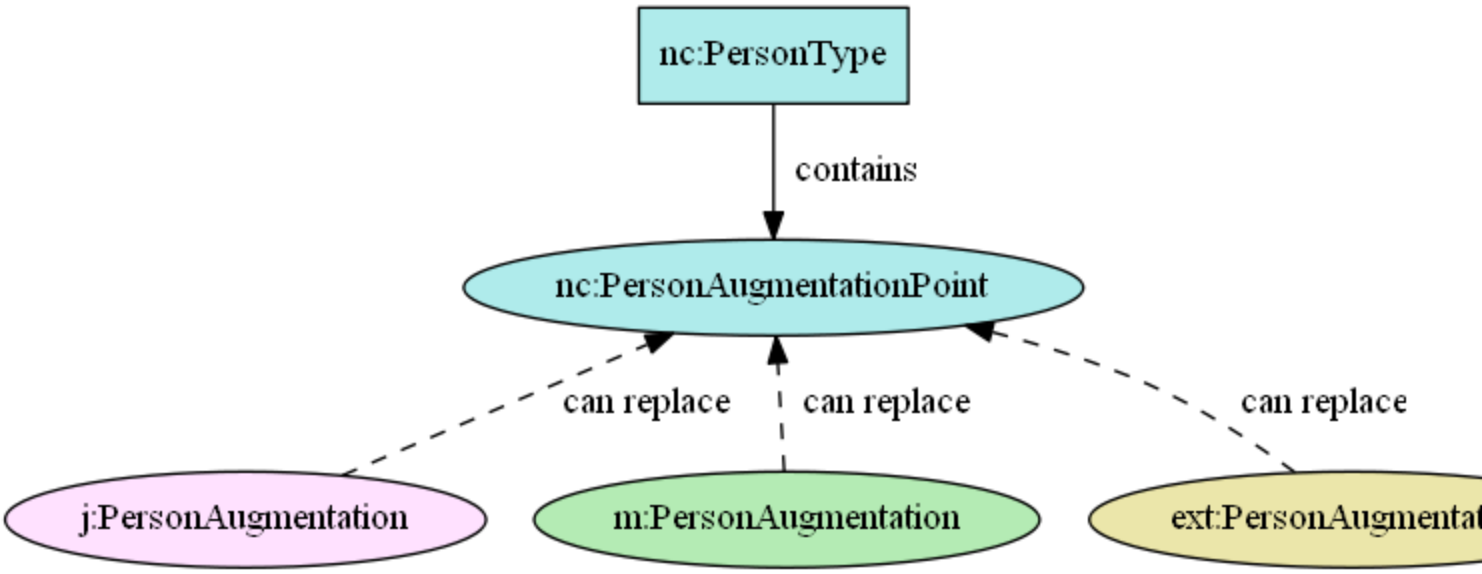
The augmentation approach has two parts to it: the additional content itself, and the means by which it can be used with other types. **Augmentation elements** are the additional content defined in other namespaces, and **augmentation point elements** are the means that make this approach work, via substitution. Multiple augmentation elements can be substituted and used together.



- Type and AugmentationPointElement are defined in the same namespace.
- AugmentationElements 1, 2, and 3 are each defined in separate namespaces.

Example

This example shows the diagram above with actual properties and types from NIEM:



The augmentation element j:PersonAugmentation contains additional person-related properties that the Justice domain may need. Similar augmentation elements are created here for the Maritime domain and an IEPD extension namespace. These augmentations can be used separately or together any time nc:PersonType is used.

When to use

Augmentations should be created only when trying to add additional content to an existing type from a namespace outside of your control. If you find a type that has the right semantics and almost meets your requirements, but it is missing some properties that you need, an augmentation should be used to add the missing content.

Specialization in NIEM is still handled via type extension. Specialization is different because it doesn't just deal with some missing properties - the existing type itself is not the right semantic match for the requirements.

> If a user wants to model a train with additional properties like whistle, locomotive, engineer, and caboose, nc:VehicleType might be the closest match but it isn't an exact match. The semantics are different. All trains may be vehicles, but not all vehicles are trains. Rather than creating an augmentation for nc:VehicleType, a new TrainType should be created that extends nc:VehicleType and adds the train-specific properties.

Using type extension for specialization does not present the same problems that augmentations were designed to solve. Multiple augmentations for a type may be used together in an IEPD because they are all meant to provide additional content for the same thing. Specializations do not need to be able to be combined with other specializations in instances - a regular vehicle property does not need to contain train, boat, and airplane properties.

IEPD exceptions

Because IEPDs do not have to be built with the same degree of flexibility and reusability as NIEM

release schemas, they may use either augmentations or extensions. Augmentations may be easier to use in certain circumstances, such as trying to add additional content to a nested object which might otherwise require cascading extensions.

See more

This section is split into two sub-sections:

- Augmentation Points

  An augmentation point is a special substitution group to allow for later replacement by additional content from other namespaces.

- Augmentation Elements

  An augmentation element is additional content that replaces an augmentation point defined in another namespace.

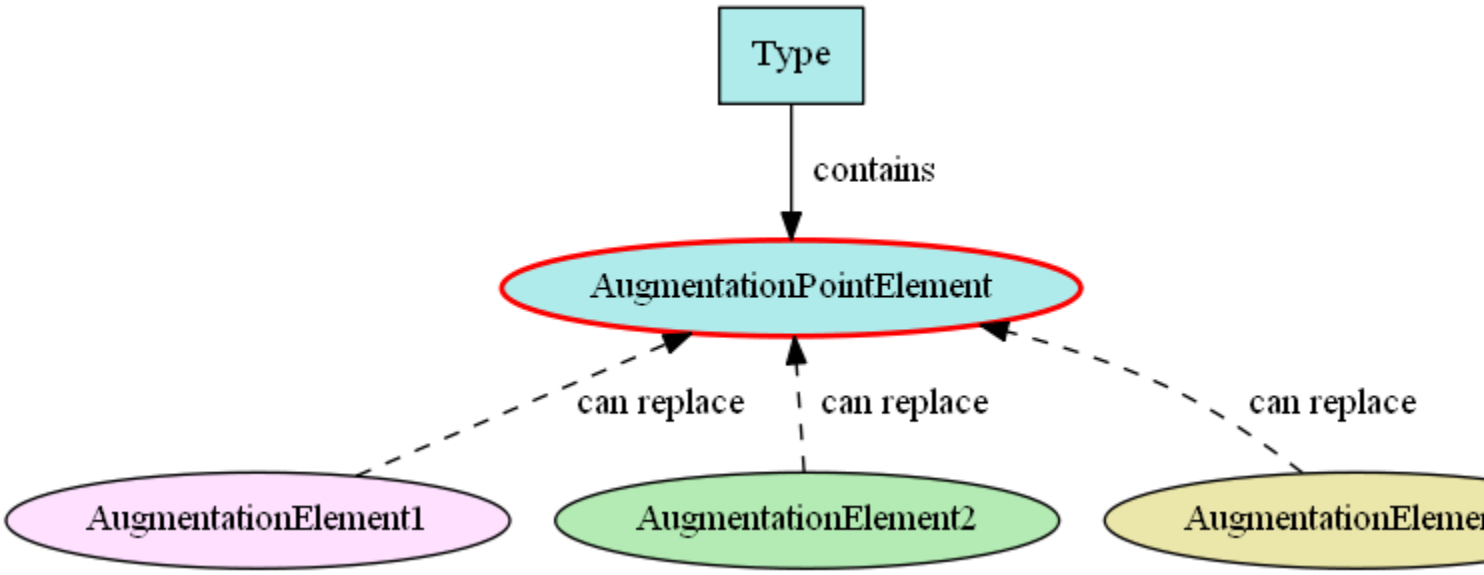
Augmentation Points

An augmentation point is a special substitution group created to allow for later replacement by additional content from other namespaces.

- Modeling guidance
  - Name
  - Abstract
  - No type
  - Sub-element position
  - IEPD exceptions
- XML
  - Instances
  - Schema example
  - Schema template
  - Partial template
- JSON
- NDR references

Every NIEM release type that contains sub-elements also contains an augmentation point element. These elements are abstract, have no type, and cannot appear in any instance. They serve strictly as substitution points, to be replaced by other content - the augmentation elements.

The name of each augmentation point element corresponds to the type in which it appears, with the representation term `"Type"` being replaced by `"AugmentationPoint"`.



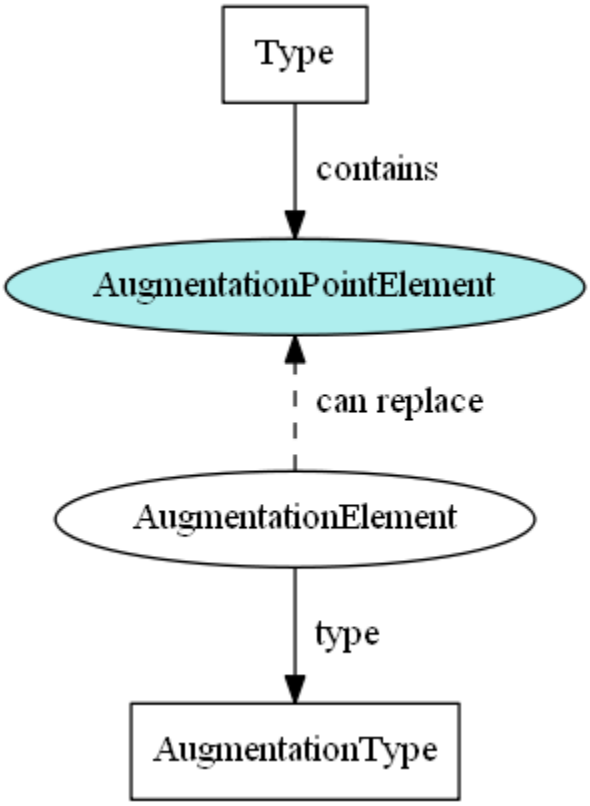
Examples

Each of the types in the table of examples below contain the augmentation point element listed next to it. The names and definitions follow a regular pattern.

Type	Augmentation Point Element	Definition of Augmentation Point Element
nc:PersonType	nc:PersonAugmentationPoint	An augmentation point for PersonType
nc:ContactInformationType	nc:ContactInformationAugmentationPoint	An augmentation point for ContactInformationType
em:AccessType	em:AccessAugmentationPoint	An augmentation point for AccessType
it:AgentType	it:AgentAugmentationPoint	An augmentation point for AgentType

Each of these augmentation point elements may be replaced in an instance with other content.

Modeling guidance



Name

The augmentation point element name must correspond with the type name it appears in, with the `"Type"` representation term being replaced by `"AugmentationPoint"`.

The name of the augmentation point element for `nc:PersonType` is `"nc:PersonAugmentationPoint"`.

Abstract

An augmentation point element has no content of its own and is merely a construct to support additional content from other namespaces. Making this element abstract ensures that it must be omitted or replaced in an instance by another element.

No type

An augmentation point element does not have a type. This ensures that there will not be a type conflict when it is later substituted by one or more augmentation elements.

Sub-element position

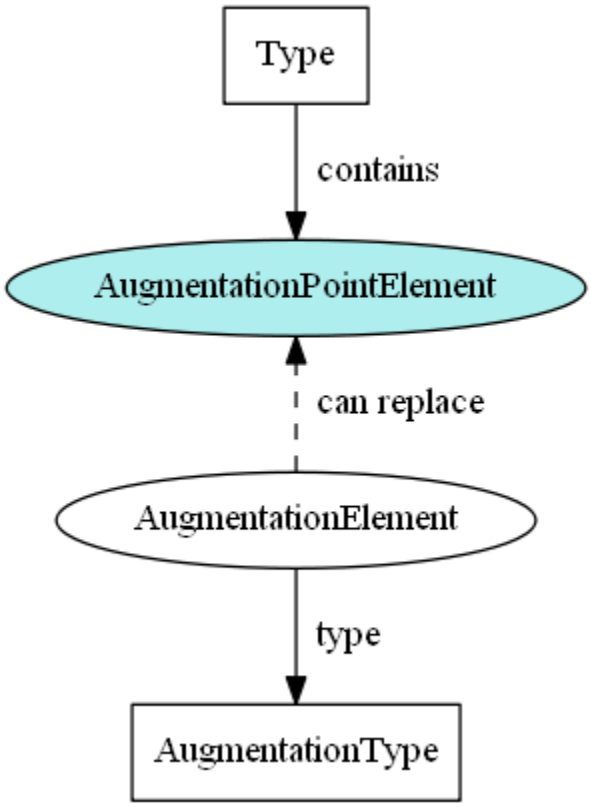
The augmentation point element must appear as the final sub-element in its type. This makes it easy to find and makes sure the type's original content appears in instances before the augmentations do.

IEPD exceptions

Augmentation point elements are required in NIEM release schemas, but not for IEPDs.

NIEM releases are required to define augmentation points in order to make it as easy as possible for IEPDs to later add additional content. IEPDs may, but are not required to, provide this same level of support.

XML



Instances

Augmentation point elements do not appear in XML instances because they are abstract. They may only be omitted or replaced by their concrete substitutions. See Augmentation Elements in XML for an example of how augmentation elements replace an augmentation point in an instance.

Schema example

This snippet, from a subset of Core, shows how augmentation point element `nc:PersonAugmentationPoint` is defined and referenced by `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="unbounde
        <xs:element ref="nc:PersonName" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="nc:PersonAugmentationPoint" minOccurs="0" maxOccurs="
      </xs:sequence>
    </xs:extension>
```

```
  </xs:complexContent>
</xs:complexType>

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

- `nc:PersonType` includes a reference to its augmentation point (`nc:PersonAugmentationPoint`) as its last sub-element.
- The augmentation point element is abstract and has no type.

Schema template

This template shows the declaration of a complex type that references its augmentation point, and the declaration of the augmentation point element itself.

```
<!-- NIEM type with sub-elements, like nc:PersonType -->
<xs:complexType name="NAMEType">
  <xs:annotation>
    <xs:documentation>A data type for a(n) {$Definition}</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="PARENT_TYPE">
      <xs:sequence>
        <xs:element ref="ELEMENT1" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="ELEMENT2" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="NAMEAugmentationPoint" minOccurs="0" maxOccurs="unbou
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<!-- Augmentation Point Element, like nc:PersonAugmentationPoint -->
<xs:element name="NAMEAugmentationPoint" abstract="true">
  <xs:annotation>
    <xs:documentation>An augmentation point for NAMEType</xs:documentation>
  </xs:annotation>
</xs:element>
```

Partial template

Augmentation points do not actually require that many lines to create. The majority of the template above shows how to create a type. Only one line in the type declaration actually involves the augmentation point.

Without the rest of the type declaration, the template would look like:

```
<!-- Complex type declaration -->
        <xs:element ref="NAMEAugmentationPoint" minOccurs="0" maxOccurs="unbou
<!-- End of complex type declaration -->

<!-- Augmentation Point Element, like nc:PersonAugmentationPoint -->
<xs:element name="NAMEAugmentationPoint" abstract="true">
  <xs:annotation>
    <xs:documentation>An augmentation point for NAMEType</xs:documentation>
  </xs:annotation>
</xs:element>
```

This is much simpler: one new augmentation point element, and one reference to it from its type.

JSON

JSON Schema does not have a concept similar to element substitution. Augmentation points are therefore not defined in NIEM JSON. Substitutions of augmentation elements should be made directly.

NDR references

NDR 5.0

- NDR Section 10.4.2 - Augmentation point element declarations
- NDR Section 10.4.3 - Augmentation point element use

Rule	Applicability	Title
NDR 10-25	REF, EXT	Augmentation point element corresponds to its base type
NDR 10-26	REF, EXT	An augmentation point element has no type
NDR 10-27	REF, EXT	An augmentation point element has no substitution group
NDR 10-28	REF, EXT	Augmentation point element is only referenced by its base type
NDR 10-29	REF	Augmentation point element use is optional
NDR 10-30	REF	Augmentation point element use is unbounded
NDR 10-31	REF, EXT	Augmentation point element use must be last element in its base type
NDR 11-31	REF, EXT	Standard opening phrase for augmentation point element data definition

NDR 4.0

- NDR Section 10.4.2 - Augmentation point element declarations
- NDR Section 10.4.3 - Augmentation point element use

Rule	Applicability	Title
NDR 10-25	REF, EXT	Augmentation point element corresponds to its base type
NDR 10-26	REF, EXT	An augmentation point element has no type
NDR 10-27	REF, EXT	An augmentation point element has no substitution group

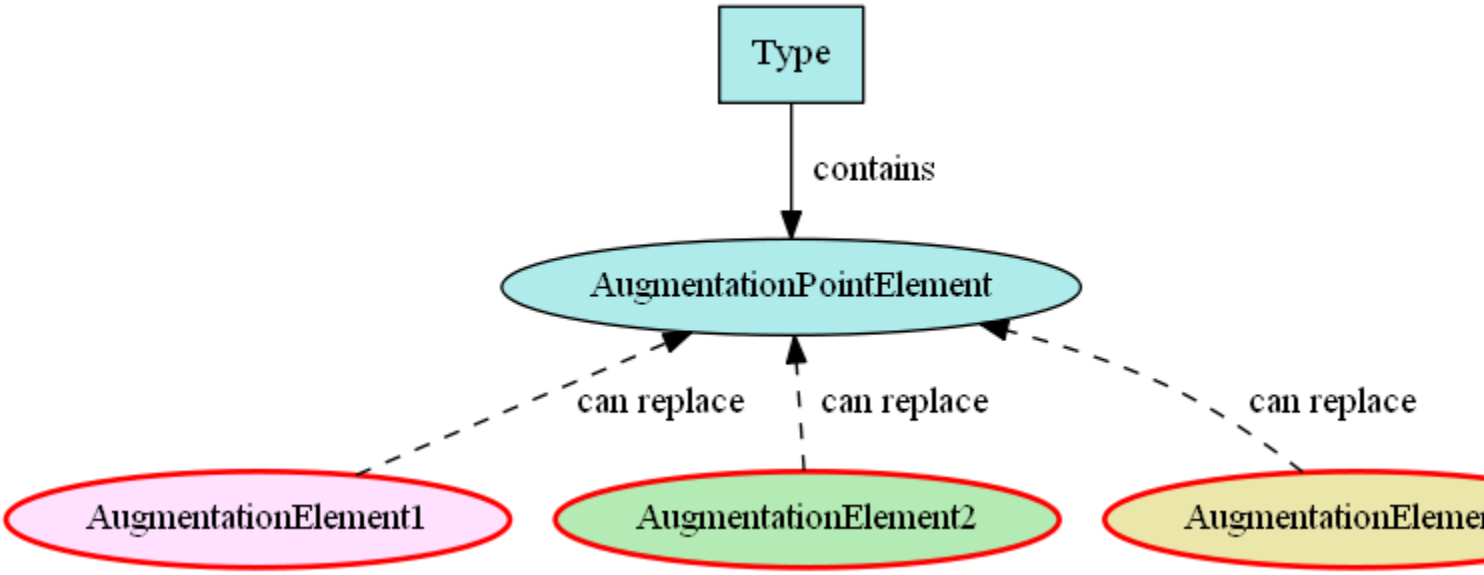
Rule	Applicability	Title
NDR 10-28	REF, EXT	Augmentation point element is only referenced by its base type
NDR 10-29	REF	Augmentation point element use is optional
NDR 10-30	REF	Augmentation point element use is unbounded
NDR 10-31	REF, EXT	Augmentation point element use must be last element in its base type
NDR 11-29	REF, EXT	Standard opening phrase for augmentation point element data definition

Augmentation Elements

Augmentation elements are the additional content to be used along with a type from a different namespace. This is done by making each augmentation element substitutable for the augmentation point element of the original type.

- Kinds
  - Containers
  - Direct substitutions
- XML Container
  - Instance example
  - Schema example
  - Schema template
- XML Direct Substitution
  - Instance example
  - Schema example
  - Schema template
- JSON
  - Instance examples

NIEM has two different ways of creating augmentation elements. The most common is by creating a container element and type that bundles related additions together into a single block, and making the container element substitutable for the augmentation point element. The other method is to make each of the additions individually substitutable for the augmentation point element, without being bundled together.



Kinds

Containers

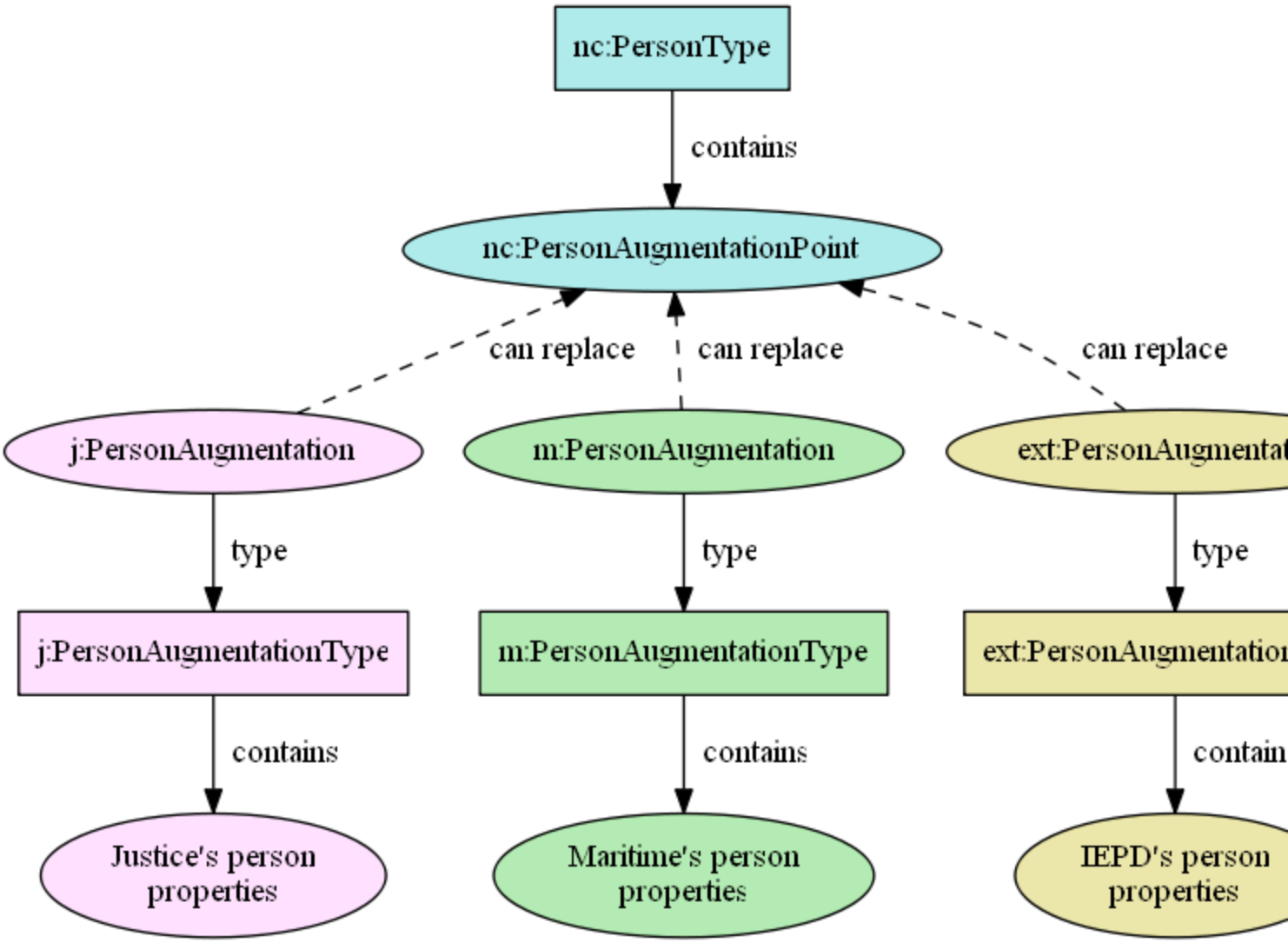
In NIEM releases, augmentations are created as containers that hold and group the additional content together. The name of the container element corresponds to the type being augmented, with the `"Type"` representation term being replaced by `"Augmentation"`. This augmentation element is made substitutable for the augmentation point element of the desired type.

Augmentation Container Element	Type	Substitutable for
intel:PersonAugmentation	intel:PersonAugmentationType	nc:PersonAugmentationPoint
j:PersonAugmentation	j:PersonAugmentationType	nc:PersonAugumentationPoint

The `intel:PersonAugmentationType` contains properties like `intel:AgencySubjectInterest` and `intel:PersonSystemIdentification`.

The `j:PersonAugmentationType` contains properties like `j:PersonAdultIndicator` and `j:PersonDrugAlcoholHistoryText`.

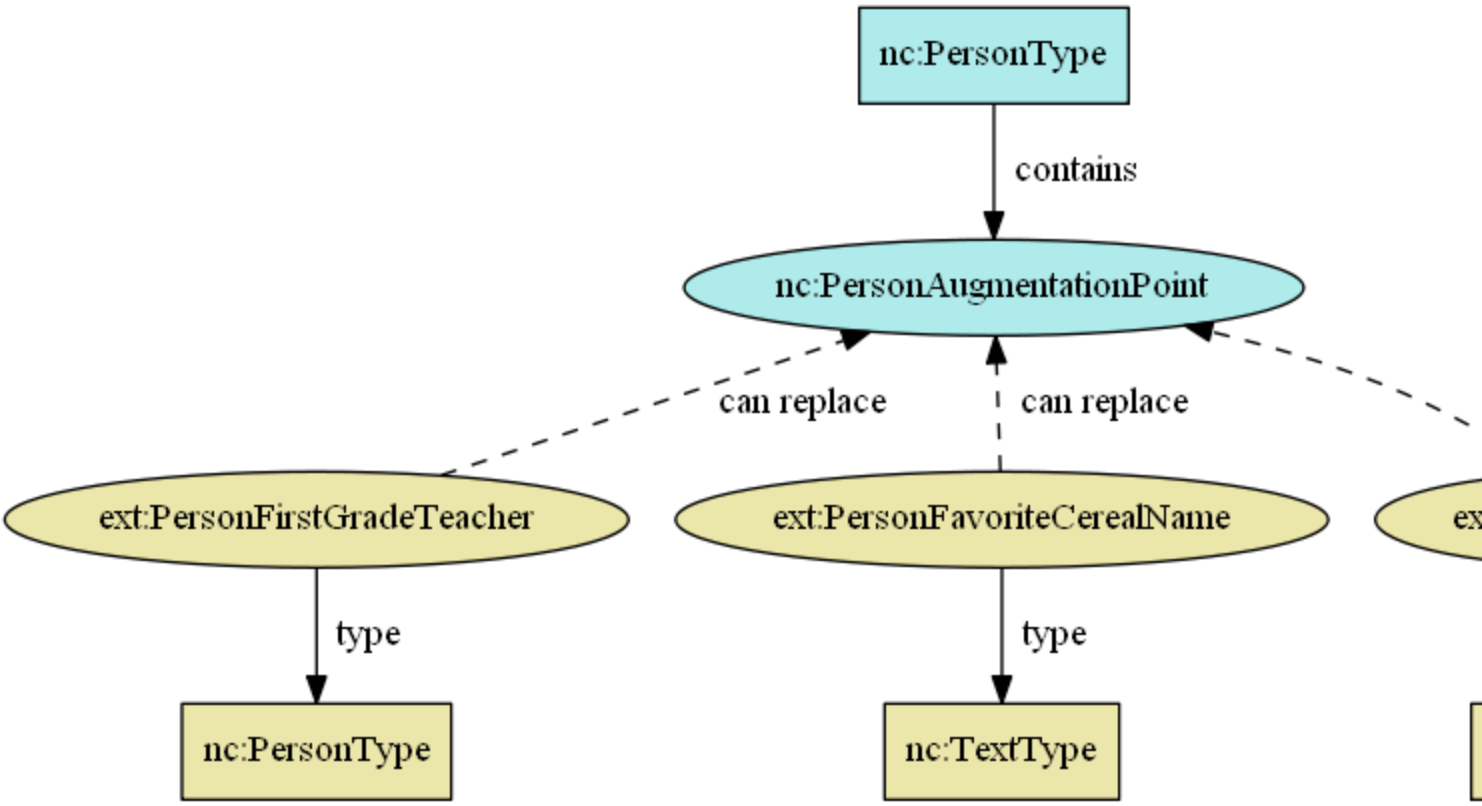
This diagram shows a type with an augmentation point element, substitutable by three augmentation container elements:



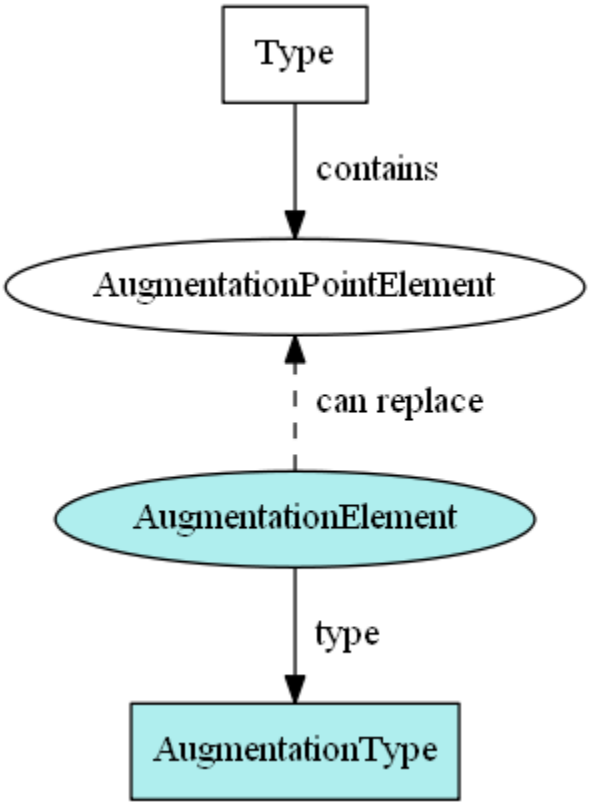
Direct substitutions

In an IEPD, rather than creating a container to bundle properties together into a single augmentation container element, each property itself may be made directly substitutable for an augmentation point element.

For example, local element `PersonFirstGradeTeacher` may be defined to be substitutable for `nc:PersonAugmentationPoint`.



XML Container



Instance example

The examples below show element nc:Person in an XML instance, with and without augmentations.

nc:Person without augmentations:

```
<nc:Person>
  <nc:PersonName>
    <nc:PersonFullName>John Smith</nc:PersonFullName>
  </nc:PersonName>
  <nc:PersonBirthDate>
    <nc:Date>1950-01-01</nc:Date>
  </nc:PersonBirthDate>
</nc:Person>
```

nc:Person with domain and local augmentations:

```
<nc:Person>

  <!-- Original nc:PersonType properties -->
  <nc:PersonName>
    <nc:PersonFullName>John Smith</nc:PersonFullName>
  </nc:PersonName>
  <nc:PersonBirthDate>
    <nc:Date>1950-01-01</nc:Date>
  </nc:PersonBirthDate>

  <!-- Justice augmentation properties -->
  <j:PersonAugmentation>
    <j:PersonAdultIndicator>true</j:PersonAdultIndicator>
    <j:PersonDEAIdentification>
      <nc:IdentificationID>1234567890</nc:IdentificationID>
    </j:PersonDEAIdentification>
  </j:PersonAugmentation>

  <!-- Maritime augmentation properties -->
  <m:PersonAugmentation>
    <m:PersonBiometricURI>https://biometric.example.com/23532356684</m:PersonB
  </m:PersonAugmentation>

  <!-- Local augmentation properties -->
  <ext:PersonLocalID>52d6632</ext:PersonLocalID>

</nc:Person>
```

Element substitution allows us to easily use properties from outside namespaces as part of `nc:Person`.

Schema example

This snippet, from a subset of the Maritime domain, shows how an augmentation container element and type are defined.

> The augmentation container element is designated as substitutable for `nc:PersonAugmentationPoint`, defined in the Core snippet above.

```
<xs:complexType name="PersonAugmentationType">
  <xs:annotation>
    <xs:documentation>A data type that supplements nc:PersonType.</xs:document
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:AugmentationType">
    <xs:sequence>
      <xs:element ref="m:PersonBiometricURI" minOccurs="0" maxOccurs="unbounde
    </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="PersonAugmentation" type="m:PersonAugmentationType" substitu
  <xs:annotation>
    <xs:documentation>Additional information about a person.</xs:documentatior
  </xs:annotation>
</xs:element>
```

Schema template

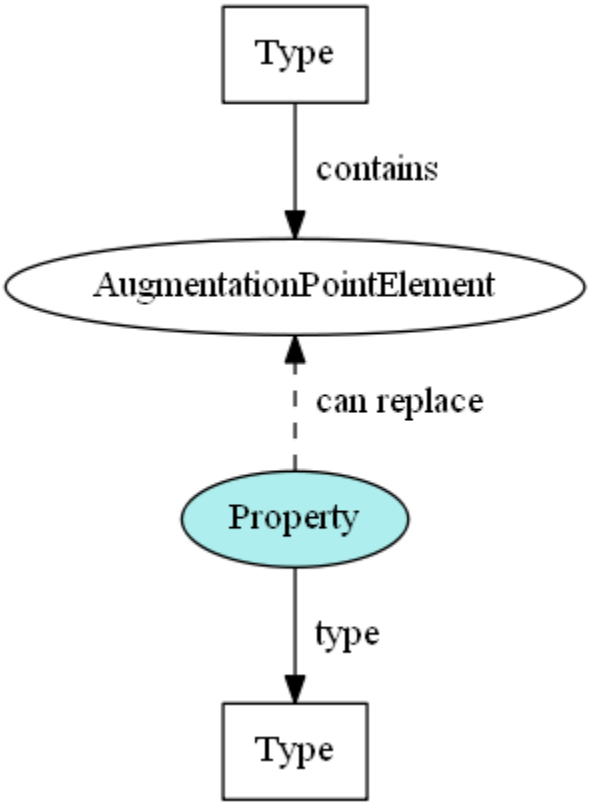
This template creates an augmentation container element and type.

- The augmentation type should contain references to the additional content.
- The augmentation element should be substitutable for an augmentation point.

```
<!-- Augmentation type, like j:PersonAugmentationType -->
<xs:complexType name="NAMEAugmentationType">
  <xs:annotation>
    <xs:documentation>A data type for additional information about NAME</xs:do
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:AugmentationType">
      <xs:sequence>
        <xs:element ref="ELEMENT1" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="ELEMENT2" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

```
<!-- Augmentation container element, like j:PersonAugmentation
(substitutable for nc:PersonAugmentationPoint) -->
<xs:element name="NAMEAugmentation" type="NAMEAugmentationType" substitutionGr
  <xs:annotation>
    <xs:documentation>Additional information about NAME</xs:documentation>
  </xs:annotation>
</xs:element>
```

XML Direct Substitution



The following example shows a local element `PersonFictionalCharacterIndicator` that is directly substitutable for nc:PersonAugmentationPoint:

Instance example

```
<nc:Person>
  <nc:PersonName>
    <nc:PersonFullName>John Smith</nc:PersonFullName>
  </nc:PersonName>
  <nc:PersonBirthDate>
    <nc:Date>1950-01-01</nc:Date>
  </nc:PersonBirthDate>

  <!-- The augmentation element, substituted directly for nc:PersonAugmentatio
  <ext:PersonFictionalCharacterIndicator>true</ext:PersonFictionalCharacterInd
```

```
</nc:Person>
```

Schema example

The only thing that distinguishes an augmentation element that is a direct substitution is its substitution group.

```
<xs:element name="PersonFictionalCharacterIndicator" type="niem-xs:boolean"
  substitutionGroup="nc:PersonAugmentationPoint">
    <xs:annotation>
      <xs:documentation>True if this person is a fictional character; false ot
    </xs:annotation>
</xs:element>
```

Schema template

```
<xs:element name="ELEMENT_NAME" type="ELEMENT_TYPE" substitutionGroup="NAMEAug
  <xs:annotation>
    <xs:documentation>ELEMENT_DEFINITION</xs:documentation>
  </xs:annotation>
</xs:element>
```

JSON

Augmentation container elements are important for XML schema validation, but have no role in the conceptual model, so they do not appear in the JSON representation. Additional content appears directly on their objects.

Instance examples

The examples below show element nc:Person in JSON, with and without augmentations.

nc:Person without augmentations:

```
{
  "nc:Person" : {
    "nc:PersonName" : {
      "nc:PersonFullName": "John Smith"
    },
    "nc:PersonBirthDate" : {
      "nc:Date": "1950-01-01"
    }
  }
}
```

nc:Person with domain and local augmentations.

```
{
```

```
  "nc:Person" : {

    "nc:PersonName" : {
      "nc:PersonFullName": "John Smith"
    },
    "nc:PersonBirthDate" : {
      "nc:Date": "1950-01-01"
    },

    "j:PersonAdultIndicator": true,
    "j:PersonDEAIdentification": {
      "nc:IdentificationID": "123456789"
    },

    "m:PersonBiometricURI": "https://biometric.example.com/23532356684",

    "ext:PersonLocalID": "52d6632"
  }
}
```

Local Terminology

Local terminology is a word, phrase, acronym, jargon, or other string of characters specially documented in a namespace because it is not defined in a standard dictionary. Literals or definitions are provided.

- Modeling guidance
  - Scope
  - Literal or definition
- XML
  - Instances
  - Schema example
  - Literal Template
  - Definition Template
- JSON
- NDR references

Because NIEM is designed to be widely reusable, it is important to make sure semantics are clear, even outside of the given subject matter area.

> Core defines “GSM” as a local term with literal “Global System for Mobile communication”.

Once defined at the namespace level, local terminology does not need to be redefined each time it is used.

Examples from Core:

Term	Literal	Definition
GSM	Global System for Mobile communication	
SSN	Social Security Number	
Alpha2		Two-letter
Alpha3		Three-letter

Modeling guidance

Scope

Local terminology declarations apply to the entire namespace in which they are defined.

This means that:

- A term that is used in multiple properties and/or types in a namespace is only defined once.
- Each namespace must define its own terminology - terminology from Core or other domains cannot be imported or reused.

Literal or definition

Each local term must have a literal translation or a definition.

XML

Instances

Local terminology does not appear in instances. They are defined in schemas to provide additional context for the properties and types defined there.

Schema example

The snippet below is an extract from Core 4.0:

```
<xs:schema>
  <xs:annotation>
    <xs:documentation>NIEM Core.</xs:documentation>
    <xs:appinfo>
      <appinfo:LocalTerm term="SSN" literal="Social Security Number"/>
      <appinfo:LocalTerm term="ISO" literal="International Organization for St
      <appinfo:LocalTerm term="Alpha3" definition="Three-letter"/>
      <appinfo:LocalTerm term="Alpha2" definition="Two-letter"/>
    </xs:appinfo>
  </xs:annotation>
</xs:schema>
```

This example also includes the namespace definition. A namespace's definition and local terminology are defined in the same schema-level `xs:annotation` block.

Literal Template

This template shows how to define a term with its literal translation:

```
<xs:appinfo>
  <appinfo:LocalTerm term="TERM" literal="LITERAL"/>
</xs:appinfo>
```

Definition Template

This template shows how to define a term with its definition:

```
<xs:appinfo>
  <appinfo:LocalTerm term="TERM" definition="DEFINITION"/>
</xs:appinfo>
```

JSON

JSON guidance is not yet available for local terminology.

NDR references

NDR 5.0

- NDR Section 10.9.2 - Local terminology

Rule	Applicability	Title
NDR 10-52	REF, EXT	Local term declaration is local to its schema document
NDR 10-53	REF, EXT	Local terminology interpretation
NDR 10-76	REF, EXT	appinfo:LocalTerm annotates schema
NDR 10-77	REF, EXT	appinfo:LocalTerm has literal or definition

NDR 4.0

- NDR Section 10.9.2 - Local terminology

Rule	Applicability	Title
NDR 10-51	REF, EXT	Local term declaration is local to its schema document
NDR 10-52	REF, EXT	Local terminology interpretation
NDR 10-75	REF, EXT	appinfo:LocalTerm annotates schema
NDR 10-76	REF, EXT	appinfo:LocalTerm has literal or definition

Metadata

Metadata is data about data. It might describe such things as who provided the content, and when it was last updated.

- Overview
  - Using metadata
  - Metadata is a set
  - Where to reference metadata
- Modeling guidance
  - Metadata Elements
    - Name
    - Type
  - Applicability
  - Metadata Types
    - Name
    - Parent type
    - No augmentation points
- XML
  - Instance example
  - Schema example
  - Schema template
- JSON
- NDR references

Overview

Because this is data that could potentially be provided about any and every property in the model (e.g., when the person name was last updated, when the address was last updated, when the vehicle tag was last updated, and so on), metadata should be defined separately from other types. This avoids duplication and clutter across the model.

Almost all properties in NIEM (those with complex types) have the ability to reference one or more metadata properties.

> Core defines metadata element `nc:Metadata`, which contains properties like `nc:ReportedDate` and `nc:SourceText`. Any element, like `nc:Person`, `j:PersonEmploymentIndicator`, or an element defined in an IEPD, may use this to provide metadata.

Using metadata

Metadata should not be embedded in the primary content. All elements in NIEM inherit attribute `structures:metadata`. This attribute can carry a list of ID references, meaning each element can link to one or more metadata elements, as applicable.

Elements and metadata have a many-to-many relationship.

1. One element can link to multiple metadata elements. This means that metadata elements from Core, one or more domains, and/or an extension namespace may all be used, if needed.
2. One metadata element can be referenced by many elements. This means that if there are multiple elements that share the same metadata in an instance (for example, if they all come from the same database record that will have the same update date), then they can all link to the same metadata value(s) rather than duplicating the information.

Metadata is a set

A metadata element in NIEM has a metadata type that bundles up one or more informational properties, such as `nc:EffectiveDate` and `nc:LastUpdatedDate`. This simplifies the usage - an element references a metadata bundle once, rather than each of its contents individually.

NIEM's rules and guidance about metadata applies to those elements and types that act as the bundles, like element `nc:Metadata` and type `nc:MetadataType`. The informational properties, like `nc:EffectiveDate`, have no special rules and are created and treated like any other property in the model.

Where to reference metadata

Metadata can be referenced from any element. It may not be necessary though to repeat the same references on each element in an instance. Metadata on a higher level element, like `nc:Person`, may be considered to carry through to children elements, such as `nc:PersonBirthDate`, unless specifically overridden.

Modeling guidance

Metadata Elements

Name

A metadata element should end with the representation term `Metadata`.

For general metadata that may be applied to anything, the name `Metadata` is sufficient. For more specific metadata, a more descriptive name should be used, like `PersonMetadata`.

Type

A metadata element must have a metadata type.

Applicability

A metadata element may be able to be used by any NIEM element or type, or it may be constrained to only certain ones.

If a metadata element is constrained to a given type, then any element of that type or of a type that extends from that type may reference it. For example, a metadata element constrained to type

`structures:AssociationType` may be used by any association element.

Use the attributes `appinfo:appliesToElements` and `appinfo:appliesToTypes` to limit the metadata element to one or more elements and/or types.

Metadata Types

Name

A metadata type should end with the representation term `MetadataType`.

For general metadata that may be applied to anything, the name `MetadataType` is sufficient. For more specific metadata, a more descriptive name should be used, like `PersonMetadataType`.

Parent type

A metadata type must extend `structures:MetadataType`.

There is no need to extend other metadata types in the model. Elements can link to multiple metadata elements.

No augmentation points

Metadata types should not contain augmentation points. Other namespaces can define their own metadata elements and types and create references to them in NIEM release elements.

XML

Instance example

An element links to metadata by referencing each applicable metadata element's ID (`structures:id`) in inherited attribute `structures:metadata`.

In the example below, a person is defined with two sub-properties: nc:PersonBirthDate and nc:PersonName. Each of these elements reference metadata IDs via their `structures:metadata` attribute.

```
<nc:Person>
  <nc:PersonBirthDate structures:metadata="m1">
    <nc:Date>1945-12-01</nc:Date>
  </nc:PersonBirthDate>
  <nc:PersonName structures:metadata="m1 m2">
    <nc:PersonFullName>John Doe</nc:PersonFullName>
  </nc:PersonName>
</nc:Person>

<nc:Metadata structures:id="m1">
```

```
  <nc:SourceText>Adam Barber</nc:SourceText>
</nc:Metadata>

<nc:Metadata structures:id="m2">
  <nc:ReportedDate>
    <nc:Date>2005-04-26</nc:Date>
  </nc:ReportedDate>
</nc:Metadata>
```

Schema example

This example shows a subset of metadata type `nc:MetadataType` and metadata element `nc:Metadata` in NIEM Core 4.0:

- The metadata type extends `structures:MetadataType`.
- The metadata element defines an attribute to specify what types may reference this element. Specifying `structures:ObjectType` and `structures:AssociationType` enables applicability from any element with a type derived from either of these.

```
<xs:schema>
  <xs:complexType name="MetadataType">
    <xs:annotation>
      <xs:documentation>
        A data type for information that further qualifies primary data; data
      </xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="structures:MetadataType">
        <xs:sequence>
          <xs:element ref="nc:CaveatText" minOccurs="0" maxOccurs="unbounded"/
          <xs:element ref="nc:DistributionText" minOccurs="0" maxOccurs="unbou
          <xs:element ref="nc:EffectiveDate" minOccurs="0" maxOccurs="unbounde
          <xs:element ref="nc:ExpirationDate" minOccurs="0" maxOccurs="unbound
          <xs:element ref="nc:LastUpdatedDate" minOccurs="0" maxOccurs="unbour
          <xs:element ref="nc:LastVerifiedDate" minOccurs="0" maxOccurs="unbou
          <xs:element ref="nc:ReportedDate" minOccurs="0" maxOccurs="unbounded
          <xs:element ref="nc:Comment" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>

<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>
```

Schema template

```
<xs:schema>
  <xs:complexType name="NAMEMetadataType">
    <xs:annotation>
      <xs:documentation>DEFINITION</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="structures:MetadataType">
        <xs:sequence>
          <xs:element ref="INFORMATIONAL_ELEMENT" minOccurs="0" maxOccurs="unb
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>

<xs:element name="NAMEMetadata" type="NAMEMetadataType" nillable="true"
    appinfo:appliesToTypes="TYPE1 TYPE2 ..."
    appinfo:appliesToElements="ELEMENT1 ELEMENT2 ...">
  <xs:annotation>
    <xs:documentation>DEFINITION</xs:documentation>
  </xs:annotation>
</xs:element>
```

JSON

Guidance has not yet been provided on metadata in JSON.

NDR references

NDR 5.0

- NDR Section 10.5 - Metadata

Rule	Applicability	Title
NDR 10-38	REF, EXT	Metadata type has data about data
NDR 10-39	REF, EXT	Metadata types are derived from structures:MetadataType
NDR 10-40	REF, EXT	Metadata element declaration type is a metadata type
NDR	REF, EXT,	Metadata element has applicable elements

Rule	Applicability	Title
10-41	SET	
NDR 11-33	REF, EXT	Standard opening phrase for metadata element data definition
NDR 11-45	REF, EXT	Standard opening phrase for metadata type data definition
NDR 12-11	INS	Metadata applies to referring entity
NDR 12-12	INS	Referent of structures:relationshipMetadata annotates relationship
NDR 12-13	INS	Values of structures:metadata refer to values of structures:id
NDR 12-14	INS	Values of structures:relationshipMetadata refer to values of structures:id
NDR 12-15	INS	structures:metadata and structures:relationshipMetadata refer to metadata elements
NDR 12-16	INS	Attribute structures:metadata references metadata element
NDR 12-17	INS	Attribute structures:relationshipMetadata references metadata element
NDR 12-18	INS	Metadata is applicable to element

NDR 4.0

- NDR Section 10.5 - Metadata

Rule	Applicability	Title
NDR 10-38	REF, EXT	Metadata type has data about data
NDR 10-39	REF, EXT	Metadata types are derived from structures:MetadataType
NDR 10-40	REF, EXT	Metadata element declaration type is a metadata type
NDR 10-41	REF, EXT, SET	Metadata element has applicable elements
NDR 11-31	REF, EXT	Standard opening phrase for metadata element data definition
NDR 11-43	REF, EXT	Standard opening phrase for metadata type data definition
NDR 12-10	INS	Metadata applies to referring entity
NDR 12-11	INS	Referent of structures:relationshipMetadata annotates relationship
NDR	INS	Values of structures:metadata refer to values of structures:id

Rule	Applicability	Title
12-12		
NDR 12-13	INS	Values of structures:relationshipMetadata refer to values of structures:id
NDR 12-14	INS	structures:metadata and structures:relationshipMetadata refer to metadata elements
NDR 12-15	INS	Attribute structures:metadata references metadata element
NDR 12-16	INS	Attribute structures:relationshipMetadata references metadata element
NDR 12-17	INS	Metadata is applicable to element

References

References are used in instances or messages to point to content defined elsewhere.

- Overview
  - Linked Data
- Modeling guidance
  - Do not recreate
  - No type validation
- XML
  - Mechanism
    - structures:id
    - structures:ref
    - structures:uri
  - Local examples
    - structures:ref
    - structures:uri
  - External examples
    - Absolute URI
    - With xml:base
- JSON
  - Local references
  - External references
    - Absolute IRIs
    - With base
- NDR references

Overview

A common reason to use references is to avoid repeating data. In addition to avoiding duplication, it clearly links multiple occurrences to the same object. Just because two properties have the same content (e.g., the same name), they are not necessarily the same.

In order to use references, a unique ID must first be assigned to where the content is defined. That ID can then be references elsewhere. Almost all properties in NIEM (those with complex types) carry attributes that let you define an ID or a reference.

Linked Data

In addition to referencing content defined in the same instance document, it is also possible to reference content defined outside of the instance document. Resources defined elsewhere may be referenced by a NIEM property via a uniform resource identifier (URI).

This capability was introduced in NIEM 4.0 in support of Linked Data and the semantic web.

Modeling guidance

References are handled differently in the NIEM XML and NIEM JSON representations. Please see these sections for more guidance.

Do not recreate

Note that in either representation, NIEM already uses or provides the infrastructure needed to create and reference identifiers. This functionality should be used as is for consistency; custom id / referencing mechanisms should not be added.

No type validation

References in XML and JSON are not type-checked via standard validation. This means that it will be possible to provide references to incompatible content, such as an identifier for a vehicle element when an identifier for a person element was expected.

This lack of validation does allow for greater flexibility in defining references. It is not necessary for one element to link to another element of the same name. Though not enforcement by validation, an element of a given type may link to…

- a different element with the same type
- an element with a type derived from or a parent of the given type
- an element that represents a role or base object of the given type

XML

Mechanism

NIEM defines three attributes in utility schema structures.xsd in support of referencing:

- structures:id
- structures:ref
- structures:uri

These attributes are added to the following NIEM constructs:

- structures:ObjectType
- structures:AssociationType
- structures:MetadataType
- structures:AugmentationType
- structures:SimpleObjectAttributeGroup

Because all NIEM elements have a type that either extends, inherits from, or reuses one of these constructs, the ability to create and reference IDs are automatically available across the model, without any additional setup required.

structures:id

Local identifiers may be created in an instance through the use of attribute `structures:id`.

This attribute has type `xs:ID`, which requires a unique value within an instance.

structures:ref

Local elements may be referenced in an instance through the use of attribute `structures:ref`.

This attribute has type `xs:IDREF`, which requires that its value appear elsewhere in an instance as the value of an attribute of type `xs:ID`. For NIEM, this means that a `structures:ref` attribute value must correspond to a unique `structures:id` value.

structures:uri

Local elements or external resources may be referenced in an instance through the use of attribute `structures:uri`, and optionally attribute `xml:base`.

The value of a `structures:uri` attribute may refer to the value of another `structures:uri` or `structures:id` attribute, or it may be unique:

- To refer to a local identifier, the value should be a URI fragment, prefixed with "#".
- To refer to an external resource, the value should be an absolute URI, or a URI fragment used in addition to a URI base provided by attribute `xml:base`.

Local examples

structures:ref

This example shows two elements that reference a third, more generic element (nc:Person):

```
<j:Arrest>
  <j:ArrestInvolvedWeapon>
    <nc:WeaponUser structures:ref="bart" xsi:nil="true"/>
  </j:ArrestInvolvedWeapon>

  <j:ArrestSubject>
    <nc:RoleOfPerson structures:ref="bart" xsi:nil="true"/>
  </j:ArrestSubject>
</j:Arrest>

<nc:Person structures:id="bart">
  <nc:PersonName>
    <nc:PersonGivenName>Bart</nc:PersonGivenName>
  </nc:PersonName>
</nc:Person>
```

This example shows one element (nc:RoleOfPerson) that references another element (nc:WeaponUser):

```
<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>
```

structures:uri

This example shows how `structures:uri` may be used to create a local reference:

```
<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:uri="#bart" xsi:nil="true"/>
  </j:ArrestSubject>
</j:Arrest>
```

Because the uri attribute is only a fragment and attribute `xml:base` is not provided, the uri refers to an identifier in the current document.

External examples

The following two examples show how structures:uri may be used alone or in combination with attribute `xml:base` to reference an externally-defined resource.

Absolute URI

```
<j:Arrest>
  <j:ArrestSubject>
    <nc:RoleOfPerson
      structures:uri="https://state.example/98723987/results.xml#bart"
      xsi:nil="true"/>
```

```
  </j:ArrestSubject>
</j:Arrest>
```

With xml:base

```
<j:Arrest xml:base="https://state.example/98723987/results.xml">
  <j:ArrestSubject>
    <nc:RoleOfPerson structures:uri="#bart" xsi:nil="true"/>
  </j:ArrestSubject>
</j:Arrest>
```

JSON

Local references

In a JSON instance, both identifiers and references are specified with the "@id" property.

```
{
  "j:Arrest": {
    "j:ArrestInvolvedWeapon": {
      "nc:WeaponUser": {
        "@id": "bart"
      }
    },
    "j:ArrestSubject": {
      "nc:RoleOfPerson": {
        "@id": "bart"
      }
    }
  },
  "nc:Person": {
    "@id": "bart",
    "nc:PersonName": {
      "nc:PersonGivenName": "Bart"
    }
  }
}
```

External references

Absolute IRIs

```
{
  "j:Arrest": {
    "j:ArrestInvolvedWeapon": {
      "nc:WeaponUser": {
        "@id": "https://state.example/98723987/results.xml#bart"
      }
```

```
    },
    "j:ArrestSubject": {
      "nc:RoleOfPerson": {
        "@id": "https://state.example/98723987/results.xml#bart"
      }
    }
  }
}
```

With base

Relative IRIs (international resource identifiers) will be resolved against the document base. This base may be set explicitly as part of the document context using the `@base` keyword:

```
{
  "@context": {
    "@base": "https://state.example/98723987/results.xml"
  },
  "j:Arrest": {
    "j:ArrestInvolvedWeapon": {
      "nc:WeaponUser": {
        "@id": "bart"
      }
    },
    "j:ArrestSubject": {
      "nc:RoleOfPerson": {
        "@id": "bart"
      }
    }
  }
}
```

NDR references

NDR 5.0

- NDR Section 12.2 - Identifiers and references

Rule	Applicability	Title
NDR 12-3	INS	Element has only one resource identifying attribute
NDR 12-4	INS	Attribute structures:ref must reference structures:id
NDR 12-5	INS	Linked elements have same validation root
NDR 12-6	INS	Attribute structures:ref references element of correct type
NDR 12-7	INS	structures:uri denotes resource identifier
NDR 12-8	INS	structures:id and structures:ref denote resource identifier
NDR 12-9	INS	Nested elements and references have the same meaning.

NDR 4.0

- NDR Section 12.2 - Identifiers and references

Rule	Applicability	Title
NDR 12-3	INS	Element has only one resource identifying attribute
NDR 12-4	INS	Attribute structures:ref must reference structures:id
NDR 12-5	INS	Linked elements have same validation root
NDR 12-6	INS	Attribute structures:ref references element of correct type
NDR 12-7	INS	structures:uri denotes resource identifier
NDR 12-8	INS	structures:id and structures:id denote resource identifier
NDR 12-9	INS	Nested elements and references have the same meaning.

Representation Pattern

In NIEM, multiple representations of a concept are typically handled via abstract elements and substitution groups. This allows for replacement of the abstract element in an instance with one or more of the substitutions.

- Overview
  - Abstract vs Representation
- Country codes example
  - Representation Pattern
    - Before
    - After
    - Comparison
  - Diagram
  - XML Examples
    - Schema
    - Instance example
  - Other representations
- Modeling guidance
  - Type
  - Representation Element
    - Abstract
    - Name
- XML
  - Instance example
  - Schema example
  - Schema template
- JSON
  - Instance example
  - Schema example
  - Schema template
- NDR references

Overview

> Abstract element nc:PersonHairColorAbstract has multiple substitutions, allowing for free-text or code value representations.

In cases where the concept not only has multiple representations but also additional properties describing the concept, a new type is needed to bundle the representations and additional properties together. This is the representation pattern.

- Type nc:DateType contains…
  - An abstract date representation element
  - Additional date-related properties

The major difference between the two examples is that there are other date-related properties in addition

to the multiple date representations; hair color has multiple representations only.

Abstract vs Representation

The table below shows an example outlining the difference between regular abstract elements + substitutions versus the representation pattern.

- Multiple representations are handled the same in both cases - an abstract element with multiple substitutions.
- The representation pattern adds additional content and a type to bundle it with the abstract element.

Pattern	Abstract Element + Substitutions	Representation Pattern
Type	n/a	nc:DateType nc:DateRepresentation
--Multiple representations	nc:PersonHairColorAbstract • substitutions: • nc:PersonHairColorText • j:PersonHairColorCode	• substitutions: • nc:Date • nc:DateTime • nc:ZuluDateTime • ...
--Additional Content	n/a	• nc:DateAccuracyAbstract • nc:DateMarginOfErrorDuration

Just because an abstract element is contained in a type does not mean it follows the representation pattern. Element nc:PersonHairColorAbstract is used in type nc:PersonType, but hair color is simply a property of a person, not a full representation of a person.

Country codes example

Country codes are one example of the representation pattern in NIEM.

Two of the authoritative sources for country codes in the NIEM 4.0 release are **GENC** (Geopolitical Names, Entities, and Codes) and **ISO 3166-1**. Both of these standards have 2-character, 3-character, and numeric representations. In addition to these, NIEM also includes a text representation.

In NIEM 4.0, country codes were refactored in order to use the representation pattern. This provided multiple benefits:

- It simplified the creation of new country code elements.
- It reduced the number of existing elements with the same semantic meaning.
- It improved consistency, by making the same set of substitutions easily reusable.

See Country Code Representations for information about the different representation formats used with country codes, including CSV files for GENC codes.

Representation Pattern

Because NIEM (1) has multiple representations for country codes and (2) country codes are used in many different contexts throughout the model, they were a good candidate for the representation pattern.

Before

In NIEM 3.2, before the representation pattern was used, a new country code element was created for each usage of a code set. These elements include:

NIEM 3.2 Element	Type
nc:DocumentCountryISO3166Alpha2Code	iso_3166:CountryAlpha2CodeType
nc:LocationCountryISO3166Alpha2Code	iso_3166:CountryAlpha2CodeType
nc:LocationCountryGENCCode	genc:CountryAlpha3CodeType
nc:MilitaryCountryGENCCode	genc:CountryAlpha3CodeType
nc:PersonCitizenshipISO3166Alpha2Code	iso_3166:CountryAlpha2CodeType
nc:PersonNationalityISO3166Alpha2Code	iso_3166:CountryAlpha2CodeType
biom:ForensicSourceCountryISO3166Alpha2Code	iso_3166:CountryAlpha2CodeType
biom:RecordingSourceCountryISO3166Alpha2Code	iso_3166:CountryAlpha2CodeType
mo:FacilityAllegianceCountryCode	genc:CountryAlpha3CodeType

After

In order to simplify and enable reuse, the representation pattern was used in NIEM 4.0 and type `nc:CountryType` was created. This type contains abstract element `nc:CountryRepresentation`, which allows for substitution by GENC and ISO 3166 code elements, as well as other NIEM and local elements.

Using the representation pattern, NIEM 4.0 country code elements include:

NIEM 4.0 Element	Type
nc:DocumentCountry	nc:CountryType
nc:LocationCountry	nc:CountryType
nc:MilitaryCountry	nc:CountryType
biom:ForensicSourceCountry	nc:CountryType
biom:RecordingSourceCountry	nc:CountryType
mo:FacilityAllegianceCountry	nc:CountryType

This approach adds greater consistency with less duplication. Any element of type `nc:CountryType` may use one or more of the different country code representations. And each semantic concept only has to be defined once.

Comparison

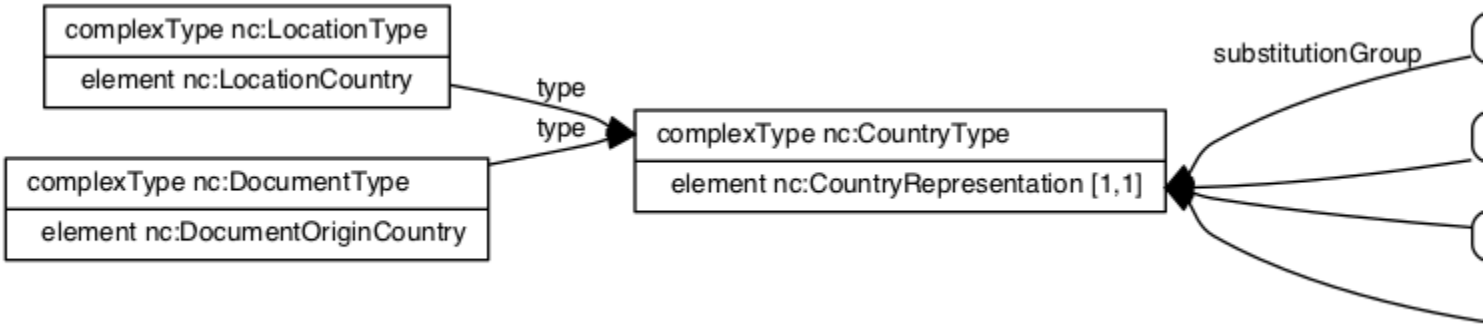
In NIEM 4.0, the addition of `nc:CountryType` made it possible to reuse multiple representations, rather than requiring a new element to be created for each usage.

Release	Location Country Elements	Types
• NIEM 3.2	• nc:LocationCountryISO3166Alpha2Code • nc:LocationCountryGENCCode	• iso_3166:CountryAlpha2CodeType • genc:CountryAlpha3CodeType
• NIEM 4.0	• nc:LocationCountry	• nc:CountryType

Diagram

The following diagram shows type `nc:CountryType`, which carries representations of a country across NIEM exchanges.

- type `nc:CountryType` is used by `nc:LocationType` and `nc:DocumentType`
- `nc:CountryType` has an element `nc:CountryRepresentation`, which may be replaced in an instance by a number of ways to identify a country.



XML Examples

Schema

The example below shows the declaration of type `nc:CountryType` and one element that uses this type:

```
<xs:complexType name="CountryType">
  <xs:annotation>
    <xs:documentation>A data type for a country, territory, dependency, or oth
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      <xs:sequence>
        <xs:element ref="nc:CountryRepresentation" minOccurs="0" maxOccurs="un
        <xs:element ref="nc:CountryAugmentationPoint" minOccurs="0" maxOccurs=
      </xs:sequence>
    </xs:extension>
```

```
  </xs:complexContent>
</xs:complexType>

<xs:element name="LocationCountry" type="nc:CountryType" nillable="true">
  <xs:annotation>
    <xs:documentation>A country, territory, dependency, or other such geopolit
  </xs:annotation>
</xs:element>
```

Instance example

The following shows three different examples of element `nc:LocationCountry`, each using a different representation provided by `nc:CountryType`:

```
<nc:LocationCountry>
  <nc:LocationCountryISO3166Alpha2Code>US</nc:LocationCountryISO3166Alpha2Code
</nc:LocationCountry>

<nc:LocationCountry>
  <nc:CountryCode>USA</nc:CountryCode>
</nc:LocationCountry>

<nc:LocationCountry>
  <nc:LocationCountryName>United States</nc:LocationCountryName>
</nc:LocationCountry>
```

Other representations

In addition to the GENC and ISO 3166 representations, other representations may be used with NIEM 4, either by additional substitutions for `nc:CountryRepresentation`, or by providing additional code list URIs with `nc:CodeType` (see Code Lists for more).

An IEPD (information exchange package definition), which specifies a message, may include one or more of these representations to convey countries. Which representations are used is determined by the community of interest developing the message specification.

Modeling guidance

Type

A representation pattern is modeled by a type that contains an abstract representation element and one or more additional properties.

Representation Element

Abstract

The representation element must be abstract. This allows for substitution by the alternate representations.

Name

The name of the representation element must end with “Representation”.

XML

Instance example

The example below shows an element with type nc:DateType.

- Element “nc:Date” substitutes for abstract element nc:DateRepresentation.
- The date also includes a margin of error duration. In this cases, the date is plus or minus a period of 3 months.

```
<nc:ActivityDate>
  <nc:Date>2017-06-01</nc:Date>
  <nc:DateMarginOfErrorDuration>P3M</nc:DateMarginOfErrorDuration>
</nc:ActivityDate>
```

Schema example

The example below shows..

- The declaration of type nc:DateType, which follows the representation pattern.
- The declaration of the representation element nc:DateRepresentation (abstract) and several substitutions.

```
<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="unbou
        <xs:element ref="nc:DateAccuracyAbstract" minOccurs="0" maxOccurs="1"/
        <xs:element ref="nc:DateMarginOfErrorDuration" minOccurs="0" maxOccurs
        <xs:element ref="nc:DateAugmentationPoint" minOccurs="0" maxOccurs="un
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
```

```
</xs:complexType>

<xs:element name="DateRepresentation" abstract="true">
  <xs:annotation>
    <xs:documentation>A data concept for a representation of a date.</xs:docum
  </xs:annotation>
</xs:element>

<xs:element name="Date" type="niem-xs:date" substitutionGroup="nc:DateRepresen
  <xs:annotation>
    <xs:documentation>A full date.</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="DateTime" type="niem-xs:dateTime" substitutionGroup="nc:Date
  <xs:annotation>
    <xs:documentation>A full date and time.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Schema template

```
<xs:complexType name="NAMEType">
  <xs:annotation>
    <xs:documentation>A data type for ...</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      <xs:sequence>
        <xs:element ref="nc:NAMERepresentation" minOccurs="0" maxOccurs="unbou
        <xs:element ref="..." minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="nc:NAMEAugmentationPoint" minOccurs="0" maxOccurs="un
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="NAMERepresentation" abstract="true">
  <xs:annotation>
    <xs:documentation>A data concept for a representation of a ...</xs:documen
  </xs:annotation>
</xs:element>

<xs:element name="REPRESENTATION-1" type="TYPE" substitutionGroup="nc:NAMERepr
  <xs:annotation>
    <xs:documentation>DEFINITION</xs:documentation>
  </xs:annotation>
</xs:element>

<xs:element name="REPRESENTATION-2" type="TYPE" substitutionGroup="nc:NAMERepr
```

```
  <xs:annotation>
    <xs:documentation>DEFINITION</xs:documentation>
  </xs:annotation>
</xs:element>
```

JSON

Instance example

The example below shows an element with type nc:DateType.

- Property “nc:Date” is used as the representation of a date.
- The date also includes a margin of error duration. In this cases, the date is plus or minus a period of 3 months.

```
{
  "nc:ActivityDate": {
    "nc:Date": "2017-06-01",
    "nc:DateMarginOfErrorDuration": "P3M"
  }
}
```

Schema example

The example below shows..

- The declaration of type nc:DateType, which follows the representation pattern.
- The declaration of several representation element.

Because JSON Schema does not support substitution groups, representation properties should be added directly to the type. In XML Schema, the representations would be created as substitutable for an abstract element instead.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "properties": {
    "nc:Date": {
      "description": "A full date.",
      "oneOf": [
        {
          "$ref": "#/definitions/niem-xs:date"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/niem-xs:date"
          }
```

```
        }
      ]
    },

    "nc:DateTime": {
      "description": "A full date and time.",
      "oneOf": [
        {
          "$ref": "#/definitions/niem-xs:dateTime"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/niem-xs:dateTime"
          }
        }
      ]
    }
  }

  "definitions": {
    "nc:DateType": {
      "description": "A data type for a calendar date.",
      "allOf": [
        {
          "$ref": "#/definitions/_base"
        },
        {
          "type": "object",
          "properties": {
            "nc:Date": {
              "$ref": "#/properties/nc:Date"
            },
            "nc:DateTime": {
              "$ref": "#/properties/nc:DateTime"
            },
            "nc:ZuluDateTime": {
              "$ref": "#/properties/nc:ZuluDateTime"
            },
            "nc:DateMarginOfErrorDuration": {
              "$ref": "#/properties/nc:DateMarginOfErrorDuration"
            }
          }
        }
      ]
    }
  }
}
```

Schema template

The template belows shows the declaration of a type following the representation pattern, plus two representation properties.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "properties": {
    "REPRESENTATION-1": {
      "description": "DEFINITION",
      "oneOf": [
        {
          "$ref": "#/definitions/TYPE"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/TYPE"
          }
        }
      ]
    },

    "REPRESENTATION-2": {
      "description": "DEFINITION",
      "oneOf": [
        {
          "$ref": "#/definitions/TYPE"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/TYPE"
          }
        }
      ]
    }
  }

  "definitions": {
    "REPRESENTATION-TYPE": {
      "description": "A data type for ...",
      "allOf": [
        {
          "$ref": "#/definitions/_base"
        },
        {
          "type": "object",
          "properties": {
```

```
            "REPRESENTATION-1": {
              "$ref": "#/properties/REPRESENTATION-1"
            },
            "REPRESENTATION-2": {
              "$ref": "#/properties/REPRESENTATION-2"
            },
            "ADDITIONAL-CONTENT": {
              "$ref": "#/properties/ADDITIONAL-CONTENT"
            }
          }
        }
      ]
    }
  }
}
```

NDR references

NDR 5.0

- NDR Section 10.7 - The “Representation” pattern

Rule	Applicability	Title
NDR 10-42	REF, EXT	Name of element that ends in Representation is abstract
NDR 10-43	REF, EXT	A substitution for a representation element declaration is a value for a type

NDR 4.0

- NDR Section 10.7 - The Representation pattern

Rule	Applicability	Title
NDR 10-42	REF, EXT	Name of element that ends in Representation is abstract
NDR 10-43	REF, EXT	A substitution for a representation element declaration is a value for a type

Roles

NIEM differentiates between an object and a role of an object. Roles are used to represent a non-exclusive function or part played by an object. An object may have one or more roles. A role may a function of one or more objects.

- Overview
  - Extension vs Roles
  - Simple roles
  - Custom role types
  - Examples
- Modeling guidance
  - Role types
    - Parent type
    - RoleOf element usage
  - RoleOf elements
    - Reuse
    - Name
    - Type
    - Substitution Group
- XML
  - Role types
    - Inline instance example
    - Reference instance example
    - Schema example
    - Schema template
  - Role-of element
    - Schema example
    - Schema template
- JSON
  - Role types
    - Inline instance example
    - Reference instance example
    - Schema example
    - Schema template
  - Role-of element
    - Schema example
    - Schema template
- NDR references

Overview

- Element nc:Weapon is a role of an item
- Element j:Evidence is a role of…
  - an item
  - biometric data
  - a binary file, like an image, video, or audio

Things like cars and chairs are items in NIEM. In certain situations, however, they may be used as a weapon and/or function as evidence in a trial or hearing.

Extension vs Roles

NIEM uses type extension for exclusive specialization. In NIEM Core, an aircraft, a vehicle, and a vessel are all special kinds of a conveyance. Since the same conveyance doesn't typically act like an aircraft in one situation and a vessel in another, NIEM uses extension to model these types. Types nc:AircraftType, nc:VehicleType, and nc:VesselType each extend nc:ConveyanceType.

A person, on the other hand, may act in several different roles based on the situation. One person may act as an enforcement official and as a witness (both defined in the Justice domain), and as a caregiver (defined in the Human Services domain) as well. Because these functions of a person are not exclusive, they are defined as roles rather than extensions.

Simple roles

Note that in order to preserve simplicity whenever possible, NIEM only creates role types when there are role-specific properties to add. In 4.0, the Human Services domain has element "Caregiver". Since there are no caregiver-specific properties being added, the element simply uses type "nc:PersonType". This property is considered a role, but is defined like other standard elements.

Custom role types

If there are role-specific properties to add, a role type must be created. Because the Justice domain has specific properties for an enforcement official, it has created type j:EnforcementOfficialType that sets up the role and adds the additional content.

A role type contains one or more "RoleOf" elements and the additional role-specific properties. "RoleOf" elements are used in place of type extension. Instead of j:EnforcementOfficialType extending nc:PersonType, it contains element "nc:RoleOfPerson" (type nc:PersonType). It also contains additional properties for things like badge and unit.

RoleOf properties can be used like other properties. In an instance, content can appear inline or the RoleOf property can reference content listed elsewhere.

> "RoleOf" properties provide an additional benefit over type extension, allowing one role to be the function of **one or more objects**. This is handled by simply adding multiple "RoleOf" properties in a role type. Type extension is limited to derivation from a single type.

Examples

Kind	Property Name	Type	Comments
Simple	nc:AssessmentPerson	nc:PersonType	Nothing additional required
Role - one object	nc:Weapon	nc:WeaponType	Type contains element "nc:RoleOfItem"
Role - multiple	nc:Evidence	nc:EvidenceTye	Type contains elements "nc:RoleOfItem", "nc:RoleOfBiometric", and "nc:RoleOfBinary"

Kind	Property Name	Type	Comments
objects			

Modeling guidance

Create a role type for a non-exclusive function of an object only if there are role-specific properties to add. Otherwise, simply reuse an existing type, like nc:PersonType or nc:ItemType.

Role types

Parent type

A role type should extend “structures:ObjectType”.

RoleOf element usage

A role type will contain references to one or more RoleOf elements. It is not required, but NIEM customarily adds role-of elements first to a role type, and then the remaining properties.

RoleOf elements

Reuse

NIEM already defines several role-of properties, including…

Role-of Element	Type
nc:RoleOfPerson	nc:PersonType
nc:RoleOfOrganization	nc:OrganizationType
nc:RoleOfItem	nc:ItemType
nc:RoleOfFacility	nc:FacilityType
j:RoleOfBinary	nc:BinaryType
j:RoleOfBiometric	biom:BiometricDataType
mo:RoleOfUnit	mo:UnitType

Reuse an existing role-of property when creating a custom role type if it meets your requirements; otherwise, you may define your own.

Name

The name of a role-of property must begin with “RoleOf”.

Type

The type of a role-of property should be the type that represents the object of the role.

If the role is a function of a person, the type of the role-of element should be “nc:PersonType”.

Substitution Group

A role-of element must be substitutable for element “nc:RoleOfAbstract”.

XML

Role types

Inline instance example

In the instance example below, the basic information about the person (name, birthdate) appears **under** the nc:RoleOfPerson element.

```
<j:CrashDriver>
  <nc:RoleOfPerson>
    <nc:PersonName>
      <nc:PersonFullName>John Doe</nc:PersonFullName>
    </nc:PersonName>
    <nc:PersonBirthDate>
      <nc:Date>1966-06-06</nc:Date>
    </nc:PersonBirthDate>
  </nc:RoleOfPerson>
  <j:CrashDriverViolationCode>A10</j:CrashDriverViolationCode>
  <j:CrashDrivingViolationCode>S16</j:CrashDrivingViolationCode>
</j:CrashDriver>
```

Reference instance example

In the instance example below, the basic information about the person (name, birthdate) is **linked** from the nc:RoleOfPerson element to element nc:Person, which is defined elsewhere in the instance.

```
<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:PersonName>
    <nc:PersonFullName>John Doe</nc:PersonFullName>
  </nc:PersonName>
  <nc:PersonBirthDate>
    <nc:Date>1966-06-06</nc:Date>
  </nc:PersonBirthDate>
</nc:Person>
```

Schema example

This example shows a role type (CrashDriverType) and an element with that type (CrashDriver).

- The role type
  - uses one role-of element (nc:RoleOfPerson)
  - adds additional role-specific properties
  - extends structures:ObjectType (not nc:PersonType)
- The element of the role type (CrashDriver) requires no special syntax or features.

```
<xs:complexType name="CrashDriverType">
  <xs:annotation>
    <xs:documentation>A data type for a motor vehicle driver involved in a tra
  </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"
        <xs:element ref="j:CrashDriverDistraction" minOccurs="0" maxOccurs="un
        <xs:element ref="j:CrashDriverViolation" minOccurs="0" maxOccurs="unbo
        <xs:element ref="j:CrashDrivingViolation" minOccurs="0" maxOccurs="unb
        <xs:element ref="j:CrashDriverAugmentationPoint" minOccurs="0" maxOccu
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="CrashDriver" type="j:CrashDriverType" nillable="true">
  <xs:annotation>
    <xs:documentation>A motor vehicle driver involved into a traffic accident.
  </xs:annotation>
</xs:element>
```

Schema template

- In the template below…
  - One or more role-of elements may be referenced to represent the object(s) of the role.
  - One or more additional properties may be referenced, representing the role-specific content.

```
<xs:complexType name="NAMEType">
  <xs:annotation>
    <xs:documentation>A data type for ...</xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      <xs:sequence>
```

```
        <xs:element ref="ROLE-OF-ELEMENT-NAME" minOccurs="0" maxOccurs="unbour
        <xs:element ref="ADDITIONAL-PROPERTY" minOccurs="0" maxOccurs="unbound
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Role-of element

Schema example

The example below shows the definition of role-of element “RoleOfPerson” from Core.

- The role-of element has the type of the object being represented, in this case, nc:PersonType.
- The role-of element must substitute element “nc:RoleOfAbstract”

```
<xs:element name="RoleOfPerson" type="nc:PersonType" substitutionGroup="nc:Rol
  <xs:annotation>
    <xs:documentation>A person of whom the role object is a function.</xs:docu
  </xs:annotation>
</xs:element>
```

Schema template

```
<xs:element name="RoleOfNAME" type="NAMEType" substitutionGroup="nc:RoleOfAbst
  <xs:annotation>
    <xs:documentation>DEFINITION</xs:documentation>
  </xs:annotation>
</xs:element>
```

JSON

Role types

Inline instance example

In the instance example below, the basic information about the person (name, birthdate) appears **under** the nc:RoleOfPerson element.

```
{
  "j:CrashDriver": {
    "nc:RoleOfPerson": {
      "nc:PersonName": {
        "nc:PersonFullName": "John Doe"
      },
      "nc:PersonBirthDate": {
        "nc:Date": "1966-06-06"
      }
```

```
    },
    "j:CrashDriverViolationCode": "A10",
    "j:CrashDrivingViolationCode": "S16"
  }
}
```

Reference instance example

In the instance example below, the basic information about the person (name, birthdate) is **linked** from the nc:RoleOfPerson element to element nc:Person, which is defined elsewhere in the instance.

```
{
  "j:CrashDriver": {
    "nc:RoleOfPerson": {
      "@id": "BRAVO"
      }
    },
    "j:CrashDriverViolationCode": "A10",
    "j:CrashDrivingViolationCode": "S16"
  },

  "nc:RoleOfPerson": {
    "@id": "BRAVO",
    "nc:PersonName": {
      "nc:PersonFullName": "John Doe"
    },
    "nc:PersonBirthDate": {
      "nc:Date": "1966-06-06"
    }
  }
}
```

Schema example

This example shows a role type (CrashDriverType) and an element with that type (CrashDriver).

- The role type
  - uses one role-of element (nc:RoleOfPerson)
  - adds additional role-specific properties
  - extends structures:ObjectType (not nc:PersonType)
- The element of the role type (CrashDriver) requires no special syntax or features.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "properties": {
    "j:CrashDriver": {
      "description": "A motor vehicle driver involved into a traffic accident.
```

```
      "oneOf": [
        {
          "$ref": "#/definitions/j:CrashDriverType"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/j:CrashDriverType"
          }
        }
      ]
    }
  }

  "definitions": {
    "j:CrashDriverType": {
      "description": "A data type for a motor vehicle driver involved in a tra
      "allOf": [
        {
          "$ref": "#/definitions/_base"
        },
        {
          "type": "object",
          "properties": {
            "nc:RoleOfPerson": {
              "$ref": "#/properties/nc:RoleOfPerson"
            },
            "j:CrashDriverViolation": {
              "$ref": "#/properties/j:CrashDriverViolation"
            },
            "j:CrashDrivingViolation": {
              "$ref": "#/properties/j:CrashDrivingViolation"
            }
          }
        }
      ]
    }
  }
}
```

Schema template

- In the template below…
  - One or more role-of elements may be referenced to represent the object(s) of the role.
  - One or more additional properties may be referenced, representing the role-specific content.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
```

```
  "additionalProperties": false,
  "properties": {
  }

  "definitions": {
    "ROLETYPE": {
      "description": "A data type for ...",
      "allOf": [
        {
          "$ref": "#/definitions/_base"
        },
        {
          "type": "object",
          "properties": {
            "ROLE-OF-PROPERTY": {
              "$ref": "#/properties/ROLE-OF-PROPERTY"
            },
            "ADDITIONAL-PROPERTY": {
              "$ref": "#/properties/ADDITIONAL-PROPERTY"
            }
          }
        }
      ]
    }
  }
}
```

Role-of element

Schema example

The example below shows the definition of role-of element “RoleOfPerson” from Core.

- The role-of element has the type of the object being represented, in this case, nc:PersonType.
- The role-of element must substitute element “nc:RoleOfAbstract”

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "properties": {
    "nc:RoleOfProperty": {
      "description": "A person of whom the role object is a function.",
      "oneOf": [
        {
          "$ref": "#/definitions/nc:PersonType"
        },
        {
          "type": "array",
          "items": {
```

```
            "$ref": "#/definitions/nc:PersonType"
          }
        }
      ]
    }
  }
}
```

Schema template

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "properties": {
    "nc:RoleOfNAME": {
      "description": "DEFINITION",
      "oneOf": [
        {
          "$ref": "#/definitions/OBJECT-TYPE"
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/OBJECT-TYPE"
          }
        }
      ]
    }
  }
}
```

NDR references

NDR 5.0

- NDR Section 10.2.2 - Role types and roles

Rule	Applicability	Title
NDR 10-3	REF, EXT	RoleOf element type is an object type
NDR 10-4	REF, EXT	Only object type has RoleOf element
NDR 10-5	REF, EXT, INS	RoleOf elements indicate the base types of a role type
NDR 10-6	INS	Instance of RoleOf element indicates a role object

NDR 4.0

- NDR Section 10.2.2 - Role types and roles

Rule	Applicability	Title
NDR 10-3	REF, EXT	RoleOf element type is an object type
NDR 10-4	REF, EXT	Only object type has RoleOf element
NDR 10-5	REF, EXT, INS	RoleOf elements indicate the base types of a role type
NDR 10-6	INS	Instance of RoleOf element indicates a role object

Content

NIEM is a large data model, with over 12,000 elements. Because of the size, it might seem overwhelming to new users, but it isn't necessary to try to review each element. A basic familiarity with Core and the domains and the ability to search the model is enough to get started.

- Core
- Domains
    - Agriculture
    - Biometrics
    - Chemical, Biological, Radiological, Nuclear
    - Emergency Management
    - Human Services
    - Immigration
    - Intelligence
    - Infrastructure Protection
    - International Trade
    - Justice
    - Maritime
    - Military Operations
    - Screening
    - Surface Transportation

A good way to begin is to browse through the lists of common terms below to get a very high level overview of the kinds of components that are in Core and the domains. After that basic overview, follow the links that are most relevant to you to learn more.

These terms are counted by the number of times they appear within NIEM property names. This list is provided as a starting point for learning about what kind of content is represented in these namespace. The links will take you to search results in the Movement tool.

- Want to view components in the SSGT? Click here to open the tool, enter the terms manually in the search box and click on "Advanced Search" to filter by namespace.

Core

- Person
- Location
- Document
- Item
- Name
- Organization
- Facility
- Activity
- Contact
- Address
- Device

- Status
- Binary
- Obligation
- Unit

Domains

Agriculture

- Acreage
- Geospatial
- Modified
- Crop
- Reason

Biometrics

- Image
- Minutiae
- Ridge
- Segment
- Feature

Chemical, Biological, Radiological, Nuclear

- Data
- Rad
- File
- Time
- Rate

Emergency Management

- Alarm
- Notification
- Resource
- Event
- Alert

Human Services

- Juvenile
- Child
- Case
- Support
- Enforcement

Immigration

- Alien
- Benefit
- Immigration
- Program
- Role

Intelligence

- Person
- Agency
- Subject
- Interest
- Handling

Infrastructure Protection

- Asset
- Sector
- Segment
- Sub
- Name

International Trade

- Transport
- Goods
- Means
- Location
- Container

Justice

- Subject
- Person
- Assessment
- Hearing
- Parole

Maritime

- Vessel
- Port
- Location
- Cargo
- Crew

Military Operations

- Unit
- Location
- Event
- Observed
- Object

Screening

- Person
- Status
- Encounter
- Classification
- Document

Surface Transportation

- Roadway
- Traffic
- Location
- Annual
- Average

IEPDs

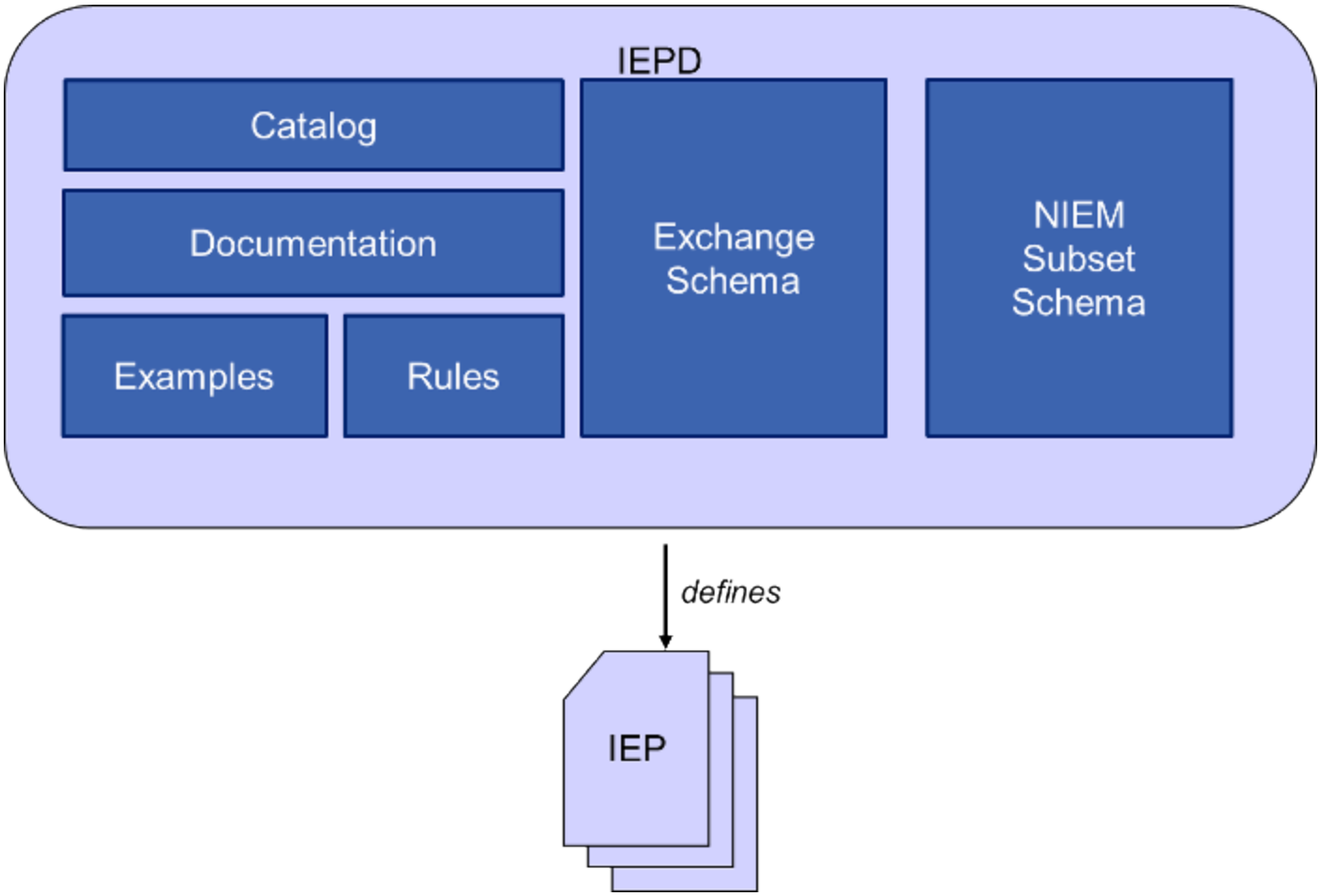
Overview

An Information Exchange Package Documentation (IEPD) is a collection of NIEM artifacts. They define and describe the context, content, semantics, and structure of one or more implementable information exchanges.

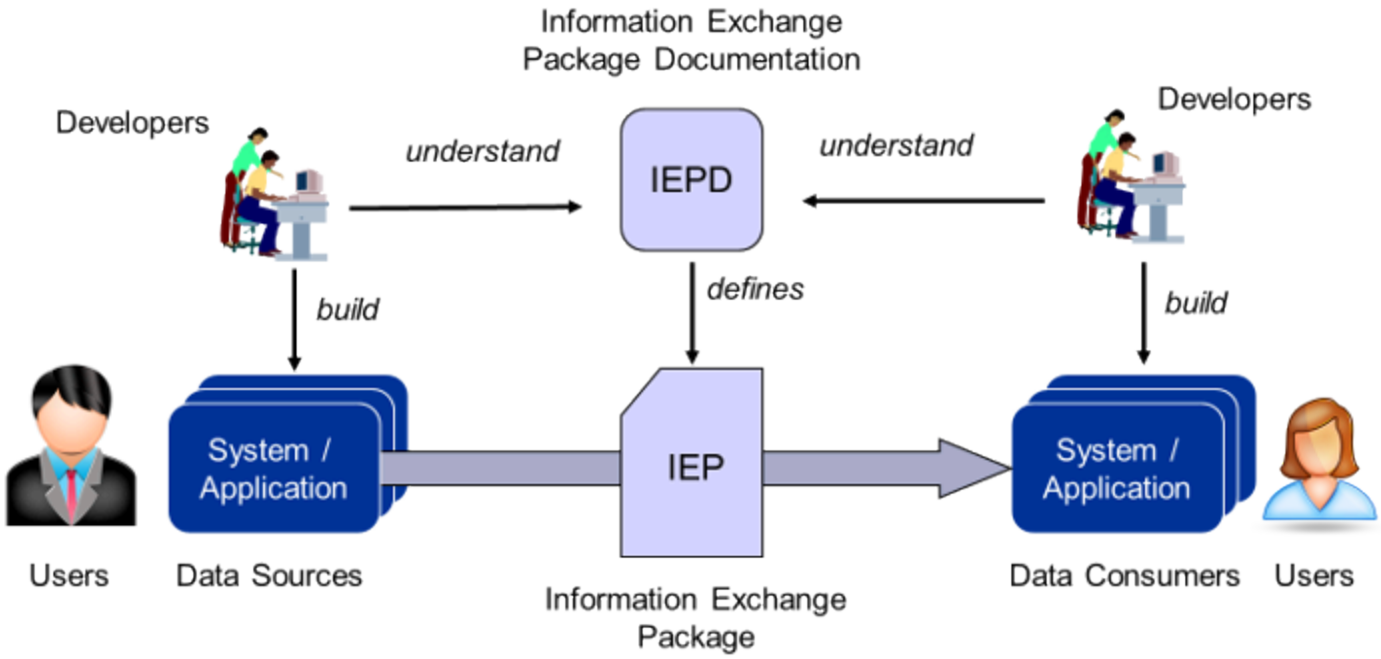
- Overview
- IEPs
- See more

- The rules for constructing IEPDs are defined by the Information Exchange Package Documentation (IEPD) Specification.
- IEPDs are composed from other NIEM artifacts, including subset schemas, extension schemas, sample instances, and an IEPD catalog.
- IEPDs provide a self-contained definition for derived information exchanges, thus they must be self documenting.

An Information Exchange Package Documentation (IEPD) is a NIEM message specification. It is a set of artifacts that define and document a message, using applicable content from NIEM and custom content as needed. Extra work goes into creating an IEPD so that it contains all of the information needed by developers to implement it.



To NIEM, the IEPD is considered the point of interoperability. Creation and management of IEPDs is the responsibility of their stakeholders and developers. As such, IEPDs have their own versioning processes, and are managed independently of NIEM and NIEM Domains. The NIEM Management Office defines IEPD conformance, but IEPD development and management fall outside its scope.



An IEPD contains a NIEM-conformant Schema Document Set (SET) that may include portions of NIEM Core and Domain Reference Schema Documents (REFs), and enterprise-specific or IEPD-specific Extension Schema Documents (EXTs). The schema documents contained in an IEPD work together to define one or more classes of instance documents that consistently encapsulate data for meaningful information exchanges. Any instance document that is valid for the SET, and that satisfies the conditions which define an IEP, is a member of a class of IEPs.

Schema documents in an IEPD conform to the NIEM Naming and Design Rules and may use or extend data component definitions drawn from NIEM. An IEPD may also incorporate and use schema documents from other standards that do not conform to NIEM. An IEPD consists of a set of artifacts (schema documents, documentation, sample instance documents, etc.) that together define and describe one or more implementable information exchanges. An IEPD should contain all materials necessary to:

- Understand information exchange context, content, semantics, and structure.
- Create and validate documents defined by the IEPD, and used for information exchanges.
- Identify the lineage of the IEPD itself and optionally its artifacts.

IEPs

Information Exchange Packages (IEPs) are the actual messages that carry data and are exchanged between stakeholders. The format and meaning of an IEP or message is specified by its Information Exchange Package Documentation (IEPD), which includes schemas, documentation, and other artifacts needed to fully describe the structure and semantics of the message.

See more

- IEPD Lifecycle

An Information Exchange Package Documentation (IEPD) developer designs, builds, and validates the components (artifacts) of an Information Exchange Package(IEP). The process consists of a six-phase lifecycle.

- IEPD Artifacts

  Each IEPD is a package that contains schemas and other supporting files.

- IEPD Starter Kit

  Learn about NIEM IEPD conformance, view sample IEPDs, and more.

- IEPD Change Control and Version Management

  Best practices for NIEM IEPD change control and version management.

IEPD Lifecycle

An Information Exchange Package Documentation (IEPD) developer designs, builds, and validates the components (artifacts) of an Information Exchange Package(IEP). The process consists of a six-phase lifecycle.

Lifecycle Phases

1. **Scenario Planning**: Review background information related to your information exchange, assess resource impact, understand business context, and identify information exchange business scenarios.

2. **Analyze Requirements**: Further elaborate on the information exchange scenario to understand and document the business context and data requirements.

3. **Map and Model**: Compare the data requirements for the exchange gathered in the last step to what is available in NIEM. Record results in a mapping spreadsheet and plan how requirements that aren't available in NIEM should be modeled and added as IEPD extensions.

4. **Build and Validate**: Create subset and extension schemas for the IEPD based on the mapping and modeling decisions made in the previous step. Check that schemas are valid and NIEM-conformant. Build and validate sample instances for additional testing.

5. **Assemble and Document**: Finalize the documentation and other IEPD artifacts and then package all related files for the IEPD into a single, self-contained, self-documented, portable archive file.

6. **Publish and Implement**: Publish the IEPD for search, discovery, and reuse, and implement the IEPD for production.

Scenario Planning

Review background information related to your information exchange, assess resource impact, understand business context, and identify information exchange business scenarios.

- Getting Started
  - Describe the scenario
  - Identify areas of concern
  - Determine whether to use NIEM
  - Define business requirements
  - Assess effort level
- Build scenario models
  - Use Case Diagrams
  - Business Process Diagrams
  - Sequence Diagrams
- Output
  - IEPD catalog basic data
  - Readme basic data
  - Diagrams and other documentation

Getting Started

Develop a high level scope and goal of the intended exchange. Decide what information needs to be a part of it. Interview stakeholders to gain an understanding of the information exchange environment. Stakeholders can include the organizations with whom you are exchanging information, community stakeholders in the same mission space, and practitioners from your organization.

Describe the scenario

Review existing documentation and ask exchange stakeholders questions such as the following:

- What information do you want to exchange?
- How much of this information do you exchange now; what changes need to be made?
- Who are the intended exchange partners?
- How are the current exchanges documented or specified?
- Are there any related NIEM IEPDs that could be reused or leveraged?
- Is there a subject matter expert group that has relevant expertise?
- What is the timeline and resources for development?

Identify areas of concern

Beyond the basic questions, consider complicating factors that may need to be addressed:

- jurisdictional boundaries
- privacy concerns
- sharing restrictions
- identity of participants

- people problems
- technical characteristics (e.g., types of data, number of data objects, physical and logical data structure)
- use of external standards

Determine whether to use NIEM

Determine if NIEM is right for this particular exchange.

Reasons to use NIEM include:

- Improve consistency and minimize redundancy in definition of data objects
- Reusability across multiple exchanges leads to cost reduction
- Better interoperability if NIEM is already being used within the organization or across the exchange partners
- The subject matter of the exchange aligns with content in NIEM, or is not already represented by a well-established standard
- Data is not managed consistently with an organization and NIEM could provide standard data objects and modeling guidance for internal systems to leverage when sharing data with other internal systems

Reasons why NIEM might not be right for the exchange:

- The subject matter of the exchange is already represented by another well-established standard or system that meets current needs
- Reusability does not provide value in this context

Define business requirements

Business requirements are the drivers for an information exchange that are primarily operational or functional requirements.

> The verification service shall respond only to search requests issued by the client application, the Verification Information System (VIS).

Developing business requirements

Follow a formal process to define and validate business requirements and accurately document an information exchange:

- Gather user requirements
- Define security and privacy requirements
- Specify exchange requirements

Not all requirements for an exchange are applicable to an IEPD. Some requirements are related to implementation and fall out of the scope of NIEM and IEPDs.

> A large, urban city identified the need to create a series of exchanges that would allow parents, city-wide, to apply online for their children's school meals program and automate the

processing of the request between the Human Services Agency and the school.

This will be implemented via a web-based service that, upon submission, will automatically check the city's Human Services agency database to ensure the applicant is from a low-income family and qualifies for the school meals program. If qualified, the web-based service will then notify the respective school to add the child to the meal program.

Currently, the only means the city has to process school meal program applications is through paper-based applications at the city Human Services Agency office. Once approved, the Human Services Agency faxes approved applications to the particular school where the child attends. Overall, it is a manual process that inhibits timely delivery of citizen services.

Examples of good business requirements from the business case above include:

Candidate Message Exchange Packages

- Request from Parent to Human Services Agency for meal program
- Eligibility request from Human Services Agency web-based service to eligibility system
- Eligibility reply from Human Services Agency to parent
- Notification to add child to meal program from Human Services Agency to school
- Request from school to Human Services Agency for monthly report
- Monthly report reply from Human Services Agency to school

Performance Requirements

- The web-based service will notify parents via email whether or not their child qualifies for the school meals program ten minutes after receiving their application.
- A school will receive notification within 24 hours of a new qualified child within their school.

Reporting Requirements

- Each school in the city shall be able to receive a monthly report of every child in their school who is currently enrolled in the school meals program.

Note: Data Quality Requirements should be taken into consideration once above requirements are discussed.

Assess effort level

Differences in exchange complexity will have an effect on the resources required to develop the IEPD and will assist in determining which IEPD artifacts should be developed as part of the exchange.

IEPD development efforts tend to be lower in the following situations:

- An exchange currently exists but needs to be updated or converted to NIEM
- Requirements are already known and well-documented
- NIEM is already being used in other exchanges

The following situations can lead to a higher level of effort required to develop or update and exchange:

- A new exchange is being developed
- An existing exchange or service needs to be converted to NIEM, but is poorly documented
- The exchange is very large and complex

All stakeholders associated with this exchange should be involved in determining the complexity so that all factors are understood and taken into account when planning the exchange.

Build scenario models

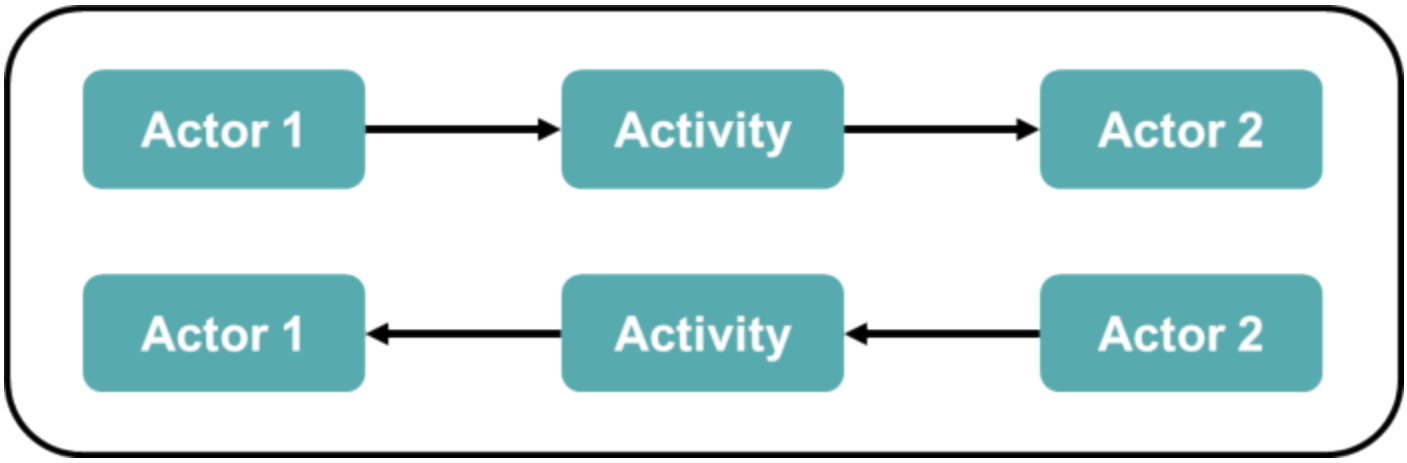
An effective scenario is informative, strategic, and comprehensive. A model is a graphical representation of information exchange requirements and is essential to depict the scenario that will ultimately drive the building of the information exchange. Use diagrams to create business scenarios as an effective first step to accomplish this.

Recommended diagram types include the following:

- **Use Case** - describes functionality
- **Business Process** - describes activity sequencing
- **Sequence** - describes message flow

Use Case Diagrams

A use case diagram graphically represents the functionality of a particular information exchange as perceived by an external observer.



Components

A use case diagram has the following components:

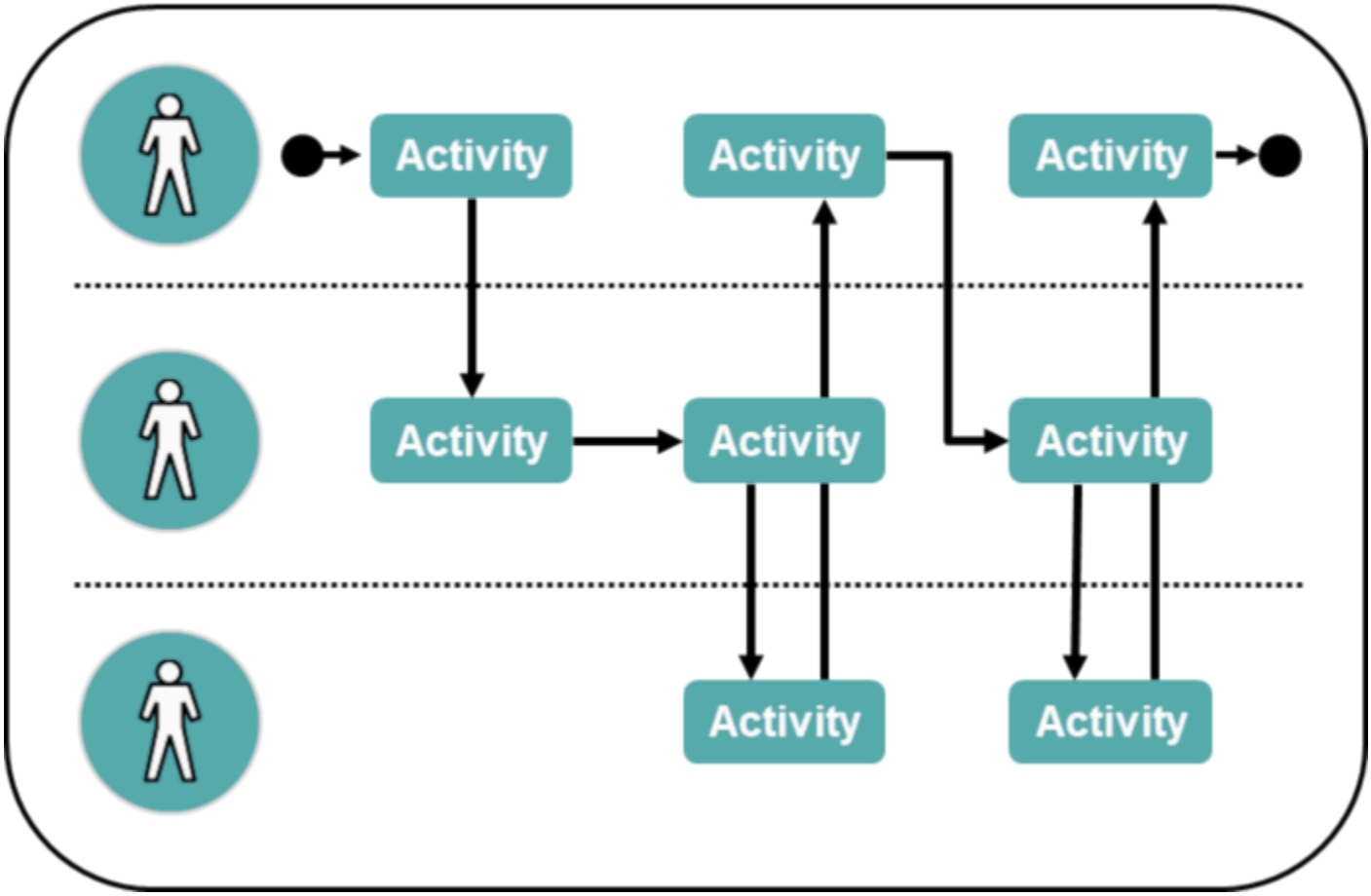
Actor	Depicts system interactions; a role that a human, device, or system “plays” within the diagram.
Use Case	Depicts system functionality such as main or sub functionality, a user goal, or an activity system.

Relationship	Depicts the relationships actors have with use cases and each other.
System	Contains all functionality and limits scope of the diagram.

Several of these simple diagrams can form a more complex diagram.

Business Process Diagrams

A business process diagram shows the graphical and sequential activities involved in an exchange. It is similar to a workflow diagram.



Components

A business process diagram has the following components:

Stakeholder	Any person, organization, or system directly or indirectly involved in the information exchange.
Activity	Correlates the data being exchanged with any activity that drives it.
Gateway-Event Based	Activities that force a decision upon a stakeholder.
Flow	Connects business processes and events to show the direction of activities in the diagram.
Start Event/Stop	Acts as an activity trigger or represents the result or completion of an activity.

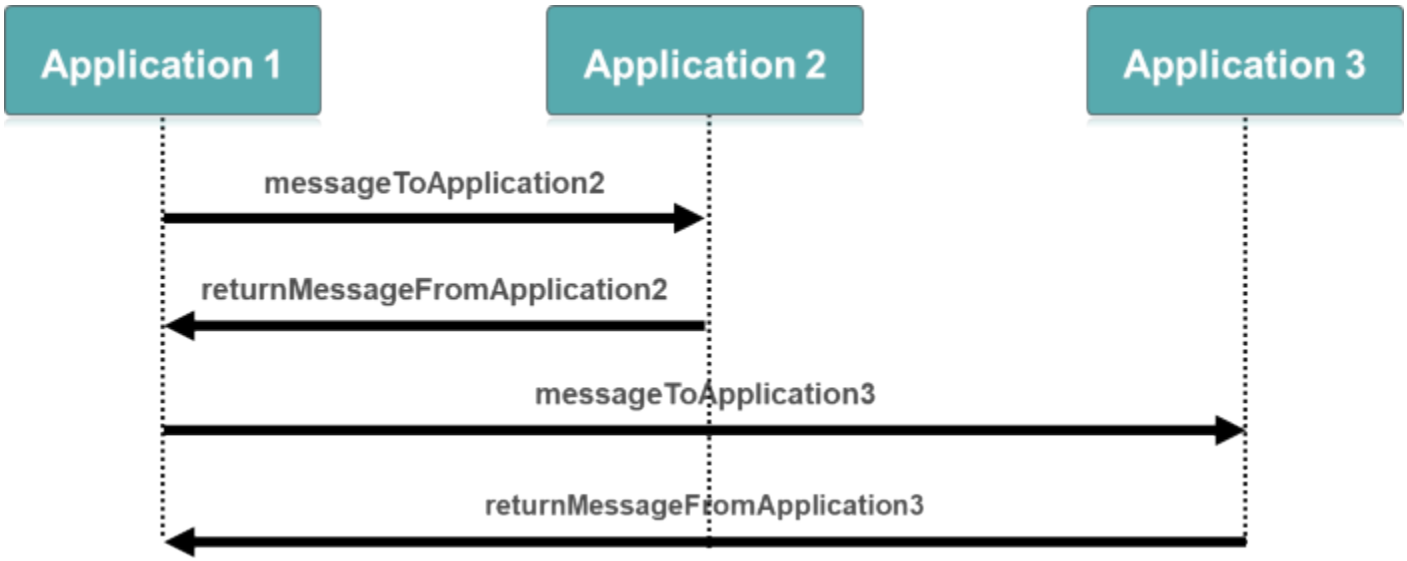
Event

Steps to develop a business process diagram:

1. Identify the stakeholders involved in the information exchange.
2. Create a box to serve as a frame to contain your diagram.
3. Divide the box into horizontal rows ("swim lanes") for each of the stakeholders in the process.
4. Draw a line dividing the header section to indicate each stakeholder from the section that will contain the stakeholder's activities.
5. Insert each of the stakeholders into the header section so that they have their own "swim lane" in the diagram.
6. Create a starting node on the diagram in the far left of the swim lane for the stakeholder that initiates the process for the information exchange.
7. Fill in the business process diagram from left to right, beginning after the starting node, with each of the activities or events, in chronological order, involved in the information exchange. Also make sure to insert a terminal node that indicates the end of the process.
8. Fill in the arrows that connect each of the activities in the diagram.
9. Review the diagram to ensure that it matches the process described in the scenario.

Sequence Diagrams

A sequence diagram shows how applications or systems operate with one another. It displays the sequential order of operational processes or messages between applications as horizontal arrows between the parallel, vertical lines that are used for applications. This allows the specification of simple runtime scenarios in a graphical manner.



Components

A typical sequence diagram should include the following elements:

Applications Any application involved in sending messages within the information exchange.
Messages Any message being sent between applications within the information exchange.

Output

At the end of this phase, high-level information about the IEPD can begin to be documented and captured. While this is not required and the information can continue to be updated and refined during the rest of the IEPD development process, it can be useful to start recording thoughts and decisions now.

IEPD catalog basic data

An IEPD Catalog is one of the required artifacts of an IEPD. This will be filled in during the *Assemble and Document* phase of the IEPD lifecycle (Phase 5), but some basic data that will be needed can begin to be collected now.

Field	Description
IEPD name	A name or title for the IEPD
IEPD version	Identifier for a specific release of an IEPD. “1.0” is typical for new exchanges.
NIEM release	Determine which NIEM release should be used. Typical to use the current NIEM release unless an older release is already in use within the organization or by the exchange partners. This will determine (1) which release will be used for the NIEM subset in the IEPD (2) which version of the Naming and Design Rules (NDR) should be used for conformance testing (3) whether the IEPD should follow the 3.0 or 5.0 version of the MPD / IEPD specification
Related IEPDs	Capture information about any related IEPDs that may be extended, reused, or otherwise leveraged
Message names	One IEPD can be built that defines multiple kinds of related messages. One example is a request / response pair of messages. Document the names and descriptions of each kind of message if more than one will be built.
Authoritative source	What organization will serve as the authoritative source for the IEPD? Will need a name and point of contact information.
Keywords	Common aliases, terms, or phrases that would help other users discover this IEPD
Subject matter domain	The community in which this IEPD is applicable or used. This does not need to correspond to a domain that already exists in NIEM or will be used in the IEPD.
Purpose	A description of the intended usage and reason for the IEPD
Exchange pattern	A description of the transaction or design pattern by which information will be shared
Exchange partners	Names or kinds of the organizations that will use this IEPD to share data.

Readme basic data

A Readme is another required artifact in an IEPD. This will serve as the introductory human-readable documentation and should include the following:

- Purpose of the IEPD

- Scope of its deployment, usage, and information content
- Business value and rationale for developing it
- Type of information it is intended to exchange, in business terms
- Identification (or types) of senders and receivers
- Typical interactions between senders, receivers, and systems
- References to other documentation within the IEPD
- Links to external documents that may be needed to understand and implement the IEPD

Diagrams and other documentation

Any of the diagrams or documents created to analyze and document the exchange scenario should also be included in the final IEPD package. Keep these together and update as needed during the IEPD development process.

Analyze Requirements

Further elaborate on the information exchange scenario to understand and document the data requirements and business rules.

- Requirements spreadsheet
  - Example fields
  - Example codes
- UML class diagrams
  - Components in a diagram
    - Objects
    - Elements
    - Associations
      - Generic
      - Aggregation
      - Composition
      - Specialization
    - Cardinality
  - Steps
  - Example
    - Identify and draw objects
    - Identify and draw elements
    - Identify and draw associations
- Output

After planning out the scenario for the information exchange at a higher level, the next phase involves analyzing the exchange requirements at a more detailed level. This includes capturing data requirements, business rules, and other detailed requirements of the exchange.

Data requirements

Data requirements describe the actual data that will be shared in the exchange.

> The exchange will need to capture information about a student, including the student's first, middle, and last name and date of birth.

Business rules

Business rules are specific qualifications for data that refer to how it should be structured (e.g., field length, constraints).

> A student's enrollment status is not available or is unknown. The assumption is they are not enrolled, and an information-needed flag will be added by setting X element to the value of TRUE.

Compile as much information as possible about the data requirements and business rules. This includes documenting existing field names, data types, definitions, cardinality, sample values, and code sets in current exchanges or data sources. Plans for new fields or modifications to existing ones should be

documented as well.

Information about these data requirements can come from a variety of sources, including schemas or API documentation for current or related exchanges, and database fields. Sometimes a lot of details will be available; at other times, only minimal documentation will exist. Collect what's available and fill in the blanks as much as possible.

Two common ways of capturing this kind of information are via spreadsheets and UML diagrams.

Requirements spreadsheet

Example fields

Data requirements can be captured in a spreadsheet

The following shows how a few simple data requirements might be documented in a spreadsheet:

Field	Type	Definition	Min	Max	Constraints	Comments
first	string	First name	1	1		
middle	string	Middle name	0	unbounded		
last	string	Last name	1	1		
DOB	date	Birthday	0	1	Valid only between 1900-01-01 and current date	Check to see if this field needs to be required
eye color	code set		0	1		

A spreadsheet like this can be expanded as needed to add additional information. In some cases, data requirements may come from multiple sources and may sometimes overlap. Columns can be added to record the source of the requirement and to capture how they map to other requirements. If the current requirements are structured so that elements are nested in a hierarchy, columns can be added to capture the parent class or path of the field as well.

Example codes

The following shows how code sets might be documented in a spreadsheet:

Type	Code	Definition
eye color	BLK	black
eye color	BRN	brown
eye color	BLU	blue
eye color	GRN	green
eye color	HZL	hazel
eye color	OTH	other or unknown

One benefit of using a spreadsheet is that it can be easily used in the next phase of the IEPD lifecycle

when capturing mappings of data requirements to components in NIEM.

UML class diagrams

Data requirements can also be captured in Unified Modeling Language (UML) class diagrams.

Components in a diagram

The following is the major kinds of information that will appear in a class diagram:

Objects

- Something that represents a physical or conceptual thing; person, place, location, idea, etc.
- These are structured and able to contain instances of themselves and elements
- In a class diagram, these are represented by a rectangle with the object name at the top.

Elements

- Defines the unique characteristics of an object
- In a class diagram, these are represented inside the object's rectangle, in a separate panel under the object, and notated with the element name and type (e.g., `age: number`)

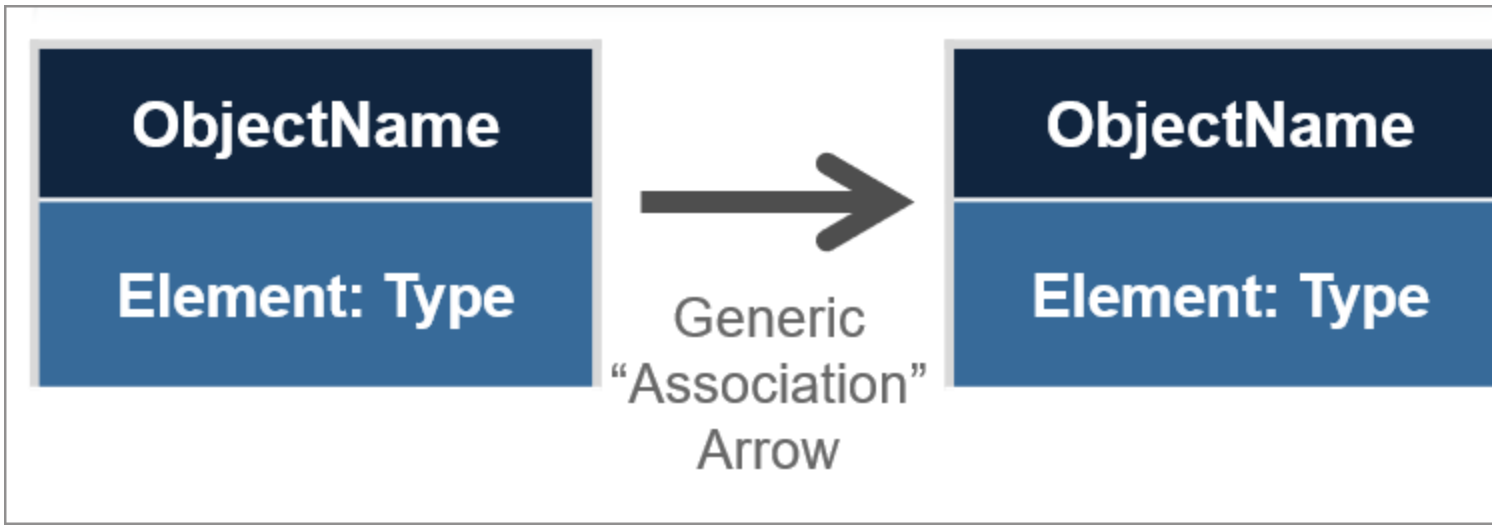
Associations

- Defines the relationships between objects
- Many different types of associations exist to relate one object to another
- Associations use types of directional arrows to represent the role of each object to the other (e.g., relates-to, contains-a, has-a)
- A good practice is to name / label associations in a class diagram to make relationships clearly understood

The kinds of associations typically used in class diagrams to describe exchanges include:

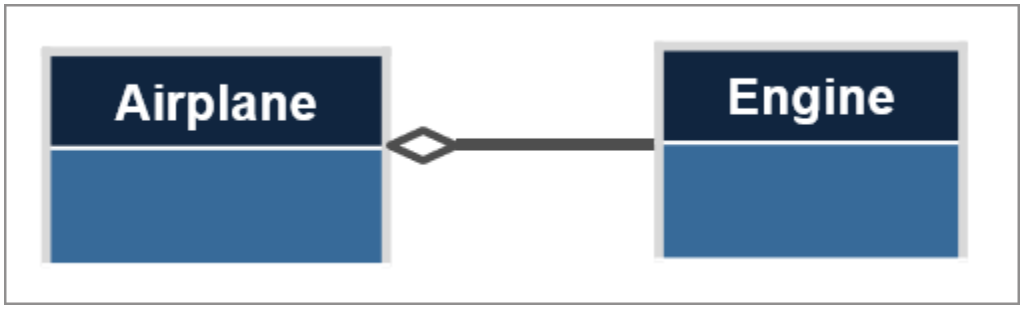
Generic

- Generic associations are non-specific relationships between objects.
- These are drawn with a solid line from one object to another. In NIEM, they are typically unidirectional with an open-ended arrow but can be bidirectional.



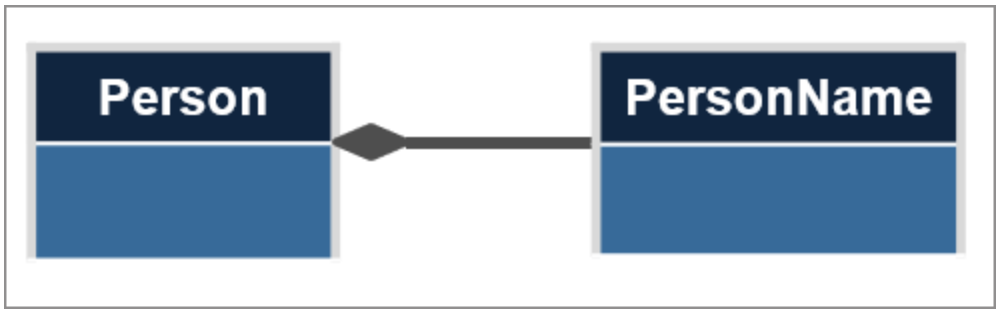
Aggregation

- Aggregation associations depict a non-exclusive relationship between the parent and child objects, where the child object may exist independently of the specific parent object.
- For example, aggregation could be used to describe the association between an airplane and an engine - an engine can exist independently from an airplane and could appear under other kinds of conveyances.
- These are drawn using a solid line with an empty diamond arrowhead at the parent and no arrowhead at the child.



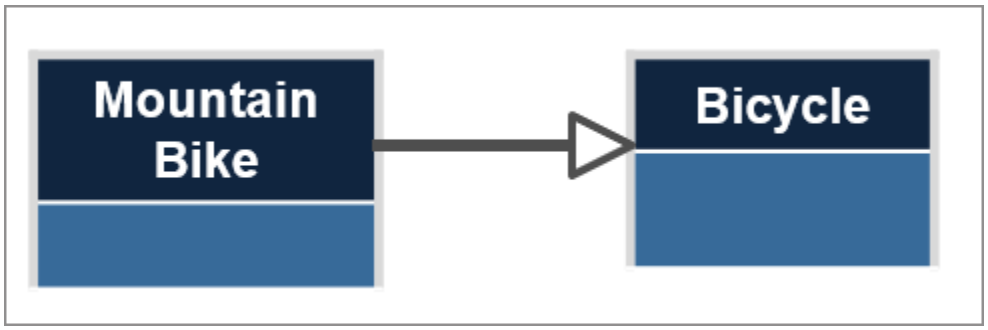
Composition

- Composition associations depict an exclusive relationship between the parent and child objects, where the child object would not exist without the parent object.
- For example, composition could be used to describe the relationship between a person object and the person name object.
- These are drawn using a solid line with a filled diamond arrowhead at the parent and no arrowhead at the child.



Specialization

- Specialization associations depict a special kind of object that inherits and extends attributes from its parent. Can also be referred o as inheritance or generalization.
- For example, a mountain bike is a kind of bicycle and should inherit from it.
- These are drawn using a solid line with an open triangle arrowhead at the parent and no arrowhead at the child.



Cardinality

- Defines the number of occurrences between objects, elements, and associations
- In a class diagram, this can be represented by:
    - one number that defines the exact number of occurrences (e.g., `1`)
    - two numbers that represent the minimum and maximum number of occurrences (e.g., `0..1`)
    - use an asterisk to represent many occurrences (e.g., `0..*`).

Steps

The recommended steps for representing data requirements as a content model in a UML class diagram are:

- Identify and draw objects
- Identify and draw elements
- Identify and draw associations

The end result is a visual representation of the data objects and elements that will be included within an exchange

Example

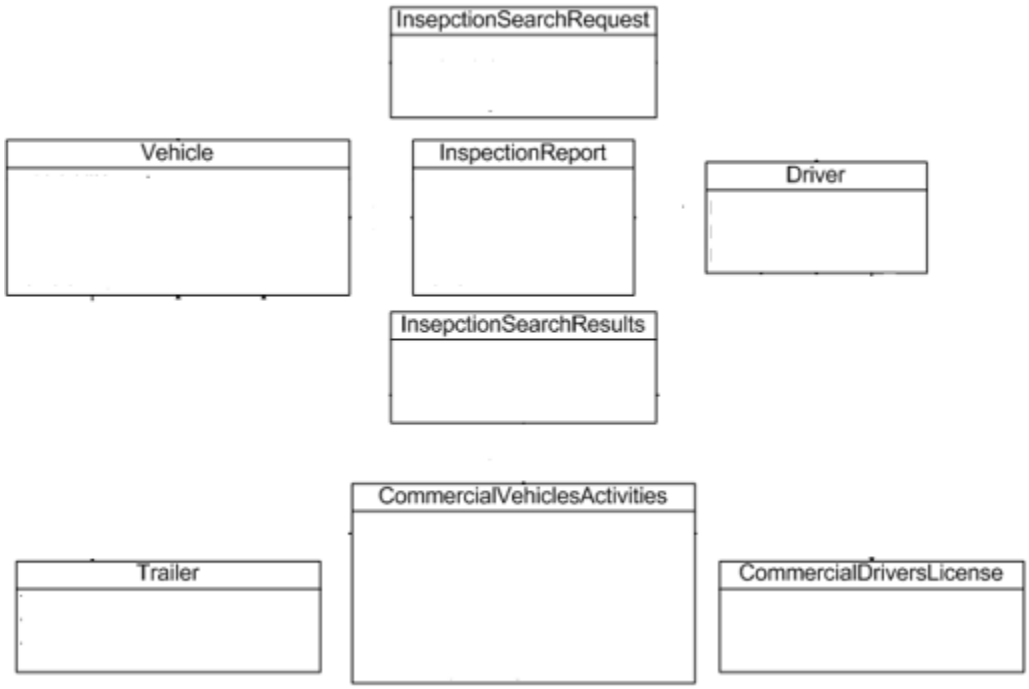
The following example demonstrates how to build a content model in a UML class diagram for the Customs Commercial Vehicle Tracker information exchange.

Identify and draw objects

After planning out the exchange scenario, the following objects have been identified as the major categories of data requirements:

- Driver
- Commercial Driver License
- Vehicle
- Commercial Vehicle Activities
- Trailer
- Inspection Search Request
- Inspection Search Result
- Inspection Report

For each object, add a segmented rectangle to the class diagram with the object name in the top panel, as shown below:



Identify and draw elements

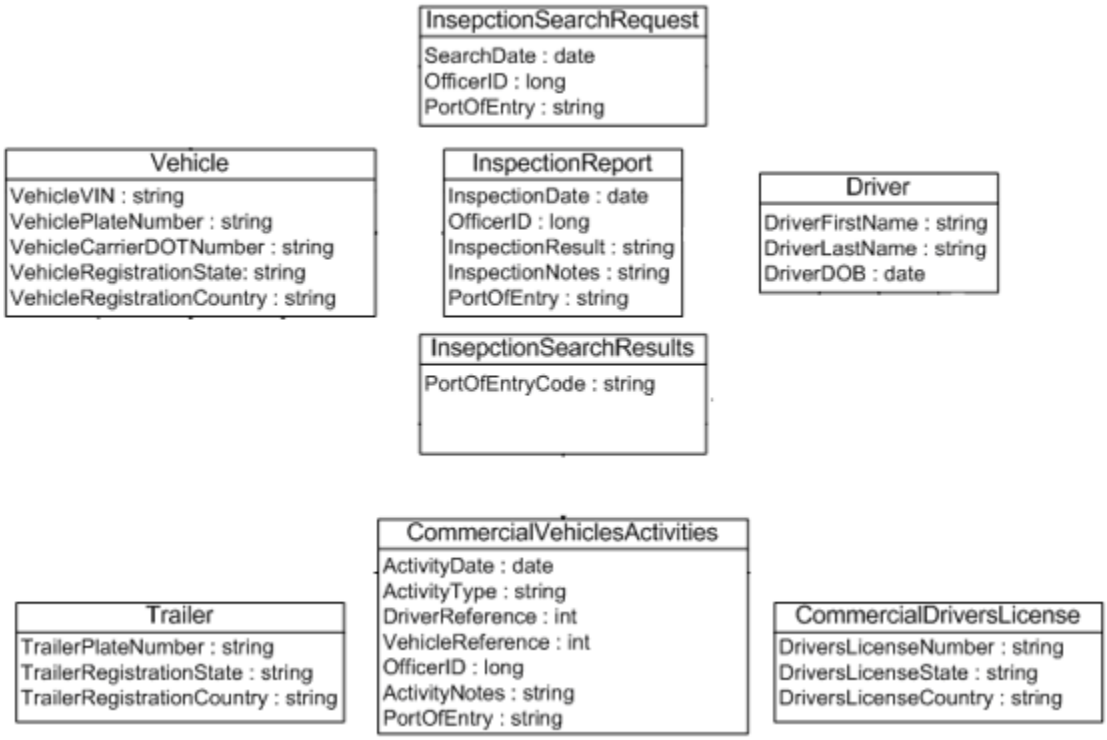
After identifying the objects and drawing the objects, determine the specific characteristics of each object. For example, the Driver object can have the following elements:

- Driver First Name : string
- Driver Surname : string

- Driver DOB : date

Similarly, identify the elements for each of the remaining objects.

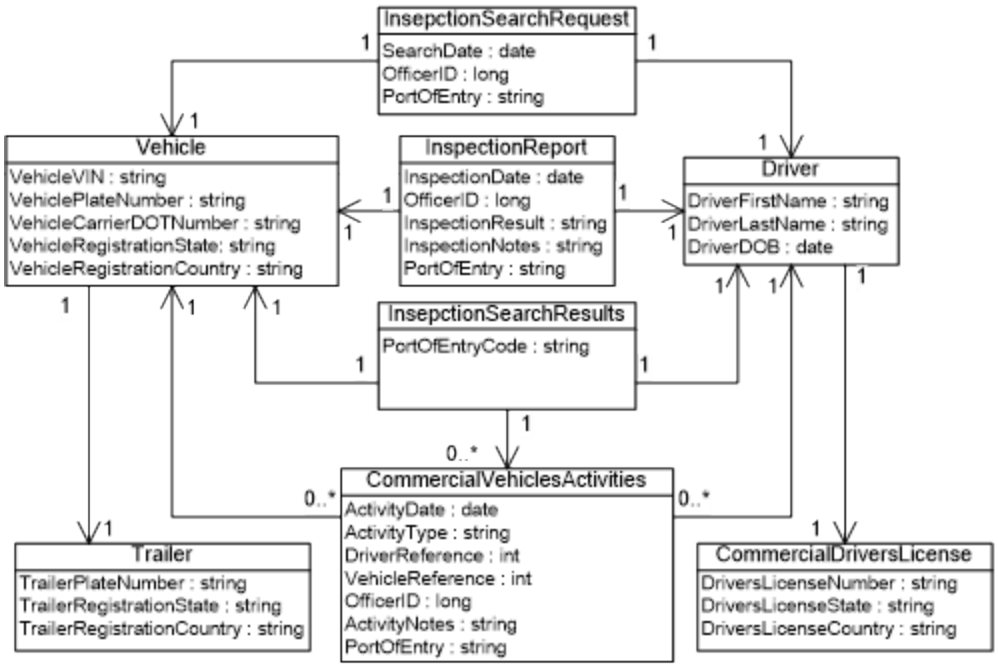
After identifying the elements of each object, fill in the element names and types in their corresponding object rectangles as shown below:



Identify and draw associations

After identifying and drawing the objects and elements of an exchange in a class diagram, the relationships between the objects should be captured. In this case, we find that all the objects are associated through inheritance. After finalizing the type of association between each object, determine the cardinality of such association.

Draw the appropriate arrows for the given association type and add cardinality as shown below:



Output

Documentation of data requirements and business rules should be included in the final IEPD package, whether specified via spreadsheet, UML class diagram, or another format. If a spreadsheet is used, that can help feed into the next phase of IEPD development, *Map and Model*.

Map and Model

Review the data requirements captured during the previous phase and discover if there are similar components in NIEM. Document results in a mapping spreadsheet. Begin modeling the data requirements without a match in NIEM; these will be later added added to the IEPD as extensions.

- Purpose
- Map requirements to NIEM
  - How to search NIEM
  - Mapping spreadsheet
    - Simple example
    - Template and examples
- Model extensions
  - Create new properties
  - Determine what types to use
    - Reuse a NIEM type
    - Create an augmentation
    - Create an extension
    - Create an association
    - Create a role
    - Create a new type
  - Create code sets
    - Create new code types
    - Create a union type
    - Add the code element
- Output

Purpose

Some IEPD developers skip the mapping document and go straight from documenting data requirements in the previous step to building schemas in the next step. This can be done and a mapping document is not a required artifact in an IEPD; however, this kind of mapping can provide the following advantages:

- Provides an easy way to allow others to review and confirm mapping and modeling decisions before beginning to build extension schemas.

  It can be useful to allow others, especially subject matter experts involved with the exchange, to review the work in progress. Spreadsheets are typically easier to use when reviewing lists of elements names and definitions, plus can be used to capture comments and feedback.

- Clearly captures the source data requirement for each of the properties that will appear in the exchange.

  This information is often fairly obvious during development, but can be easily forgotten over time or lost as the original IEPD developers move on to other projects. Reverse-engineering where initial requirements came from at a later date can be challenging.

Map requirements to NIEM

Identifying and recording mapping is a manual process. It becomes easier as you gain familiarity with NIEM and learn what's available and where things are, but can be slow-going for those new to NIEM. There is no automated way of identifying matches. For each data requirement, NIEM should be searched and the success or failure of that search should be recorded.

How to search NIEM

There are multiple ways NIEM can be searched for matches.

Tool / Product	Description
SSGT	Use the SSGT to search any NIEM release. Property, type, and code information will be available.
Movement	Use Movement to search the current NIEM release. Movement has a fast search capability and a more modern UI, but note that some information about NIEM is not displayed, like abstract elements, associations, code sets, and type details like inherited properties.
Model spreadsheet	Download the model spreadsheet that comes with a NIEM release and review property and type information in Core and domain-specific tabs.
Release schemas	Download a NIEM release package and explore the model in an XML or text editor.
Tools catalog	Check the Tools Catalog on niem.gov for a listing of third-party tools, some of which can be used to search and discover content in the NIEM model.

Mapping spreadsheet

A mapping spreadsheet should have three main sections:

Data requirements for the exchange

- Data requirements should have been collected as part of the *Analyze Requirements* stage.
- Include as much information as available. Field / element names are the required minimum. Additional information like definitions and basic type info should be included if present.
- This is usually captured on the left-hand side of the mapping spreadsheet.

Mapping status

- Record whether a match exists in NIEM for the local data requirement.
- When a match isn't found, this status should be recorded explicitly rather than left as a blank. It can be difficult to tell if a blank value in the mapping status means that there is no match, or if it means that the row simply hasn't been reviewed yet.

NIEM information

- At a minimum, capture the namespace prefix and name for the matching property in NIEM.
- If it's helpful, especially for review, also capture additional information like the type and definition of the NIEM property.

Simple example

The most basic kind of a mapping spreadsheet is shown below. This is the minimum amount of information needed to capture mappings between data requirements and NIEM components.

Data requirements	Mapping	NIEM
first	map	nc:PersonGivenName
middle	map	nc:PersonMiddleName
last	map	nc:PersonSurName
DOB	map	nc:PersonBirthDate
work experience	map	em:PersonEmploymentExperienceText
facebook username	no match	

Template and examples

NIEM provides a mapping spreadsheet template that can be used to both document mappings and to begin modeling extensions. To use the spreadsheet for mappings only, focus on the "Property" tab and ignore the other tabs unless needed. Customize the spreadsheet by hiding columns that won't be needed.

See the **Mapping Spreadsheet** page under the IEPD Artifacts section for the template and for a spreadsheet with some example data filled in.

Model extensions

IEPD developers can begin to plan out and model the content that does not map to NIEM and therefore will need to be added in extension schemas.

See the Mapping Spreadsheet – Modeling examples section for basic information about modeling new extensions in the mapping spreadsheet.

The following are some very basic tips for getting started with modeling new extensions:

Create new properties

> See the **Property** section under *Reference / Concepts* for more details about the various kinds of properties in NIEM, along with naming and definition tips and examples.

Come up with NIEM-conformant property names and definitions.

NIEM property names are composed of class terms, property terms, qualifier terms, and representation terms.

NIEM definitions for properties that carry a value (like a string, boolean, or number) should begin with the standard opening phrases provided by the NIEM Naming and Design Rules (NDR).

Determine what types to use

> See the **Type** section under *Reference / Concepts* for more details about the various kinds of types in NIEM, along with examples.

Reuse a NIEM type

Reuse a type directly from NIEM if possible.

> If `nc:PersonType` already contains everything needed for a new vice principal element, reuse that type from NIEM as is.

IEPDs can reuse types from Core, a domain (even if the domain doesn't directly relate to the IEPD's subject matter), a code set, or any other namespaces in NIEM.

Very frequently reused types in NIEM include `nc:PersonType`, `nc:OrganizationType`, `nc:ActivityType`, `nc:LocationType`, `nc:ContactInformationType`, `nc:ItemType`, `nc:TextType`, and `nc:DateType`.

If a NIEM type is close but doesn't quite fit all of the requirements, then there are several options:

Create an augmentation

Create an augmentation if the NIEM type is semantically a match but needs some extra properties added.

> If `nc:PersonType` seems like a good fit for the new vice principal element but the IEPD needs a few extra person-related properties that aren't specific to a vice principal (like a LinkedIn profile URL), create an augmentation.

Create an extension

Create a typical type extension if the requirements need a specialization of a NIEM type. This is also known as type inheritance or type specialization.

> If the IEPD needs to model a train, `nc:VehicleType` should be extended. An augmentation would not make sense here because additional train-specific properties for things like locomotive and number of cars would not apply for other generic vehicles.

Create an association

Create an association if the requirements need to relate two or more peer-level objects, often within a certain date range.

Associations can be especially useful to eliminate duplication in messages for many-to-many relationships.

If the IEPD needs to relate teachers and students, and also needs to capture information like grade, subject, and class period, create an association. This can be used to represent that one teacher might instruct one subject all day to different groups of students while another teacher might instruct the same group of students all day. The basic information about the teachers and students would appear once in the message, and then multiple instances of the association could be used, each containing pointers to the teacher and student data objects, plus the data values for the class-specific information.

Create a role

Create a role if:

1. the requirements need a specialization of a NIEM type,
2. the object being modeled could appear in different contexts or with different functions, and
3. duplication of content is a concern in the messages

If the IEPD needs to model a vice principal with specific properties for that position, but the person may also need to be represented in the IEPD as a coach and as a study hall monitor, then a role could be created. The role types for vice principal, coach, and monitor would each contain a pointer to person object, so basic person information does not have to be duplicated in the message.

Create a new type

Create a brand new type when no type in NIEM matches requirements.

The type should still likely extend `structures:ObjectType` in NIEM. This will allow it to be referenced, link to metadata, and contain security markup information.

Create code sets

If similar code sets exist in NIEM and the local codes from the data requirements can be mapped to them, try to use the NIEM version for better interoperability.

Create new code types

If that is not an option or a similar code set does not exist in NIEM, create the code set.

Create a union type

If an existing NIEM code set has most of the code values needed, but the IEPD requires a few extra codes, a union type can be created. Union types allow separate simple types to be combined together so multiple code sets can be combined into a single simple type without duplication.

Add the code element

A new code element should be created for the new code set type. This element can either then be added

directly to an extension type or to a substitution group in NIEM if one exists.

- Add new code element `ext:ClassSubjectCode` as a sub-element of type `ext:ClassType`.
- Add new code element `ext:PersonHairColorCode` as a substitution for NIEM element `nc:PersonHairColorAbstract`

Output

The mapping spreadsheet, filled with exchange data requirements, matching components in NIEM, and optionally containing information about how to start building extensions, will be used in the next phase of IEPD development, *Build and Validate*. This spreadsheet should also be included in the final IEPD package.

Build and Validate

Create subset schemas, and extension schemas, and sample instances based on the previous mapping and modeling work. Verify that schemas and instances are valid and that they meet NIEM conformance requirements described by the NIEM Naming and Design Rules (NDR) specification.

- Build
  - NIEM subset schemas
  - Extension schemas
  - Sample instances
- Validate
  - Schema validation
  - Sample instance validation
    - Validate format
    - Validate requirements
  - NDR conformance validation
    - Automated rules
    - Manual rules
- Output

Build

There are three main kinds of artifacts that should be built during this stage of the IEPD development process: NIEM subset schemas, IEPD extension schemas, and sample instances.

NIEM subset schemas

NIEM subset schemas should contain all of the NIEM properties and types that will be reused in the IEPD, plus any dependencies. IEPDs typically contain customized NIEM subsets rather the full set of NIEM release schemas (12,000+ elements in NIEM 5.0 and dozens of schemas) in order to improve performance and reduce size and complexity.

NIEM subset schemas are usually built with the Subset Schema Generation Tool (SSGT), which lets users search the NIEM model, select properties and types, and generate NIEM XML subset schemas. See the **SSGT** page under the Reference / Tools section for a lot more information about how to search NIEM, build subsets, and reload wantlists later for additional work.

Starting with the Mapping Spreadsheet filled in during the previous stage of the IEPD lifecycle, find each mention of a NIEM property or type, search for it in the SSGT, and add it to the subset. When finished, download the subset. Extract the subset schemas from the downloaded zip file into a NIEM subset folder within the IEPD working directory.

> Build NIEM subsets while filling in the mapping spreadsheet during the *Map and Model* phase in order to prevent looking up the same properties and types a second time. As matches are found in the SSGT and mappings are recorded, add the NIEM property or type to build the subset at the same time.

As extension schemas are developed, the NIEM subset schemas may need to be updated several times to account for additional NIEM properties or types that were not identified during the previous mapping phase. If the subset is no longer loaded in the SSGT, reload the wantlist from the previous subset package to resume work on the subset. Regenerate the subset and update the schemas in the IEPD working directory.

Extension schemas

Create one or more schemas to hold the new properties and types that will need to be created specifically for this IEPD. Extension schemas typically import multiple schemas from the NIEM subset, most often including Core and structures (for basic NIEM infrastructure support).

With the extension schema, create NIEM-conformant properties and types for the requirements identified in the mapping spreadsheet as either a “no match” or an “add”. Preliminary modeling work will have already been done for the requirements labeled as “add”. See the modeling information if needed from the previous lifecycle stage for a starting point on how to model local data requirements as NIEM extensions.

> See the **Concepts** section for lots of information about the various NIEM modeling constructs, and for schema and instance examples of each in XML and JSON. Start with Namespaces to set up the extension namespace.

Sample instances

Create at least one sample instance for each kind of message defined by the IEPD. Create more than one of each if needed to better illustrate different options for sending data.

Oxygen and XMLSpy are capable of initializing sample instances given schemas and a selected element to use as the root of the document. Instances generated in this manner should then be updated to add desired fields the tool may have left out of the message and to replace generic strings and other values with realistic-looking ones. This will help reviewers and IEPD implementers understand what messages should really look like.

Validate

If NIEM subset schemas are generated by the SSGT, no extra validation for these schemas should be needed. All IEPD extension schemas and any NIEM subset schemas generated by hand or by other methods should be tested for basic schema validation and NDR conformance validation.

Schema validation

Use tools or libraries like XMLSpy, Oxygen, Saxon, Xerces, or Ajv (JavaScript validator for JSON schema) to perform schema validation.

Sample instance validation

Validate format

Using the tools or libraries listed above, check the sample instances against their schemas to ensure they are valid.

Validate requirements

In addition to verifying schemas and instances conform to basic format and NDR rules, it's important to make sure that the IEPD meets the original data requirements defined in the *Analyze Requirements* lifecycle phase.

Go back to the data requirements that were documented in UML, a mapping spreadsheet, or a custom format, and compare that to the sample instance. Make sure all of the requirements are accounted for in the instance. It can be easy to leave small things out and sometimes difficult to discover without carefully comparing the original requirements to a thorough example of the final product.

NDR conformance validation

Automated rules

For XML schemas, the NIEM Naming and Design Rules (NDR) defines may of its rules in Schematron, allowing automated conformance testing. This can be done by either uploading the schemas to ConTesA or by validating the NDR Schematron rules with Oxygen. Review errors, update schemas and sample instances as needed, and rerun all validation steps until conformance issues have been resolved.

Manual rules

Not all rules in the NDR are able to be written in Schematron and automated; some rules must be evaluated manually. The conformance report generated by ConTesA will also provide a list of manual rules. The Rule Browser is another way to find and learn about the text-style NDR rules that require manual evaluation.

Output

At the end of this stage, the full set of NIEM subset schemas and IEPD extension schemas should be complete, and there should be one or more sample instances that illustrate what messages should look like. These are required artifacts for the IEPD.

Assemble and Document

Create any additional documentation needed for the IEPD, fill out the IEPD catalog, assemble all of the artifacts, and bundle the files into the final IEPD zip file package.

- Document
- Assemble
- Review
    - Basic checks
    - IEPD review checklist
    - Peer review
- Output

During this process, it may be necessary to go back to earlier steps to fix bugs, update schemas, fill in missing mappings, etc.

Document

At this stage, the schemas and sample instances for the IEPD should be complete. It's time to finish up the rest of the artifacts and documentation for the IEPD.

See IEPD Artifacts for information about the required artifacts and optional artifacts for an IEPD.

Among the other documentation that is being prepared, make sure to prepare the following required artifacts:

- IEPD catalog
- Change log
- Readme

Assemble

Once all IEPD artifacts have been prepared, the the next step is to compile and archive the final package:

See IEPD Artifacts – Recommended folder layout for the recommended layout of the IEPD directory.

Review

An IEPD Review Process is a good practice to maintain a consistent standard of quality in documents that are accepted for publication in support of organizational information sharing initiatives.

Benefits:

- Promotes IEPD Artifact consistency
- Minimizes redundant or inconsistent details between IEPDs

- Increases higher quality IEPD delivery
- Increases confidence in IEPD information
- Increases a standardized information sharing environment
- Promotes an ease of search, discovery and reuse of IEPDs

A review process will help to decrease the occurrence of published IEPDs that are difficult to reuse. Peer participation in an IEPD review group also increases individual knowledge of information exchanges and will promote accountability in verifying that the exchange meets established requirements.

Basic checks

- The readme contains documentation necessary to effectively describe the information exchange
- The IEPD Catalog (XML file)
    - validates against the IEPD catalog schema, defined by the IEPD Specification
    - resides in the root directory of the IEPD bearing the file name "iepd-catalog.xml"
- The change log
    - is included in the root directory of the IEPD
    - has a file name that begins with "changelog"
    - records the change history from previous IEPDs
- Extension schemas and sample instances validate
- Extension schemas conform to the NIEM Naming and Design Rules
    - *See Extension schema - Checking conformance for more*

IEPD review checklist

While the NIEM Naming and Design Rules (NDR) provides conformance rules and guidances for NIEM schemas and instance documents, it does not address IEPDs. NDR conformance is only part of the requirement for full NIEM IEPD conformance.

The **NIEM IEPD Specification** provides rules and guidance about the IEPD as a whole, including which artifacts are required and what kind of metadata should be provided.

Checking IEPD conformance is an additional review step that must be taken after the IEPD is fully assembled. Thus, it is not one of the validation steps described in the previous *Build and Validate* phase of the IEPD lifecycle.

IEPD conformance rules are not automated. ConTesA only checks NDR Schematron rules for schemas and instances. IEPD conformance rules must be checked manually.

To make it easier to identify rules from the IEPD specification and record the results, an IEPD review checklist has been provided. This is a spreadsheet that includes:

- Rule summaries from the IEPD specification and information about to evaluate them
- Additional QA tests that can be useful in assessing an IEPD
- Columns to capture the manual review results and additional comments or feedback

Download an IEPD review checklist

- IEPD 5.0 review checklist - for IEPDs using NIEM 5.0 or a more recent release

- MPD 3.0 review checklist - for IEPDs using a release between NIEM 3.0 and NIEM 4.2

Checklists last updated 2021-06-25.

The image below shows several rows from the checklist, along with some example rule status codes filled in.

Rule	Target	Files	Summary
3-1	IEPD	iepd-catalog.xml	**IEPD has IEPD conformance target** The IEPD catalog attribute c:IEPDCatalog/@c:iepdConformanceTargetIdent include the value http://reference.niem.gov/niem/specification/iepd/5.0
3-2	WF-IEPD	iepd-catalog.xml	**IEPD has WF-IEPD conformance target** The IEPD catalog attribute c:IEPDCatalog/@c:iepdConformanceTargetIdent include the value http://reference.niem.gov/niem/specification/iepd/5.0
4-1	Schema-subset	NIEM subset schemas	**NIEM subset is valid** - Subset schemas pass XML schema validation check (can use XMLSpy, Oxyg - Briefly do a manual scan of files to look for file names, components, or sy appear in the full NIEM release. Subsets cannot add or make breaking char content.
5-1	WF-IEPD	iepd-catalog.xml	**IEPD catalog in root directory** The root directory of the IEPD must contain a file named iepd-catalog.xml.
5-2	IEPD-catalog	iepd-catalog.xml	**IEPD catalog is valid** The IEPD catalog (iepd-catalog.xml) must be valid against the IEPD catalog https://reference.niem.gov/niem/specification/model-package-descripti XMLSpy, Oxygen, Xerces, etc)

"Pass" and "fail" status codes are fairly standard for results. "Review" is provided as an additional option to make it easy to identify rows that need to be returned to later for further review.

Peer review

Some organizations use an IEPD Peer Review Process to establish a standard process for reviewing IEPDs.

Submit

IEPD is provided to the reviewing group for review

Preliminary Review

IEPD developer briefs the reviewing group on the IEPD

Detailed Review

IEPD is critically assessed by member(s) of the reviewing group
IEPD developer provides assistance to the review process, if necessary
IEPD is reviewed against business requirements and NIEM specifications

Accept

The IEPD is updated as needed and finally accepted by the review group

During this review process, IEPD developer responsibilities include:

- Review the IEPD Package prior to submission to check for conformance
- Submit IEPD to the governing authority for review
- Provide brief overview of IEPD to the governing authority, if necessary, to answer any questions and clear any concerns with the submission of the package.
- Coordinate with review team to include all recommended changes that are agreed upon in the revised IEPD
- Submit revised IEPD for reviewing authority, if applicable

Responsibilities for the review team include:

- Review IEPDs in accordance with organizational guidelines
- Coordinate with IEPD developer(s) during the review process to incorporate all feedback into the document as necessary
- Coordinate status updates to the IEPD developer and best practice briefings for the NIEM community of interest
- Submit the finalized IEPD to exchange stakeholders

Output

By the end of this stage, the IEPD should be complete. The package should contain all required artifacts, including extension and NIEM subset schemas, an IEPD catalog, a readme, a change log, and a conformance assertion. Many other kinds of documentation and artifacts can be included as well. The final package should be zipped and ready for distribution and implementation.

Publish and Implement

Publish the IEPD for search, discovery, and reuse, and implement the IEPD for production.

- Publish
- Implement
    - Security and Privacy Considerations

The search, discovery, and reuse of IEPDs are necessary for effective information exchange across organizations. IEPD publication allows other organizations to search for needed information without having to recreate it.

Publish

Publish an IEPD to all repositories that are relevant to the exchange; the more exposure the better. Many of the NIEM Communities have IEPD repositories where you can search for a package that is suitable or adaptable to your exchange. You may be able to find a home for your exchange with one of them.

Implement

There are many ways to implement NIEM. Two common implementation examples include the following:

Message Queue	An organization can store NIEM-conformant XML instances within a message queue during its response to a stakeholder's information request service.
Web Services	An organization can embed a NIEM conformant schema into a new or existing Web Service and perform an electronic transfer with one or more exchange partners potentially through a Service Oriented Architecture (SOA) environment.

Security and Privacy Considerations

NIEM does not dictate how you handle security and privacy issues. You must use your organization's standards.

You can tag data with security and privacy, however, other technologies are required upon exchange implementation to enforce security and privacy rules.

You also may use metadata to describe specific requirements in regard to information security and the handling of sensitive, privacy-protected information. The metadata allows systems that implement NIEM to automatically enforce rules that govern the use, protection, dissemination, and access controls for data being shared.

IEPD Artifacts

Each IEPD is a package that contains schemas and other supporting files.

- Required artifacts
  - NIEM subset schemas
  - Extension schema
  - Change log
  - Conformance assertion
  - IEPD catalog
  - Readme
  - Sample messages
- Optional artifacts
  - Business rules
  - Code lists
  - Constraint schemas
  - External standards
  - Mapping spreadsheet
  - Wantlist
  - XML catalog
- Recommended folder layout
- Additional details

See the Information Exchange Package Documentation (IEPD) Specification for more information about these artifacts.

Required artifacts

NIEM subset schemas

An IEPD must include either a local copy of the schemas from a NIEM release or a custom subset of that NIEM release. Because NIEM is a large data model, a subset is usually much easier to deal with in an IEPD. A subset should include only the properties and types from the release that are actually needed by the IEPD, plus any dependencies.

> The SSGT can be used to generate a NIEM XML schema subset.

See the Subset schemas page in this section for more details.

Extension schema

An IEPD will include one or more extension schemas. These are schemas created by the IEPD developer in order to define properties and types needed by the IEPD that do not already exist in NIEM.

The NIEM Naming and Design Rules (NDR) specification provides rules and guidance on how components for both release schemas and IEPD extension schemas should be modeled. in Reference / Concepts section on this site explains some of Artifacts section for more.that information.

Many of the rules from the NDR are defined in Schematron so that conformance testing can be automated against NIEM XML schemas. ConTesA and Oxygen are both capable of evaluating these NDR Schematron rules.

- ConTesA is free to use and will generate a conformance report that can be included in the IEPD.
- Oxygen requires a paid license but may be easier to use frequently during schema development because Schematron errors are checked and displayed similarly to standard XML validation errors.

See the Extension schema page in this section for more details.

Change log

A change log can be a summary or a detailed description of the changes in this IEPD since the previous versions. NIEM release provide change logs in the form of a spreadsheet but free text descriptions in a document or text file are fine.

Conformance assertion

A conformance assertion is a document that declares the IEPD conforms to relevant NIEM specifications and associated rules. Although it increases the level of confidence that an IEPD was checked for NIEM conformance and quality, it does NOT constitute a guarantee or contract because it can be self-asserted.

Inclusion of a conformance assertion made by a reputable, independent, trusted entity (person or organization) can increase confidence in conformance, especially if it includes information such as a formal conformance test report or similar artifact.

A conformance assertion can also contain information about any known issues in the IEPD and the reasons for which they may have been necessary.

As part of a conformance assertion, additional artifacts may also be included in the IEPD:

Conformance report
: ConTesA can be used to check if IEPD extension schemas conform to the NIEM Naming and Design Rules (NDR) specification and provides the results in a conformance report.

IEPD review checklist
: Manually review rules about the full IEPD package from the IEPD Specification and document results with the IEPD review checklist spreadsheet.

IEPD catalog

An IEPD Catalog is an XML document that contains basic information about the package (name, description, purpose, point of contact, etc.) and a listing of the package's key artifacts. This file has the name `iepd-catalog.xml` and resides in the IEPD root directory.

See the IEPD Catalog page in this section for more details.

Readme

A readme artifact is mandatory for all IEPDs. Additional documentation can be included in an IEPD as desired, but a readme will serve as the common starting point and is the required minimum for human-readable documentation.

A readme should include information that describes the following:

- Purpose of the IEPD
- Scope of its deployment, usage, and information content
- Business value and rationale for developing it
- Type of information it is intended to exchange, in business terms
- Identification (or types) of senders and receivers
- Typical interactions between senders, receivers, and systems
- References to other documentation within the IEPD
- Links to external documents that may be needed to understand and implement the IEPD

The readme does not have a required file format. Word documents, text, and markdown are common formats.

Sample messages

A sample message (also known as sample instances or sample IEPs) is an example of an actual message that will be exchanged based on the IEPD. The data should be realistic and the sample message should clearly illustrate to IEPD implementers what the real-world messages should look like.

More than one kind of message can be defined within a single IEPD. At least one sample message should be provided for each.

Sample messages in an IEPD also provide the IEPD developer with the opportunity to better test the IEPD schemas that are being developed. Confirming that sample messages can carry all of the expected data from the data requirements helps to ensure that the schemas are wired up correctly and that little things haven't been forgotten, like missing substitutions, incomplete subsets, or properties that haven't been added to the right types.

Optional artifacts

Aside from the required artifacts, IEPD content is relatively flexible. A variety of other optional documentation files may be incorporated into an IEPD. When applicable, these may include (but are not limited to) files that describe or explain:

- Implementation details (hardware, software, configuration, etc.)
- Use of multiple root elements
- Use of multiple subsets or mixed releases
- How to use/reuse an IEPD for various purposes (such as Web Services)
- Rationales and/or business purposes

Business rules

Business rules are defined in artifacts with informal text or formally coded machine-readable statements that describe business policy or procedure or special data constraints that may be difficult or impossible to specify in schema. Schematron is the recommended format for specifying additional data constraints on XML messages. Informally written rules may be very difficult or impossible to validate but allow the author to present them in the IEPD.

Code lists

NIEM supports enhanced code lists in addition to typical schema enumerations. These code lists may be static (defined in the IEPD) or dynamic (defined at a URL and potentially updated after the IEPD is locked and published) and can be defined as CSVs or Genericode.

See Code Lists in the Reference / Artifacts section for more.

Constraint schemas

A constraint schema is a schema that usually starts from a subset schema and is modified by hand in order to add custom-tailored constraints and restrictions on instance messages that go beyond setting cardinality limits.

Some things in a constraint schema must remain the same in order to maintain fidelity with the original schemas. For example, property names and the order in which they appear cannot be changed because this would affect the tags in the instances. However, because type information usually does not appear in an instance, properties can be rewritten with customized local types.

- Properties with type `nc:TextType` or `niem-xs:string` can be rewritten to have custom string types with individual maximum character lengths or patterns.
- Properties that leverage the same type but need to use it in different ways can be rewritten. For example, if a message contains information about a child and other adults, these properties would all leverage the same type `nc:PersonType` in the NIEM release or subset schemas and `nc:PersonType` would include the superset of all of the person-related requirements. In a constraint schema, however, the child property could be rewritten so that it only includes the exact properties that are allowed to be shared (perhaps like first name and age only) while the other kinds of persons in the message could be defined with additional fields.

The use of constraint schemas requires instance validation to be performed in two passes:

- To ensure NIEM conformance, IEPs must be valid against the original set of NIEM subset and/or IEPD extension schemas.
- To ensure conformance to the additional set of rules, IEPs must also be valid against the constraint schemas.

XML catalogs can be used to assign different file locations to schema namespaces for the separate validation passes.

Constraint schemas do not provide clear visibility or explanation of the constraints they enforce; nor do they provide clear validation failure messages. Thus the use of constraint schemas is discouraged. A

generally better way to impose additional rules on an IEP is to use Schematron. It provides facilities for better understanding of the business rules, their intent, and error handling of failures. Constraint schemas do remain an option, however, that may be useful for situations where the XML schemas themselves are required to be tightened down as much as possible.

External standards

External standards may be included in IEPDs for information that is already commonly shared within the exchange's target community according to a standard defined outside of NIEM. GML is one such external standard already included with NIEM releases, but IEPDs may require others. In these cases, it may not make sense to provide an alternate NIEM-representation of data that already has a well-defined and commonly understood format.

To use an external standard:

- Add the schemas for the external standard to the IEPD. The IEPD specification recommends adding these to a separate `external` folder under the `base-xsd` folder, next to other folders for the NIEM subset and IEPD extension schemas.
- Define adapter types to wrap external properties. An adapter type is a way to identify where the external, non-NIEM conformant properties are in an exchange and enables NIEM NDR rules to avoid throwing errors at these known exceptions.

Mapping spreadsheet

A mapping spreadsheet can be used to explicitly document mappings from data requirements to properties and types in NIEM. It can also be used to model extensions for requirements that do not exist in NIEM.

See the Mapping Spreadsheet page in this section for more details, plus the spreadsheet template and examples.

Wantlist

A wantlist is a custom XML file generated by the SSGT that saves the list of properties and types that were added to the subset. In other words, it describes what an exchange "wants" from the NIEM data model.

If the IEPD includes a NIEM subset and that subset was generated by the SSGT, the wantlist should be included in the IEPD as well. This ensures that the subset can be reloaded in the SSGT for future updates without having to start the subset over.

XML catalog

XML catalog files can be used to specify or override the file locations associated with schema namespaces.

See XML Catalog in the Reference / Artifacts section for more.Artifacts section for more.

Recommended folder layout

The IEPD Specification does not contain normative rules on how to structure the folders and place the files in an IEPD, but guidance is provided. A recommended layout is provided below:

> See **Appendix E. Guidance for IEPD Directories (non-normative)** from the IEPD Specification for more information and helpful tips on how to organize an IEPD.

```
myIEPD-2.0/

  iepd-catalog.xml          # IEPD catalog
  changelog.*               # Change log
  conformance-assertion.*   # Conformance assertion
  readme.*                  # Readme

  base-xsd/                 # Schema folder

    niem/                   # NIEM subset schemas
      xsd/
        adapters/
        codes/
        domains/
        external/
        utility/
        niem-core.xsd
        wantlist.xml        # SSGT wantlist
        xml-catalog.xml

    extension/              # IEPD extension schemas
      extension1.xsd
      extension2.xsd
      ...
      xml-catalog.xml

    external/               # Schemas from external standards
      ic-ism/
      ...
      xml-catalog.xml

  iep-sample/               # Sample messages
    message1.xml
    message2.xml

  documentation/            # Human-readable documentation
    ...
```

Additional details

For more details on some of the artifacts described above, please see the following pages in this section:

- Subset Schemas

  A subset schema is a customized version of a NIEM schema that contains only the properties, types, and codes that are needed for a particular information exchange, plus any of their required dependencies.

- Extension Schema

  In an extension schema, an IEPD developer can create new NIEM-conformant properties and types to represent message data requirements that are not available in a NIEM release. Extensions schemas are used in combination with NIEM subset schemas to define the structure and meaning of a message.

- IEPD Catalog

  An IEPD Catalog is an XML document that contains basic information about the package (name, description, purpose, etc.) and a listing of the package's key artifacts. It is a required artifact in an IEPD.

- Conformance Report

  A conformance report specifies how and to what degree the IEPD is NIEM-conformant.

- Mapping Spreadsheet

  A mapping spreadsheet can be used to explicitly document mappings from data requirements to properties and types in NIEM. It can also be used to model extensions for requirements that do not exist in NIEM.

Subset Schemas

A subset schema is a customized version of a NIEM schema that contains only the properties, types, and codes that are needed for a particular information exchange, plus any of their required dependencies.

- Size
- Subsets and IEPDs
- Generating a Subset
- Example subset of a NIEM type
- Valid Subset Operations
    - Limit cardinality
    - Remove components and items
    - Other constraints

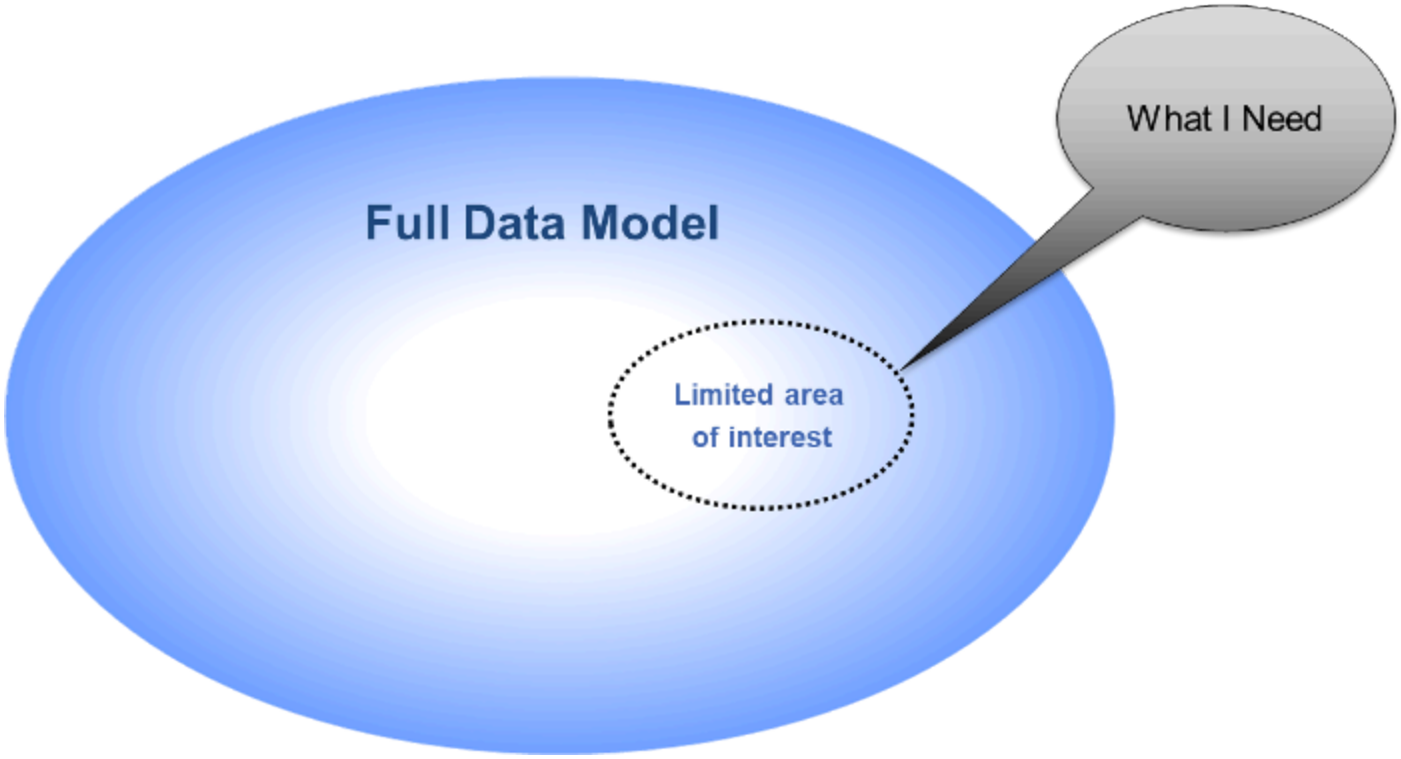
The 4.0 NIEM release includes 11,000 NIEM elements and over 150 files (the external standards, including GML, contribute to the file count). While its possible to use a full NIEM release package in an IEPD, it is typical for an IEPD developer to pick and choose only those release components that are needed by that exchange and to generate a much smaller subset of the release.

> A subset schema must still conform to the NDR and cannot allow any content that is not permitted by the full corresponding NIEM schema.

The collection of individual subset schemas for a release are typically referred to as a NIEM subset. A subset will likely be much smaller than the corresponding full release.

Size

The size of a NIEM subset will vary based on how much of the NIEM model is being reused, but a subset can easily be narrowed down to a couple dozen or a couple hundred components across a dozen files. The reduced size and scope of a subset should improve validation and tool performance. It also makes the IEPD much easier to understand for other users.



Subsets and IEPDs

IEPDs are self-contained packages, meaning that each one should contain all the files and information needed for implementation. To reuse components from a NIEM release, the IEPD needs to either contain that full set of release schemas or a subset of that release in order to provide the source of those NIEM components.

In an IEPD package, NIEM XML schemas are typically included in the `base-xsd` folder. The NIEM release or NIEM subset would be included in the `niem` folder.

```
# Partial IEPD package directory

iepd-name/
  base-xsd/
    niem/
      codes/
      domains/
      niem-core/
      utility/
    extension/
```

Extension schemas, which will contain user-created properties and types to represent requirements not found in NIEM, will be included in the `extension` folder. These extension schemas will import and reuse NIEM components form the `niem` folder.

If you are developing multiple related IEPDs, you can choose to build a custom subset for each IEPD or reuse a single combined subset that represents the full set of requirements in each IEPD.

Generating a Subset

Creating a subset manually can be time-consuming and error-prone. The **Schema Subset Generation Tool (SSGT)** can be used to search the NIEM data model, select components for a subset, calculate dependencies, and generate subset XML schemas.

Example subset of a NIEM type

The following shows the full **`PersonNameType`** from the 4.0 Core namespace:

```
<xs:complexType name="PersonNameType">
  <xs:annotation>
    <xs:documentation>A data type for a combination of names and/or titles b
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      <xs:sequence>
        <xs:element ref="nc:PersonNamePrefixText" minOccurs="0" maxOccurs="u
        <xs:element ref="nc:PersonGivenName" minOccurs="0" maxOccurs="unbour
        <xs:element ref="nc:PersonMiddleName" minOccurs="0" maxOccurs="unbou
        <xs:element ref="nc:PersonSurName" minOccurs="0" maxOccurs="unbounde
        <xs:element ref="nc:PersonNameSuffixText" minOccurs="0" maxOccurs="u
        <xs:element ref="nc:PersonMaidenName" minOccurs="0" maxOccurs="unbou
        <xs:element ref="nc:PersonFullName" minOccurs="0" maxOccurs="unbound
        <xs:element ref="nc:PersonNameCategoryAbstract" minOccurs="0" maxOcc
        <xs:element ref="nc:PersonNameSalutationText" minOccurs="0" maxOccur
        <xs:element ref="nc:PersonOfficialGivenName" minOccurs="0" maxOccurs
        <xs:element ref="nc:PersonPreferredName" minOccurs="0" maxOccurs="ur
        <xs:element ref="nc:PersonSurNamePrefixText" minOccurs="0" maxOccurs
        <xs:element ref="nc:PersonNameAugmentationPoint" minOccurs="0" maxOc
      </xs:sequence>
      <xs:attribute ref="nc:personNameCommentText" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

The following shows a subset of this type, generated by the SSGT, with only the given name, middle name, and surname included. The cardinality has also been restricted.

```
<xs:complexType name="PersonNameType">
  <xs:annotation>
    <xs:documentation>A data type for a combination of names and/or titles b
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="structures:ObjectType">
      <xs:sequence>
        <xs:element ref="nc:PersonGivenName" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="nc:PersonMiddleName" minOccurs="0" maxOccurs="unbou
```

```
        <xs:element ref="nc:PersonSurName" minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Valid Subset Operations

The following describes a set of operations to consider when constructing subset schemas. It is possible to apply them in combinations that could break the subset relationship, or even result in invalid schemas. Apply these operations carefully and thoughtfully.

Limit cardinality

NIEM adopts an optional and over-inclusive strategy in order to support a broad user base with very different sets of needs. A subset for an IEPD represents a single set of exchange requirements and is the right place to tailor the cardinality as needed.

Cardinality can be adjusted as long as the new cardinality is permitted by the original release cardinality.

> If an element in a type has cardinality `(1, unbounded)`, then the following are examples of valid and invalid adjustments in a subset to the original cardinality.
>
> These cardinalities fall within the original cardinality range:
>
> - (1, 1)
> - (2, 10)
>
> These cardinalities fall outside the original cardinality range (0 was not permitted):
>
> - (0, 1)
> - (0, unbounded)

Element options:

- Increase the value of an `xs:element/@minOccurs` as long as it remains less than or equal to its corresponding `xs:element/@maxOccurs` defined in the original schema.
- Decrease the value of an `xs:element/@maxOccurs` as long as it remains greater than or equal to its corresponding `xs:element/@minOccurs` defined in the original schema.
- Remove an `xs:element` with cardinality (`xs:element/@minOccurs="0"`) from its type.

Attribute options:

- Change an `xs:attribute/@use="optional"` to `"required"`.
- Change an `xs:attribute/@use="optional"` to `"prohibited"`.
- Remove the reference of an `xs:attribute` with `@use="optional"` from a type.

Remove components and items

Attributes, elements, and types can be removed from a subset if they are not being used by or required by other components in the subset.

- Remove an unused `xs:element` declaration.
- Remove an unused `xs:complexType` or `xs:simpleType` declaration.
- Remove an unused element with representation term *AugmentationPoint* (these are specifically required for REF schemas in the NDR but can be removed from a subset).
- Remove an unused `xs:import` statement.
- Remove an unused file.
- Remove a comment.
- Remove an `xs:annotation` (definitions can be removed from a subset).
- Remove an `xs:enumeration` from an `xs:simpleType` as long as it is not the only remaining `xs:enumeration`. (Removing the last enumeration would change the type to free text instead of a list of codes, which would not be permitted by the original schema.)

Other constraints

- Add or apply a constraining facet to an `xs:simpleType`.
- Change a concrete (non-abstract) `xs:element` declaration to `xs:element/@abstract="true"`.
- Change an `xs:element/@nillable="true"` to `xs:element/@nillable="false"`.
- Substitute one or more `xs:element/@substitutionGroup` members for its associated substitution group head.
- Replace a wildcard with a composition, i.e. an ordered sequence of elements.

Extension Schema

In an extension schema, an IEPD developer can create new NIEM-conformant properties and types to represent message data requirements that are not available in a NIEM release. Extensions schemas are used in combination with NIEM subset schemas to define the structure and meaning of a message.

- Conformance Targets
- Extension schemas in an IEPD
- Reusing NIEM components
- Checking Conformance
- NIEM JSON and Extension Schemas

> The NIEM Naming and Design Rules (NDR) Specification defines what an extension schema document (EXT) is and the set of rules it must follow in order to be considered NIEM-conformant.

New components added to extension schemas are often referred to as "extensions". These extensions may augment, extend, substitute for, or reuse components from NIEM; or they may simply follow the NIEM syntax rules if no reuse is possible.

> See the Modeling Concepts section on this site to learn more about setting up a NIEM-conformant schema, how to add basic properties and types, and how and when to use special NIEM techniques. This material also includes XML and JSON examples and templates.

Conformance Targets

An XML extension schema must declare an ExtensionSchemaDocument conformance target as defined by the NDR.

- The snippet below shows a very general outline of an XML extension schema that has declared its conformance target to be an extension schema document as defined by the NDR, version 5.0:
- Notice that the end of the conformance target identifier in the example is formatted as
  - `/specificationName/version/conformanceTarget`:
  - `/naming-and-design-rules/5.0/#ExtensionSchemaDocument`

```
<xs:schema
  ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-a
<xs:schema>
```

See the page about the Conformance Target Attribute Specification for more details about how to declare a conformance target attribute, additional schema declarations that are required to support this attribute, and other conformance target values.

Extension schemas in an IEPD

An IEPD package might be arranged in the following way, with the `base-xsd` folder containing three sub-folders for:

- the SSGT-generated NIEM subset schemas
- the developer-created NIEM extension schemas
- any external standards that do not follow NIEM conformance rules

```
myIEPD-2.0/

  base-xsd/

    niem/                # NIEM subset schemas
      xsd/
        adapters/
        codes/
        domains/
        external/
        niem-core.xsd
        ...

    extension/           # NIEM-conformant extension schemas
      extension1.xsd
      extension2.xsd
      ...
      xml-catalog.xml

    external/            # External standards that are not NIEM-conformant
      ic-ism/
      ...
```

The Information Exchange Package Documentation (IEPD) Specification page on this site contains more information about how an IEPD package might be structured.

Reusing NIEM components

In order to reuse components from NIEM, the following steps should be taken:

- Add a NIEM subset to your IEPD package by using the Schema Subset Generator Tool (SSGT).
- Import the relevant NIEM schemas into your extension schema.
- Declare namespace prefixes for the imported NIEM schemas. If possible, try to use reuse the same namespace prefixes that NIEM uses for better consistency and clarity for other users.
- Reference components from the NIEM schemas. Do not duplicate NIEM properties and types.

- This simplified example of an extension schema shows how to reuse content from NIEM.
- This snippet assumes that the schemas are located in the same relative directories as the sample IEPD package layout above.
- Note that this snippet is incomplete and only shows the parts of a schema that are relevant to reusing NIEM components. It is not complete enough to be valid or NIEM-conformant.

```
<!-- Declare a prefix for the NIEM Core namespace -->
<xs:schema
  xmlns:nc="http://release.niem.gov/niem/niem-core/5.0/"
```

```
>
  <!-- Import the NIEM Core namespace -->
  <xs:import schemaLocation="../niem/niem-core/5.0/niem-core.xsd"
             namespace="http://release.niem.gov/niem/niem-core/5.0/"/>

  <!-- Reuse a NIEM Core type -->
  <xs:element name="Veterinarian" type="nc:PersonType">
    <xs:annotation>
      <xs:documentation>
        A professional who practices veterinary medicine by treating diseases,
      </xs:documentation>
    </xs:annotation>
  </xs:element>

  <!-- Reuse a NIEM Core property (see the element reference below) -->
  <xs:type name="PatientFileType">
    <xs:annotation>
      <xs:documentation>
        A data type for a medical file containing a record health and treatmen
      </xs:documentation>
    </xs:annotation>
    <xs:complexContent>
      <xs:extension base="structures:ObjectType">
        <xs:sequence>
          <!-- NIEM Core element reference -->
          <xs:element ref="nc:Address">
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:type>

</xs:schema>
```

Checking Conformance

There are several steps to follow when checking the conformance of an XML extension schema:

Validate XML Schema

Make sure the extension schema is a valid XML Schema. Tools like XMLSpy, Oxygen XML Editor, xmllint, and Xerces can run this validation step.

Declare conformance targets

Make sure the schema declares a `ct:conformanceTargets` attribute, with the appropriate value as defined by the NDR.

Check NDR conformance for automated rules

Make sure the schema is conformant to whatever conformance target(s) are declared in the previous step.

Use the Conformance Testing Assistant (ConTesA) or Schematron validation in Oxygen XML Editor to check the extension schema against all automated NDR rules.

> ConTesA cannot check extension schema conformance without the conformance target attribute specified. This attribute tells ConTesA which rule set to use for validation. Without this information, ConTesA will not run any tests.

Check NDR conformance for manual rules

Review the extension schema manually against the NDR free-text rules to ensure they are met.

NIEM JSON and Extension Schemas

In NIEM JSON, it can be easier to define properties and types from different namespaces together in the same JSON schema. This is because the NIEM conceptual model uses techniques like element substitution and type extension, which do not have the same counterparts in JSON schema. Because of this, JSON extensions can be added to the same JSON schema that defines the NIEM subset for an IEPD and a separate JSON extension schema is not necessary.

There is currently no automated rule assertions for NIEM JSON schemas. The guidance for now is to follow the principles and intents of the NDR rules as closely as possible.

IEPD Catalog

An IEPD Catalog is an XML document that contains basic information about the package (name, description, purpose, etc.) and a listing of the package's key artifacts. It is a required artifact in an IEPD.

- Conformance
- Example
- IEPD Specification references

The IEPD Catalog establishes identification metadata, conformance targets, purpose, general content, lineage, and other metadata. Metadata includes:

- Uniform Resource Identifier (URI)
- Name
- Version number
- The conformance target identifier
- A copy of all schema documents needed to validate any instance document class it defines
- (Optional) Alternate representations in addition to XML Schema (e.g., generic diagram, UML/XMI, database format, spreadsheet, etc.)
- (Optional) Miscellaneous other documentation or file artifacts for assisting with usage or implementation

Conformance

The IEPD Specification defines a IEPD Catalog as a conformance target, with a set of rules that must be followed in order to be considered conformant.

> The first step is to make sure that the catalog file validates against its **IEPD Catalog XML Schema**.

Example

The following is an example IEPD catalog from the Cursor on Target IEPD, provided by the IEPD specification.

```
<?xml version="1.0" encoding="UTF-8"?>
<c:IEPDCatalog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://reference.niem.gov/niem/resource/iepd/catalog/5
    xmlns:c="http://reference.niem.gov/niem/resource/iepd/catalog/5.0/"
    xmlns:nc="http://release.niem.gov/niem/niem-core/5.0/"
    xmlns:structures="http://release.niem.gov/niem/structures/5.0/"
    xmlns:cot="http://example.com/cot-niem/0.9/" c:iepdURI="http://example.com
    c:iepdConformanceTargetIdentifierURIList="http://reference.niem.gov/niem/s
    c:iepdName="cot-iepd" c:iepdVersionID="0.9rc1">
    <c:IEPDInformation>
        <c:AuthoritativeSource>
            <nc:EntityOrganization>
```

```
                <nc:OrganizationName>CoT Program Office</nc:OrganizationName>
                <nc:OrganizationPrimaryContactInformation>
                    <nc:ContactWebsiteURI> https://partners.mitre.org/sites/Co
                    </nc:ContactWebsiteURI>
                </nc:OrganizationPrimaryContactInformation>
            </nc:EntityOrganization>
        </c:AuthoritativeSource>
        <c:CreationDate>2014-08-04</c:CreationDate>
        <c:StatusText>Release Candidate</c:StatusText>
    </c:IEPDInformation>
    <c:IEPConformanceTarget structures:id="CoT-NIEM-IEP">
        <nc:DescriptionText>
                An IEP class equivalent to Cursor-on-Target 2.0 messages
            </nc:DescriptionText>
        <c:HasDocumentElement c:qualifiedNameList="cot:Event"/>
        <c:XMLSchemaValid>
            <c:XMLCatalog c:pathURI="base-xsd/xml-catalog.xml"/>
        </c:XMLSchemaValid>
        <c:SchematronValid>
            <c:SchematronSchema c:pathURI="schematron/business-rules.sch"/>
        </c:SchematronValid>
        <c:EXIXMLSchema>
            <c:XMLCatalog c:pathURI="exi-xsd/exi-xml-catalog.xml"/>
            <c:XMLSchemaDocument c:pathURI="extension/cot-niem.xsd"/>
        </c:EXIXMLSchema>
        <c:IEPSampleXMLDocument c:pathURI="iep-samples/iep1.xml"/>
    </c:IEPConformanceTarget>
    <c:ReadMe c:pathURI="00-README.txt"/>
    <c:IEPDChangeLog c:pathURI="01-changelog.txt"/>
    <c:ConformanceAssertion c:pathURI="02-conformance.txt"/>
    <c:Wantlist c:pathURI="base-xsd/niem/wantlist.xml"/>
    <c:ExtensionSchemaDocument c:pathURI="base-xsd/extension/cot-niem.xsd"/>
    <c:ExtensionSchemaDocument c:pathURI="base-xsd/extension/cot-niem-codes.xs
    <c:ReferenceSchemaDocument c:pathURI="base-xsd/extension/milops-future-ref
</c:IEPDCatalog>
```

IEPD Specification references

See the following sections from the `IEPD Specification` for more information about the catalog file:

- 5.1. NIEM IEPD Catalog
- 5.2 Metadata Concepts
- Appendix A. IEPD Catalog XML Schema Document
- Appendix B. Example IEPD Catalog Document for Cursor on Target

Conformance Report

A conformance report specifies how and to what degree the IEPD is NIEM-conformant.

- ConTesA Report
- Additional validation steps

ConTesA Report

The Conformance Testing Assistance (ConTesA) checks uploaded schemas against Schematron rules from the Naming and Design Rules (NDR).

- The images below show portions of the spreadsheet version of the conformance report generated by `ConTesA`.
- This first image shows the summary - how many rules were run, how many rules passed, and how many rules failed.

ConTesA Conformance Report

extension.xsd
11/28/2018 13:56:29

Summary - Automatic Checks

Rules that can be run automatically by ConTesA are considered "Automatic Checks". There are two types of "Automatic Checks": "Auto Passed" and "Auto Failed". "Auto Passed" rules result in a status of "Success" or "Information". "Auto Failed" rules result in a status of "Warning", "Failure",

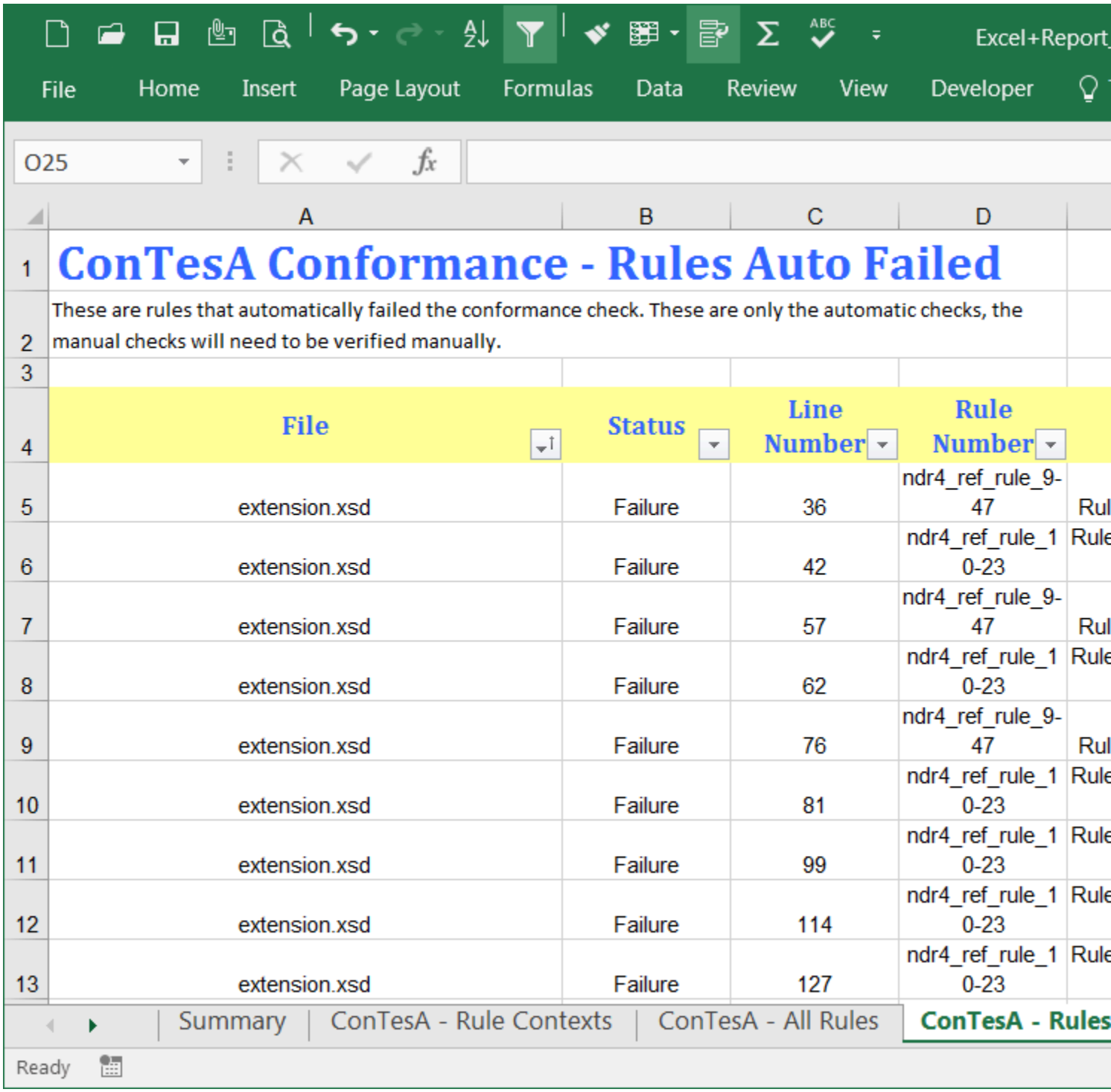
ConTesA Conformance (v3.0)

Total Number Rules in ConTesA The number of rules that are listed in the ConTesA version 3.0.	170	
Total Number Rules Auto Checked The number of rules that are automatically checked by the tool. The remainder of the rules will need to be verified manaully (see the "Manual Checks" section below).	170	(100%)
Total Number Rules Auto Passed The number of rules that automatically passed validation. A rule is considered to pass validation when it returns a status of "Success" or "Information".	164	(96%)
Total Number Rules Auto Failed The number of *different* rules that failed validation. Each rule may have failed one or more times. See the sheet "ConTesA - Rules Auto Failed" for specific locations of the failures in your schema file(s). A rule is considered to fail validation when it returns a status of "Warning", "Failure", or "Error".	6	(4%)

Summary | ConTesA - Rule Contexts | ConTesA - All Rules | ConTesA - Rules

Ready

- This image shows the NDR rules that failed and the schema line numbers where it occurred.



ConTesA Conformance - Rules Auto Failed

These are rules that automatically failed the conformance check. These are only the automatic checks, the manual checks will need to be verified manually.

File	Status	Line Number	Rule Number
extension.xsd	Failure	36	ndr4_ref_rule_9-47
extension.xsd	Failure	42	ndr4_ref_rule_10-23
extension.xsd	Failure	57	ndr4_ref_rule_9-47
extension.xsd	Failure	62	ndr4_ref_rule_10-23
extension.xsd	Failure	76	ndr4_ref_rule_9-47
extension.xsd	Failure	81	ndr4_ref_rule_10-23
extension.xsd	Failure	99	ndr4_ref_rule_10-23
extension.xsd	Failure	114	ndr4_ref_rule_10-23
extension.xsd	Failure	127	ndr4_ref_rule_10-23

Additional validation steps

Additional information can also be included specifying any additional steps that were taken to manually review and validate rules that are not automated, like free-text rules from the Naming and Design Rules (NDR) and the Information Exchange Package Documentation (IEPD) Specification. See the IEPD Specification, Appendix D. Conformance Assertion Example for more on what this might look like.

Mapping Spreadsheet

A mapping spreadsheet can be used to explicitly document mappings from data requirements to properties and types in NIEM. It can also be used to model extensions for requirements that do not exist in NIEM.

- Mapping examples
  - Data requirements
  - Simple mappings
  - Detailed mappings
- Modeling examples
  - Property tab
  - Additional tabs

This is the same template that can be used to submit change requests for NIEM release content. There are additional tabs and columns to support this level of detail, but to just use this spreadsheet for mappings, focus on the "Property" tab. Customize the spreadsheet by hiding column and tabs that aren't being used.

Download the **template** and **example spreadsheet** - *last updated 2021-07-19*.

Mapping examples

Data requirements

The image below shows some examples of how data requirements can be captured in the spreadsheet.

Property Name	Data Type	Definition
award	Text	An award won by or for this item. Supersedes awards.
birthDate	Date	Date of birth.
birthPlace	Place	The place where the person was born.
email	Text	Email address.
familyName	Text	Family name. In the U.S., the last name of an Person. This can be used along with givenName instead of the name property.
givenName	Text	Given name. In the U.S., the first name of a Person. This can be used along with familyName instead of the name property.
netWorth	MonetaryAmount or PriceSpecification	The total financial value of the person as calculated by subtracting assets from liabilities.

Simple mappings

The image below shows how the data requirements

Property Name	Data Type	Definition	Mapping Code	Notes
award	Text	An award won by or for this item. Supersedes awards.	no match	
birthDate	Date	Date of birth.	map	
birthPlace	Place	The place where the person was born.	map	
email	Text	Email address.	map	
familyName	Text	Family name. In the U.S., the last name of an Person. This can be used along with givenName instead of the name property.	map	
familyName	Text	Family name. In the U.S., the last name of an Person. This can be used along with givenName instead of the name property.	map	
givenName	Text	Given name. In the U.S., the first name of a Person. This can be used along with familyName instead of the name property.	map	
netWorth	MonetaryAmount or PriceSpecification	The total financial value of the person as calculated by subtracting assets from liabilities.	no match	
netWorth	MonetaryAmount or PriceSpecification	The total financial value of the person as calculated by subtracting assets from liabilities.	map	
netWorth	MonetaryAmount or PriceSpecification	The total financial value of the person as calculated by subtracting assets from liabilities.	map	
netWorth	MonetaryAmount or PriceSpecification	The total financial value of the person as calculated by subtracting assets from liabilities.	map	

- It may be necessary to map one data requirement to multiple properties in NIEM. Repeat the data requirement information if necessary.

Detailed mappings

While not required, if it's helpful for review, include definition and type information for NIEM properties.

Property Name	Data Type	Definition	Mapping Code	Notes
award	Text	An award won by or for this item. Supersedes awards.	no match	
birthDate	Date	Date of birth.	map	
birthPlace	Place	The place where the person was born.	map	
email	Text	Email address.	map	
familyName	Text	Family name. In the U.S., the last name of an Person. This can be used along with givenName instead of the name property.	map	
familyName	Text	Family name. In the U.S., the last name of an Person. This can be used along with givenName instead of the name property.	map	
givenName	Text	Given name. In the U.S., the first name of a Person. This can be used along with familyName instead of the name property.	map	
netWorth	MonetaryAmount or PriceSpecification	The total financial value of the person as calculated by subtracting assets from liabilities.	no match	
netWorth	MonetaryAmount or PriceSpecification	The total financial value of the person as calculated by subtracting assets from liabilities.	map	
netWorth	MonetaryAmount or PriceSpecification	The total financial value of the person as calculated by subtracting assets from liabilities.	map	
netWorth	MonetaryAmount or PriceSpecification	The total financial value of the person as calculated by subtracting assets from liabilities.	map	

Modeling examples

In addition to documenting whether or not local requirements map to NIEM, it's also possible to use the spreadsheet to model the extensions.

Property tab

For data requirements that do not map to NIEM ("no match"), fill in the following fields:

Property name

- Add a NIEM-conformant property name.
- See Property – Naming for naming tips.

Property type ("Qualified Data Type")

- Reuse an existing type from NIEM if one is suitable
- Add a new name if a new extension type will need to be created.
- Make sure that the type name starts with a namespace prefix so the source of the type can be

identified.

Property definition

- Add a NIEM-conformant property definition.
- See Property – Definitions for definition tips.

Mapping code

- Change the mapping code from “no match” to “add” once the extension has been modeled.

The image below shows how two extension properties will be modeled in a forthcoming extension schema. They are in bold text and highlighted in yellow to stand out here, but do not have to be formatted differently in the actual spreadsheet itself - the “add” mapping code and the different namespace prefix will distinguish these rows from the properties that actually exist in NIEM.

Property Name	Data Type	Definition	Mapping Code	Notes
award	Text	An award won by or for this item. Supersedes awards.	**add**	
birthDate	Date	Date of birth.	map	
birthPlace	Place	The place where the person was born.	map	
email	Text	Email address.	map	
familyName	Text	Family name. In the U.S., the last name of an Person. This can be used along with givenName instead of the name property.	map	
familyName	Text	Family name. In the U.S., the last name of an Person. This can be used along with givenName instead of the name property.	map	
givenName	Text	Given name. In the U.S., the first name of a Person. This can be used along with familyName instead of the name property.	map	
netWorth	MonetaryAmount or PriceSpecification	The total financial value of the person as calculated by subtracting assets from liabilities.	**add**	
netWorth	MonetaryAmount or PriceSpecification	The total financial value of the person as calculated by subtracting assets from liabilities.	map	
netWorth	MonetaryAmount or PriceSpecification	The total financial value of the person as calculated by subtracting assets from liabilities.	map	
netWorth	MonetaryAmount or PriceSpecification	The total financial value of the person as calculated by subtracting assets from liabilities.	map	

Additional tabs

Additional tabs can be used to model extensions:

- **Type tab** - Add basic information about types, including names, namespace prefixes, definitions, styles, and parent or base types.
- **Type-has-property** - Nest properties under a type. For example, for a person name type, the properties that would appear under the type would include first name and last name.
- **Codes | Facets tab** - Add codes and definitions, or other kinds of facets (e.g., patterns, min or max limits on a number). Each facet row should include the namespace prefix and name of the simple type that will define it.
- **Namespace tab** - Add basic information about new namespaces that will be created, including the namespace prefix, URI, definition, and file name.
- **Local terminology tab** - Add acronyms, abbreviations, or other local terminology and provide either their literals or their definitions.

IEPD Starter Kit

To use NIEM, you develop or implement Information Exchange Package Documentations (IEPDs). An IEPD defines a recurring message in XML and is built to satisfy information exchange business requirements.

A developer builds an IEPD by incorporating the necessary NIEM core and domain model content. The subset schema generation tool (SSGT) can aid in the process of developing this NIEM subset. The developer may also extend that content as needed to account for information requirements that are not yet addressed in NIEM. The IEPD will ultimately define XML instance documents that will contain the information to be exchanged. Extended and new content developed in IEPD extension schema documents should be considered for future model updates. In turn, domain and core model updates will be harmonized and integrated into future NIEM releases. In this way, NIEM evolves with new and changing needs.

There is more information on IEPD construction available at our **NIEM Training** site.

In an era when return on investment has never been more important to government services, NIEM enables organizations to exchange information across all levels of government in a manner that is both effective and efficient.

Be sure to check out the NIEM Cost Model, which allows users to quantify the associated costs of adopting NIEM.

Links

- Search GitHub for NIEM IEPDs
- See a template IEPD on GitHub

See more

- IEPD Series

  Examples designed to demonstrate various unique aspects of the NIEM technical framework.

- Terms

  A list of terms with links to their source definitions across the following technical specifications.

- Reusable XML Snippets

  Reusable XML snippets of the various NIEM XML components. These snippets could also be called patterns or templates.

IEPD Series

The material in this area is designed to demonstrate various unique aspects of the NIEM technical framework. It is referred to as the "Superhero" family of IEPDs because they describe comic book characters. This theme was chosen because most people know who Batman and Superman are, and it allowed us to show the NIEM technical framework in a context independent of the existing NIEM domains.

These examples presume a familiarity with the NIEM v3.0 technical specifications (i.e., MPD, NDR, etc.) and common XML principles (i.e., root element, complex type, simple type, etc.). If you are not, the latest NIEM technical specifications are accessible from links at http://reference.niem.gov, and knowledge of XML may be acquired from http://www.w3schools.com/xml/default.asp and other online services.

SuperHero IEPDs Archive
Download all SuperHero IEPDs as a single zip file.

Super Hero IEPD
IEPD established the parental base for the family by introducing a super hero through a simple extension schema document.

SuperHero Augmentation
Builds upon SuperHero IEPD to demonstrate several ways of implementing the augmentation concept in NIEM.

SuperHero Specialization
Builds upon SuperHero IEPD showing how to create specializations (i.e., xs:extension) in NIEM.

SuperHero Association
Builds upon SuperHero IEPD showing how to create associations in NIEM.

SuperHero Local Terminology
Builds upon SuperHero IEPD showing how to create local terminology in NIEM.

SuperHero ISM
Builds upon SuperHero IEPD showing how to apply the Intelligence Community ISM (IC-ISM) markings.

SuperHero Reference
Builds upon SuperHero IEPD demonstrating how to use references in a NIEM IEPD.

Terms

This section contains a list of terms with links to their source definitions across the following technical specifications:

- CTAS: Conformance Targets Attribute Specification v3.0, July 31, 2014
- HLVA: High Level Version Architecture Specification v3.0alpha1, September 3, 2014
- MPD: Model Package Description Specification v3.0, August 15, 2014
- NDR: Naming and Design Rules Specification, v3.0, July 31, 2014

When a term was defined in multiple specifications, a link to each definition was provided.

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

A

- alpha: HLVA Section 8.1
- appinfo namespace: NDR Section 10.9.1
- application information: NDR Section 10.9
- artifact: MPD Section 2.3
- artifact set: MPD Section 2.3
- association: NDR Section 10.3
- association type: NDR Section 10.3
- attribute: NDR Section 3.3
- augmentable type: NDR Section 10.4.1
- augmentation: NDR Section 10.4
- augmentation element declaration: NDR Section 10.4.4
- augmentation type: NDR Section 10.4.4

back to top

B

- base type definition: NDR Section 3.4.1
- beta: HLVA Section 8.1
- business rule schema: MPD Section 6.2
- business rules: MPD Section 6.2
- bump: HLVA Section 7.3

back to top

C

- change log: HLVA Section 7.2, MPD Section 5.3
- code simple type: NDR Section 11.1.2.3
- code type: NDR Section 10.2.4
- coherence: HLVA Section 4.1.2

- collaboration area: HLVA Section 3.2
- complex type definition: NDR Section 3.4.1
- conformance assertion: MPD Section 5.7
- conformance target: CTAS Section 3.1, MPD Section 3.1, NDR Section 3.6
- conformance target identifier: CTAS Section 3.1, MPD Section 3.1, NDR Section 3.6
- conformant document: CTAS Section 3.2
- conformant element information item: NDR Section 4.1.4
- conformant instance XML document: NDR Section 4.1.4
- conformant schema document set: NDR Section 4.1.3
- constraint rule: MPD Section 2.9, NDR Section 2.4.1
- content element: NDR Section 12.1
- core synchronization: HLVA Section 3.3
- core update: HLVA Section 3.1
- core update process: HLVA Section 3.3
- cross-domain harmonization: HLVA Section 3.3

back to top

D

- data component: MPD Section 2.6
- data definition: NDR Section 7.4
- deprecated component: NDR Section 10.9.1.1
- documented component: NDR Section 7.4
- domain reconciliation: HLVA Section 3.3
- domain update: HLVA Section 3.1
- domain update process: HLVA Section 3.3

back to top

E

- effective conformance target identifier: CTAS Section 4, NDR Section 3.6
- effective conformance targets attribute: CTAS Section 4
- element: NDR Section 3.3
- element declaration: NDR Section 3.4.1
- extension schema document: MPD Section 4.3, NDR Section 4.1.2
- external adapter type: MPD Section 4.4, NDR Section 10.2.3.2
- external schema document: MPD Section 4.4, NDR Section 10.2.3

back to top

F

- full NIEM information exchange package: MPD Section 3.2.3

back to top

H

- harmonization: HLVA Section 3, MPD Section 2.6

back to top

I

- IEP conformance target: MPD Section 5.6
- information exchange package: MPD Section 3.2.3
- information exchange package documentation: MPD Section 3.2.2
- informative: NDR Section 2.4
- instance document: NDR Section 3.4
- instance XML document: MPD Section 3.2.2
- interpretation rule: MPD Section 2.9, NDR Section 2.4.1
- issue tracking area: HLVA Section 3.2

back to top

L

- local term: NDR Section 10.8.2.1
- local terminology appinfo namespace: NDR Section 10.8.2.1

back to top

M

- machine-readable annotation: NDR Section 10.9
- major release: HLVA Section 3.1
- metadata element declaration: NDR Section 10.5.2
- metadata type: NDR Section 10.5.1
- micro release: HLVA Section 3.1
- minor release: HLVA Section 3.1
- model package description: HLVA Section 3.1, MPD Section 3.2.1
- MPD catalog document: MPD Section 5.1
- MPD class: MPD Section 3.1
- MPD root directory: MPD Section 7

back to top

N

- namespace URI directive: HLVA Section 7.3
- normative: NDR Section 2.4
- NIEM core: HLVA Section 1.2
- NIEM domain: HLVA Section 1.2
- NIEM governance: HLVA Section 1.2
- NIEM issue: HLVA Section 3.1
- NIEM release: HLVA Section 3.1
- NIEM wantlist: MPD Section 6.1

back to top

O

- object type: NDR Section 10.2.1

back to top

P

- path name: MPD Section 5.2.4
- pre-alpha: HLVA Section 8.1
- publication area: HLVA Section 3.2
- published update: HLVA Section 3.1

back to top

R

- readme artifact: MPD Section 5.4
- reference element: MPD Section 5.2.4.6, NDR Section 12.2
- reference schema document: MPD Section 2.8, NDR Section 4.1.1
- reference schema document set: MPD Section 2.8
- release area: HLVA Section 3.2
- release candidate: HLVA Section 8.1
- release cycle: HLVA Section 8
- release manifest: HLVA Section 7.1
- representation element declaration: NDR Section 10
- resolved element information item: NDR Section 5.6.4
- resolve URI: MPD Section 5.2.4
- role type: NDR Section 10.2.2
- RoleOf element: NDR Section 10.2.2

back to top

S

- schema component: MPD Section 2.7, NDR Section 3.4
- schema document: NDR Section 3.4
- schema document subse: MPD Section 4.2.1
- Schematron schema: MPD Section 6.2
- simple type definition: NDR Section 3.4.1
- structures namespace: NDR Section 10.10
- subset schema document: MPD Section 4.2.1

back to top

V

- valid: NDR Section 3.4
- validity constraint contex: MPD Section 5.6.1

back to top

X

- XML catalog documen: MPD Section 5.5
- XML document: MPD Section 2.7, NDR Section 3.2
- XML Schema: MPD Section 2.7, NDR Section 3.4

- XML schema assembly: MPD Section 2.7
- XML Schema definition language: NDR Section 3.4
- XML schema document: MPD Section 2.7
- XML Schema document set: NDR Section 3.4
- XML schema validation: MPD Section 2.7

back to top

Z

- ZIP file: MPD Section 3.2

back to top

Reusable XML Snippets

Contained below are reusable XML snippets of the various NIEM XML components. These snippets could also be called patterns or templates. They are available for NIEM developers to cut and paste directly into their integrated development environment to save development time.

These snippets have been updated and relocated to the **Reference / Concepts** section.

IEPD Change Control and Version Management

- NIEM change control and versioning
- IEPD change control and versioning
- Backward compatibility

This FAQ relates best practices for NIEM IEPD change control and versioning. Before discussing these concepts as they relate specifically to an IEPD, we begin with an overview of the NIEM architecture, and follow that with several applicable FAQs about NIEM change control in general.

The audience is intended to be NIEM Information Exchange Package Documentation (IEPD) developers and implementers who transmit or share NIEM IEPs (Information Exchange Packages, i.e., instance XML documents) based on one or more IEPDs.

The scope covers all aspects of IEPD change control and versioning based on NIEM releases, including how NIEM releases relate to IEPDs and their implementations, compatibility, and what users and implementers should expect. The intent is to help the NIEM community understand how to implement NIEM effectively in the context of annual releases, and know the limitations and trade-offs.

NIEM change control and versioning

How are NIEM schemas versioned?

A NIEM schema document identifies its version within the schema-level `targetNamespace` attribute. The NDR requires that every NIEM conformant schema document have a unique `targetNamespace`. This includes all NIEM schemas in releases, domain updates, and IEPDs.

For example, within NIEM 3.0:

- The `targetNamespace` for the Biometrics Domain is `http://release.niem.gov/niem/domains/biometrics/3.0/`
- The `targetNamespace` for the Justice Domain is `http://release.niem.gov/niem/domains/jxdm/5.0/`

NOTE: The only exception to this rule is a NIEM subset schema. See the next FAQ about subsets.

If no changes to a schema document are required, then its `targetNamespace` will not change, and the schema document will go forward into future releases as-is. Only when the content of a schema document changes must its `targetNamespace` change. When changed, generally only the version identifier (`3.0`, `5.0`, etc.) portion of the `targetNamespace` URI will be modified. However, this is only a convention, not a normative rule, so in reality the entire `targetNamespace` string determines the version of a NIEM schema document.

How are NIEM subset schemas versioned?

In NIEM, the `targetNamespace` of a subset schema document based on a reference schema document (within a release) is never unique. A subset schema document always takes the

`targetNamespace` of the reference schema document from which it was built.

(See Information Exchange Package Documentation 5.0 for subsetting operations.)

This is because a subset schema is essentially a reduced copy (i.e., a subset) of that reference schema document. So, an XML document instance defined by a subset schema document is also defined by its associated reference schema document. And in fact, a conformant subset schema document must be replaceable by the full reference schema. This also explains why the `targetNamespace` for a subset schema document does not change; each is based on a published reference schema document.

Can published NIEM schemas be changed?

An important practice is that after a NIEM XML schema document has been officially released and published, neither its content nor its `targetNamespace` can be changed. We say that published NIEM reference schemas (releases) live forever. This policy ensures that the official NIEM releases (i.e., reference models) are always available to support information exchanges until an author or sponsor is ready to change them.

Are NIEM versioning policies in standard practice by all organizations?

No. Some standards development organizations (SDO) do not follow the NIEM versioning policies. Some change content directly within a published `targetNamespace`. Others use a similar technique, but add a version identifier that is not a part of the namespace. In such cases, it can be difficult to know if the content of a namespace has changed (and which version you have). The NIEM namespace policy ensures absolute transparency of change. If two NIEM schema documents bear different target namespaces, then their content is different. Furthermore, within NIEM artifacts that use them (such as IEPDs), subset schema documents must be explicitly identified.

I've heard that "a NIEM minor release is backward compatible." Is this true?

There is a bit of truth to this statement. A NIEM major series of releases is numbered as follows: 3.0, 3.1, 3.2, etc. One and only one NIEM core schema document (version 3.0) and `targetNamespace` is common to all of these releases. The next NIEM major release (version 4.0) will change the NIEM core schema document. So, release 4.0 will not use the 3.0 core schema document. Thus, one might say that the NIEM 3.0 series (i.e., versions 3.0, 3.1, 3.2, etc.) is compatible in terms of Core because all versions in the series use the same NIEM core schema document. However, domain schema documents may change in ways that prevent backward compatibility between minor releases within a major series.

What is *XML schema validation*, and what does it mean to be *schema-valid*?

XML schema validation is the process of checking an XML document to determine if it conforms to (or is defined by) an XML schema with which it has been associated. Although not considered part of validation, all valid XML documents must also be syntactically well-formed. Essentially, this means its XML open/close tags are complete and balanced. A *schema-valid* XML document has passed a validation check against an associated schema. In NIEM, these terms refer to W3C XML Schema validity.

IEPD change control and versioning

How are IEPDs versioned?

The method and scheme for versioning IEPDs is very similar to that used for NIEM releases. An IEPD is assigned a unique URI that is under the control of the IEPD author or sponsor. If significant changes are made to an IEPD then authors are expected to change the URI for that IEPD. This is a MUST if changes will impact XML instance document validation with the IEPD schema.

Insignificant changes may not require a change in URI. For example, text corrections or minor clarifications to IEPD documentation (where it does not impact schema validation) might not require a URI change. This decision is left to the judgement of IEPD authors, sponsors, or certification authorities.

See the NIEM Information Exchange Package Documentation 5.0 - Section 5.2.4.1 for IEPD URI syntax details and rationales.

Who is responsible for managing versions of a NIEM IEPD?

While related to NIEM by reusing data components in published representations of the reference model (releases), an IEPD is authored independently of NIEM. The IEPD author, developer, or sponsor is responsible for managing IEPD versions.

How can I determine the version of a NIEM release (releases) used in an IEPD?

There are at least two ways to determine the NIEM release used by an IEPD. First, an IEPD author should identify the NIEM release version(s) in use within the documentation. A second method is to inspect `targetNamespace` attributes within the IEPD's schema documents and compare them to their respective reference schema documents (within the releases).

Core related schemas documents such as `niem-core.xsd` and `structures.xsd` can only identify the major series (1.0, 2.0, 3.0, etc.). This is because Core does not change from one minor release to the next within a major series. For example, NIEM 2.0 and 2.1 both contain the same 2.0 core schema document with same `targetNamespace`. Domain and code list schema documents can help to identify the minor release series (second integer). But be sure to compare to the release schema documents, because some domains (such as Justice) and some code lists have legacy versioning schemes that do not correspond directly to the release version. For example, the current version number in the Justice `targetNamespace` URI is 7.0, not 5.0.

Can an IEPD employ data components from more than one NIEM release version?

The NIEM Information Exchange Package Documentation Specification allows an IEPD to employ XML data components from multiple NIEM releases. This means that schema subsets from multiple releases can be used together in a single IEPD.

That said, mixing releases in an IEPD can introduce multiple versions of the same XML elements and types (sometimes with slightly different structures) resulting in semantic duplication, overlap, and potential ambiguity. On the other hand, each NIEM release has been harmonized to remove as much duplication as possible.

Therefore, we strongly recommend that an IEPD be designed to use a single NIEM release version. This practice is easier to control, less complex, and less error-prone.

Can I use NIEM domain updates and/or core supplements in my IEPD?

A NIEM domain update contains one or more schema documents representing changes to one or more domains. A NIEM core supplement contains one or more schema documents representing additive-only changes to a published NIEM Core. An IEPD may incorporate domain updates and core supplements.

That said, semantic overlap and duplication that can occur by mixing multiple NIEM releases within an IEPD (See FAQ above), may also happen when using a domain update or core supplement. This is because a domain update changes a given domain's published `targetNamespace` by creating a new `targetNamespace` and applying the changes there. In doing so, it may replicate data components defined by the `targetNamespace` it changes. (See the NIEM High Level Version Architecture 3.0</a>.)

Furthermore, the trade-off for allowing NIEM domains to reuse (and reference) each others' data components while also independently publish changes outside the regular release cycle, is that over time the level of semantic overlap can increase across a given release and its associated domain updates and core supplements.

To the extent possible, within an IEPD avoid mixing duplicate versions of the same data component. Use the subset technique where possible to reduce duplication; but realize that some level of semantic duplication and overlap may be unavoidable.

"An IEPD is the point of NIEM interoperability." … What does this mean?

Once published, a NIEM release is always available as-is under a unique `targetNamespace`. So, once designed and built from an existing NIEM release (and optionally any existing domain updates or core supplements, an IEPD implementation establishes interoperability among senders/receivers of XML instance documents (IEPs) that validate to the IEPD's schema document set.

This means that a NIEM implementation is immune to NIEM changes (new releases, domain updates, and core supplements), and will continue to work as long as IEPs (instance XML documents) are schema-valid to the original IEPD schema document set.

Furthermore, if the IEPD schemas change and the implementation changes accordingly, and the changes are such that the schema document set is backward compatible (see *backward compatibility above*) with the previous set before the changes, then the implementation will continue to work with the old and new XML instance documents.

Must an IEPD be updated (changed) when a new NIEM release is published?

No, an IEPD is not required to change as a result of NIEM evolution. If the data exchange requirements for which the IEPD was developed are still valid, then there is no reason to update the IEPD (even if new content in a recent NIEM release or domain update is available).

When must an IEPD be updated to a new NIEM release version?

All previously published NIEM releases are always available at the NIEM release Web site. Therefore, if an IEPD continues to satisfy its information sharing requirements as designed or implemented, then it is not necessary to update it because there is a newer NIEM release.

When an IEPD author, sponsor, implementer, or the exchanging parties determine that:

(1) data exchange requirements have evolved significantly enough; or (2) newer NIEM content is available and the need to use it within data exchanges exists;

then, it may also be worthwhile to consider an IEPD update and likely an update to all corresponding IEPD implementations.

What does IEPD versioning look like for producer and consumer implementations?

Information producers and consumers must synchronize any IEP (XML instance document) exchanged through the XML schema definitions in the IEPD that define the IEP. This means that (1) the IEPD implementation must understand the IEPD schema assembled from the IEPD schema document set; so that (2) the implementation must recognize and correctly process all elements of an incoming IEP (XML instance document) defined by the IEPD; and (3) the IEP must be schema-valid to the IEPD schema document set. Therefore, producer and consumer implementations of an IEPD must be aware of the IEPD versions their software implementations are based on.

Backward compatibility

What is *backward compatibility* in XML Schema?

Consider a single XML schema document that defines a set of XML instance documents. This means that each XML instance document in the set is schema valid to the schema document. If the schema document is changed, then the nature of the changes will determine how different the set of valid XML instance documents is now as a result of changes.

Schema document changes might be such that the original set of valid XML instance documents are still valid for the new (modified) schema, and other new XML instance documents are also valid. This can happen if, for example, one or more additional XML elements are added to an XML type definition with cardinality (`minOccurs`) set to zero (i.e., use of these elements is optional). This is essentially backward compatibility, i.e., the new changed XML schema is backward compatible with the previous XML schema, because the original set of valid XML instance documents are still valid to the new schema document.

Does NIEM support backward compatibility?

No, NIEM does not support backward compatibility out-of-the-box. NIEM does not and cannot guarantee that an IEPD built from any given major or minor release version will be backward compatible with an IEPD built from an earlier release. However, under certain conditions it may be possible to design, build, and change an IEPD such that it is backward compatible for the same set of schema-valid IEPs it defined before it was changed. See the next question below.

How can I build an IEPD that supports/maintains backward compatibility with older IEPs?

To design and build such an IEPD requires some degree of pre-planning. The author must be prepared to limit the kinds of changes he can apply to later versions of the IEPD. Subsequent changes to the IEPD schema documents must preserve the original set of valid IEPs, while allowing for new valid members of that set.

Such changes are generally additive-only and may include (but are not limited to):

- add elements with `minOccurs="0"` to types;
- add elements to existing substitution groups; relax (not restrict) cardinality constraints; and
- employ wildcards (as appropriate).

As long as changes to the IEPD schema documents do not render previously schema-valid IEPs invalid, then backward compatibility is preserved.

How can two different versions of an IEPD be processed by one implementation?

A software implementation can process one or more versions of an IEPD. The software should have a set of data structures for the data set defined by each IEPD (for example, a set of Java objects for each IEPD). The software will associate an incoming IEP (XML instance) with one of two namespaces to determine which set of data structures to use for the incoming IEP. To do this, it parses the IEP document element to identify the unique namespace of the schema that defines that element. This namespace identifies which set of data structures to load with the incoming IEP data and subsequently process.

Releases

- Overview
- Kinds of releases
  - Major release
  - Minor release
  - Published updates
    - Core Supplements
    - Domain Updates
- Release cycle
- Release features
  - Persistence
  - Formats
- Publication sites
- Release development
  - Draft stages
  - Processes
  - Areas of responsibilities
- Submitting feedback
- See more

Overview

A NIEM release is a versioned set of schemas and supporting artifacts published by the NIEM Management Office. Release schemas contain reusable properties and types meant for use in information exchanges. These schemas include NIEM Core, domains, and code tables.

Each release is produced in collaboration with the NIEM Business Architecture Committee (NBAC), the NIEM Technical Architecture Committee (NTAC), and the general NIEM community.

See these links for more information about the current release and the latest draft release.

Kinds of releases

A release can be a major or minor release.

Major release

A major release may include content changes to NIEM Core and to any other namespace in the release. It may also include architectural changes, as determined by any updates to the Naming and Design Rules and by the NIEM Technical Architecture Committee (NTAC).

The first digit of the release number indicates the major release series. The second digit will be “0”. For example, NIEM 3.0, 4.0, and 5.0 are all major releases.

Minor release

The NIEM Core namespace and the architecture are locked in minor releases. Content changes can be made to domains and to any namespace not imported by Core.

The second digit in the release number indicates the minor release series. For example, the two minor releases following the NIEM 4.0 major release are NIEM 4.1 and NIEM 4.2. Because Core is locked in minor releases, NIEM 4.0, 4.1, and 4.2 all share the same NIEM Core schema.

Core supplements (see below) can be used to publish Core-related changes additively during minor releases or anytime outside of the standard release cycle.

Changes were made starting with NIEM 5.0 to remove imports from Core to code namespaces. Breaking these dependencies now allows these code sets to be updated directly as needed during minor releases, without the need to publish updates via a separate Core Supplement.

Published updates

NIEM also has the option of publishing updates between official releases to distribute new content or provide workarounds for bugs or other issues in a release. These updates are published in separate namespaces and do not overwrite content in an existing release. In an IEPD, content can be reused from both a NIEM release and from any applicable update together.

Core Supplements

A Core Supplement is an update to adjust content from Core or a namespace imported by Core, published by the NBAC outside of a major release. Core supplements can be used in an IEPD together with the corresponding major release or any minor release in that series. The changes in a Core Supplement will be integrated back into NIEM during the next major release.

See the Core Supplement page in this section for more.

Domain Updates

A domain update is an update to adjust content from one or more domains or domain-related code namespaces, published outside of the annual release cycle. Domain updates can be used in an IEPD together with its corresponding release. Because the changes in a domain update do not affect Core, they will be integrated back into NIEM during the next major or minor release.

See the Domain Update page in this section for more.

Release cycle

Starting with NIEM 3.0, NIEM has adopted the strategy of publishing annual releases in a **three-year cycle**: major, minor, minor. A major release one year will be followed by minor releases the next two years.

This cycle allows NIEM to be more responsive to community needs and provide a more predictable

schedule, while also maintaining some stability against too-frequent Core and architectural changes that may require updates to specifications, tools, and training.

Core supplements and domain updates allow changes to be published outside of the standard release cycle in an additive manner, so these updates can be used in addition to their corresponding releases.

Release features

Persistence

A key feature of NIEM releases is that they are persistent. Information exchanges that are built based on NIEM will not break or become invalid just because NIEM publishes a new release. Old releases will remain available. The determination of if or when to update an exchange to use a newer version of NIEM is a business decision driven by the exchange partners. NIEM's release cycle does not ever force IEPDs to be updated in lock-step.

Formats

The NIEM XML schemas in a release are the authoritative source for release content. Documentation is also provided in a release via a Readme, spreadsheets, and CSV files for easier data imports.

See the Release Artifacts page in this section for more about what appears in a NIEM release.

Publication sites

NIEM releases are now hosted on two sites.

release.niem.gov

NIEM releases have traditionally been posted to **https://release.niem.gov/niem**.

- Target namespaces of release schemas will resolve to this site.

GitHub

Releases are now being also being posted to a GitHub repository at **https://github.com/NIEM/NIEM-Releases** for better developer support and release maintenance.

- Click on the tags link to switch between releases.
- The package can be downloaded in full as a zip file by clicking on the green `Clone or download` button, and then clicking `Download ZIP`.
- The issue tracker is also located here on GitHub at https://github.com/NIEM/NIEM-Releases/issues

Release development

Draft stages

A release is developed in the following stages:

Alpha

Major content submissions and issues should be submitted by the input deadline for the alpha stage. This is to ensure there is adequate time available to review, integrate, and harmonize the changes into the release.

Alpha releases are published internally to the NIEM committees and domains, who are free to share the package with their communities for review and comment.

Beta

After major changes and issues have been submitted during the alpha stage, the beta stage allows harmonization between the domains. Issue resolutions and smaller changes can occur during this stage.

Beta releases are announced and distributed to the NIEM community for public review and feedback.

Release Candidate

A release candidate acts as a draft of the final release. This stage allows for that draft to be double-checked before publication. Only bug fixes are considered during this stage; other submitted comments and requests will be queued for the following release.

Processes

The following are activities and processes relating to updating NIEM releases:

Harmonization

The process of modifying schemas in an incremental fashion for the purpose of improving the quality with respect to some criteria. For example, schemas may be harmonized to reduce semantic overlap of its components. Schemas may also be harmonized to ensure uniformity of the vocabulary used in component names and definitions.

Harmonization will be an ongoing process. As new domains, requirements, and content are added, existing NIEM components may need to be adjusted.

The purpose of harmonization is to reduce duplication, but not necessarily to reduce cross-domain dependencies. Domains can and should reuse content from another domain if it is considered to be the authoritative source for the information. Content should only be refactored into Core if there is no clear, single authoritative source for it and would be

better managed collaboratively by the NBAC.

Cross-Domain Harmonization

During this process, the NBAC may create tiger teams or working groups to resolve inconsistencies, overlaps, and other semantic issues between the domains and Core. Changes to a domain may be published with the next major or minor release. Changes to Core will be staged for the next major release.

Core Synchronization

During this process, NBAC-approved changes are applied to Core and any outstanding Core supplements are merged into the namespace. Domains will be adjusted and synchronized to the new Core content and namespace. This set of Core and domain changes will form a major release.

Areas of responsibilities

Core Content

The NIEM Business Architecture Committee (NBAC) is responsible for the content of Core and for the publication of major and minor releases and Core Supplements, along with the NIEM Management Office.

Domain Content

The NIEM model is decentralized. Individual domains are responsible for managing their respective content. Community feedback may be channeled directly through the domain's point of contact, or may be sent via the standard feedback mechanisms listed below for the appropriate distribution.

The NBAC will ensure that domain namespaces meet NIEM conformance rules for major and minor releases.

Architecture

The NIEM Technical Architecture Committee (NTAC) is responsible for the architecture of the NIEM model, as defined by the NIEM Naming and Design Rules (NDR) and other NIEM specifications.

Submitting feedback

Community feedback on the content of a NIEM release may be entered into the NIEM Release issue tracker or may be emailed directly to the NIEM comments mailing list.

Note that architectural feedback or issues should be submitted to the NDR issue tracker.

See more

- Core Supplement

  A Core Supplement is an incremental NIEM release that contains new or updated components for the NIEM Core namespace. These changes are published in separate schemas that can be used in addition to the original Core namespace. Core Supplements are used as a way of 'adjusting' Core content while it is locked between major releases.

- Domain Update

  A domain update is one or more schemas that constitute changes to a NIEM domain outside of the standard NIEM release cycle.

- Release Artifacts

  Each NIEM release comes bundled as a zip file containing schemas and other supporting files.

Core Supplement

A **Core Supplement** is a special type of NIEM release that can be used in addition to a corresponding release. It is strictly additive in nature.

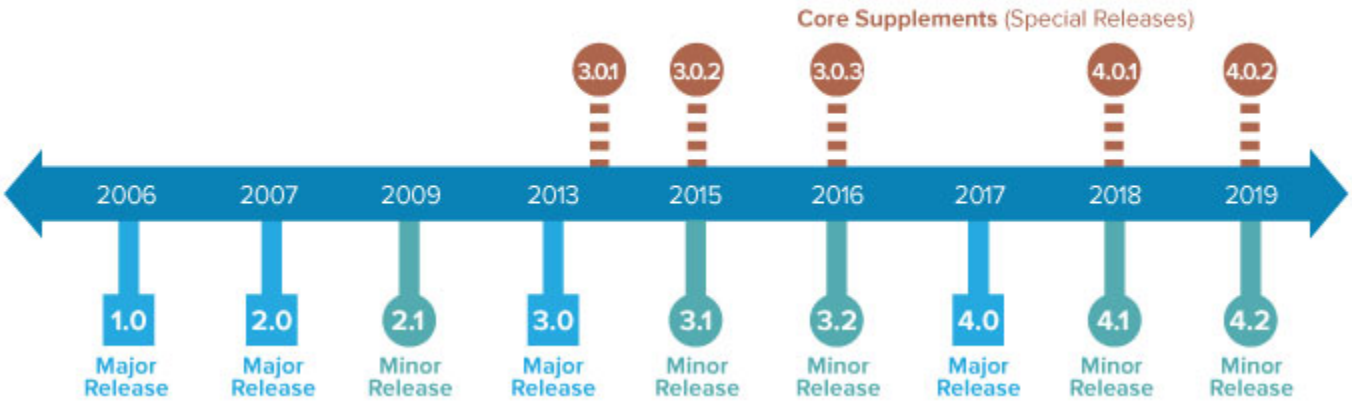
- Versions
- Namespaces
- Locations
- Using a Core Supplement

A core supplement can be issued when the NIEM Business Architecture Committee (NBAC) determines it is necessary to add content to a published NIEM Core namespace, or a namespace imported by NIEM Core. The purposes for issuing a CS may include:

- To update a code list with the latest values
- To publish bug fixes
- To support domain or community requirements

A Core Supplement can be published at any time, though they tend to coincide with NIEM minor releases as an interim way of adjusting Core content while waiting for the next major release.

During a major release, all of the changes in the Core Supplements for the previous major release series are merged back into the primary Core namespace.



Versions

Core Supplements are versioned similar to the Core namespace, with a third digit representing the supplement number.

- CS 3.0.2 is the second Core Supplement for the NIEM Core 3.0 namespace.
- CS 4.0.1 is the first Core Supplement for the NIEM Core 4.0 namespace.

Core Supplements in the same series (e.g., 4.0.x) can be used individually or together, along with the

corresponding NIEM Core namespace.

- Use CS 3.0.1 and CS 3.0.2 with NIEM Core 3.0 (same series).
- Use CS 3.0.1 with NIEM Core 4.0 (different series).

Namespaces

In order to make it easier to use Core Supplements along with the Core namespace in IEPDs, Core Supplements are given a different namespace prefix, corresponding to the CS version number.

- Element from Core 3.0 - `nc:LocationStateUSPostalServiceCode`
- Element from CS 3.0.1 - `nc-3.0.1:LocationStateUSPostalServiceCode`

Target namespaces of Core Supplements are similar to the Core target namespace, with two differences:

- `release` is changed to `publication` since it does not represent official major or minor release content.
- The supplement number is appended to the end.

- Core 3.0 target namespace: `http://release.niem.gov/niem/niem-core/3.0/`
- CS Core 3.0.1 target namespace: `http://publication.niem.gov/niem/niem-core/3.0/1/`

Locations

Core Supplements are bundled with minor releases for convenience. The files are located in a subfolder off of the folder for the primary namespace.

- The 3.2 NIEM minor release includes three Core Supplements - 3.0.1, 3.0.2, and 3.0.3.
- These files are located at:

```
- niem/
  - niem-core/
    - 3.0/
      - 1/
        - niem-core.xsd  # CS 3.0.1 Core schema
      - 2/
        - niem-core.xsd  # CS 3.0.2 Core schema
      - 3/
        - niem-core.xsd  # CS 3.0.3 Core schema
      - niem-core.xsd    # Core 3.0 schema (from the major release)
```

Core Supplements are also available individually in zip files at:

- 3.0 series of Core Supplements
- 4.0 series of Core Supplements

Using a Core Supplement

Although a Core Supplement (CS) may be published outside of the standard release cycle and will be limited in scope and content as compared to its counterparts in a full release, it is still NIEM-conformant content and should be used in a similar manner. Follow these steps:

- Determine if there is content in a CS that you need to use. If not, no further steps need to be taken and the CS may be ignored.

- Determine if you have any existing IEPDs that need to be updated. If so, reopen the IEPD subset wantlist in the Schema Subset Generation Tool (SSGT), remove any old components you wish to replace, add the relevant new CS components, and regenerate the subset. Edit the IEPD extension schema document(s) to import the CS schema document(s), reflect the changes made in the subset, and update the IEPD catalog and versioning metadata as needed.

- To use a CS as part of a new IEPD, include the relevant CS components while creating your NIEM subset and build your IEPD as normal.

- Note that you may be able to integrate CS components into your IEPD extension schemas via augmentations and/or element substitutions (especially for code list updates). If so, this requires little additional effort beyond making sure the new components are available in the NIEM subset. If augmentations and/or element substitutions are not available or appropriate, new components may be referenced and used directly in the extension schema documents.

- If you use any of the following NIEM 3.0 components:
    - Element `nc:JurisdictionUSPostalServiceCode`
    - Element `nc:LocationStateUSPostalServiceCode`
    - Type `usps:USStateCodeType`
    - Type `usps:USStateCodeSimpleType`
- Then, you have the option of using updated content from CS 3.0.1 by updating your NIEM subset and IEPD with the following:
    - Element `nc-3.0.1:JurisdictionUSPostalServiceCode`
    - Element `nc-3.0.1:LocationStateUSPostalServiceCode`
    - Type `usps-3.0.1:USStateCodeType`
    - Type `usps-3.0.1:USStateCodeSimpleType`

Domain Update

A domain update is one or more schemas that constitute changes to a NIEM domain outside of the standard NIEM release cycle.

- Kinds of domain updates
  - Incremental
  - Replacement
  - Coordinated domain update
- Key rules for a domain update
  - Rule 4-2
  - Rule 4-6
  - Rule 4-11
  - Rule 4-26
- Usage

Domain updates are published by the domain steward when it is determined to be necessary to publish additions or changes before the next release. During the next major or minor NIEM release, these changes will be integrated back into the regular domain namespace.

Kinds of domain updates

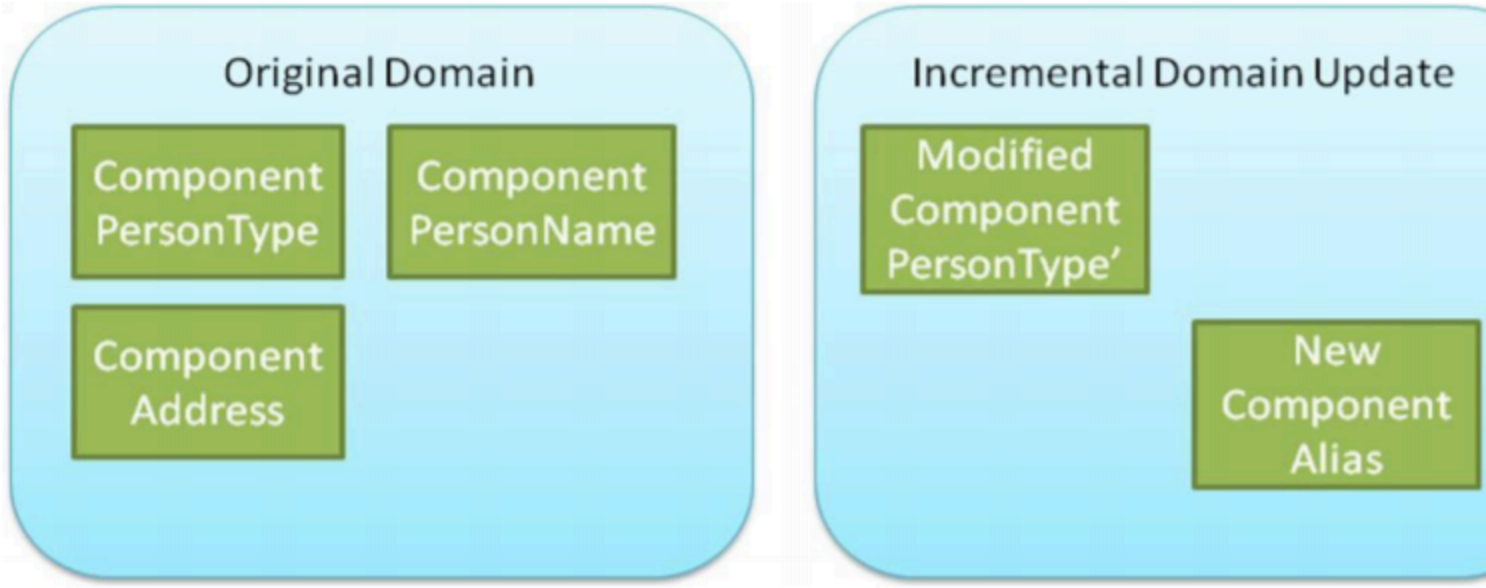
There are three kinds of domain updates:

Incremental

An incremental domain update is additive in nature. It contains only the domain components that are being added or modified. An incremental domain update may be used in an IEPD in addition to the regular domain namespace.

An incremental domain update may be a better choice if a domain has the need to immediately publish a small set of changes.

- The image below shows that an incremental domain update contains only new and modified components.
- Components with no changes are not duplicated.

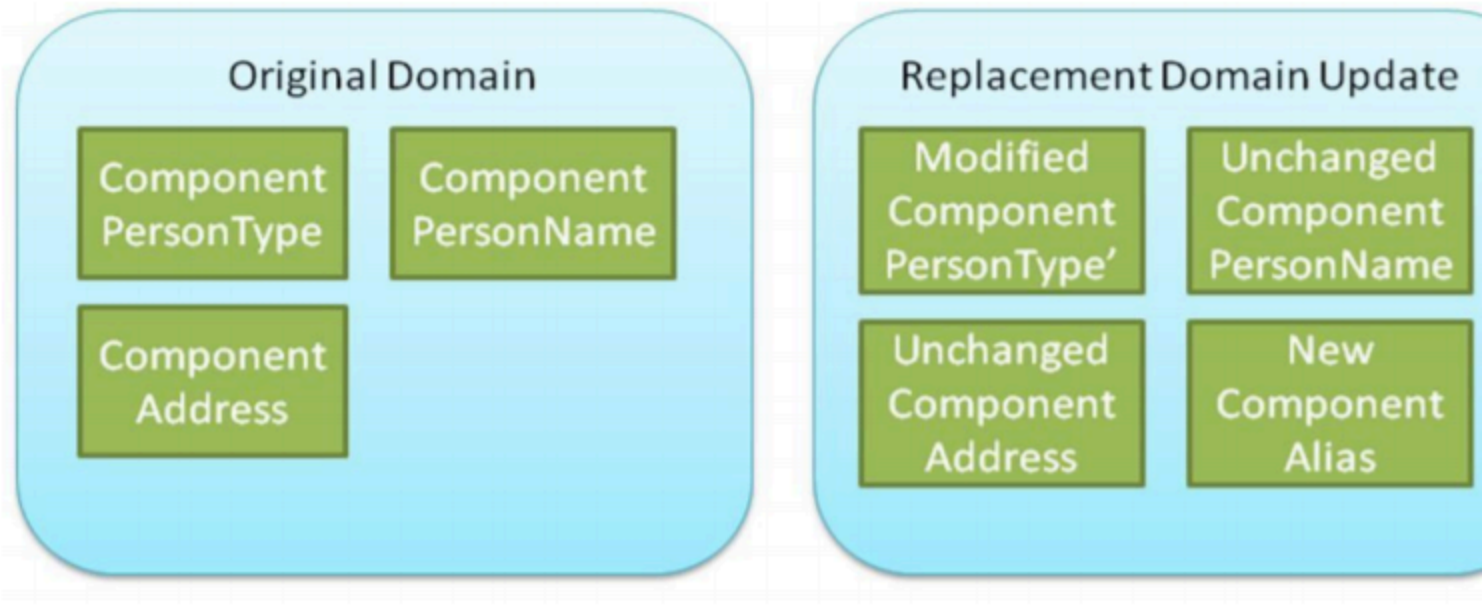


Replacement

A replacement domain update represents all domain components, with additions, modifications, and deletions applied. A replacement domain update should be used in an IEPD in lieu of the regular domain namespace if that IEPD requires the adjusted content.

A replacement domain update may be a better choice if a domain has the need to publish a large set of interdependent changes

- The image below shows that a replacement domain update contains the fully updated set of domain components.
- Components with no changes will also appear in the replacement schema.



Coordinated domain update

A coordinated domain update is a set of interdependent domain updates that are published together in one package. This allows related content that may span multiple namespaces to be updated in a more cohesive manner.

> Every domain update schema in a Coordinated domain update must be approved by its domain steward. No changes to a domain can be made without steward approval.

Key rules for a domain update

Version 1.0 of the Domain Update Specification describes what a domain update is and defines a set of rules that must be followed. The list below are highlights from the specification.

Rule 4-2

A domain update can only make changes for namespaces under its control. For example, a domain update cannot be used to redefine Core or another domain namespace.

Rule 4-6

A domain update must be conformant to the applicable NIEM Naming and Design Rules (NDR) reference schema document conformance target.

Rule 4-11

The target namespace for a domain update must follow the pattern:
`http://publication.niem.gov/niem/domains/{domain}/{domainVersion}/{domainUpdateNumber}`

The following example shows a target namespace for sample domain update for the Immigration domain, version 4.1:

```
<xs:schema
  targetNamespace="http://publication.niem.gov/niem/domains/immigration/4.1/1'
>

</xs:schema>
```

Rule 4-26

A domain update must pass the NDR conformance validation checks run by the Conformance Testing Assistant (ConTesA).

Usage

Domain updates are designed to be used in IEPDs.

- Add the domain update schemas to the schemas directory in the IEPD package.
- In an IEPD extension schema:
  - Declare a namespace prefix for the domain update schema
  - Import the domain update schema
  - Reference elements and types from the domain update schema

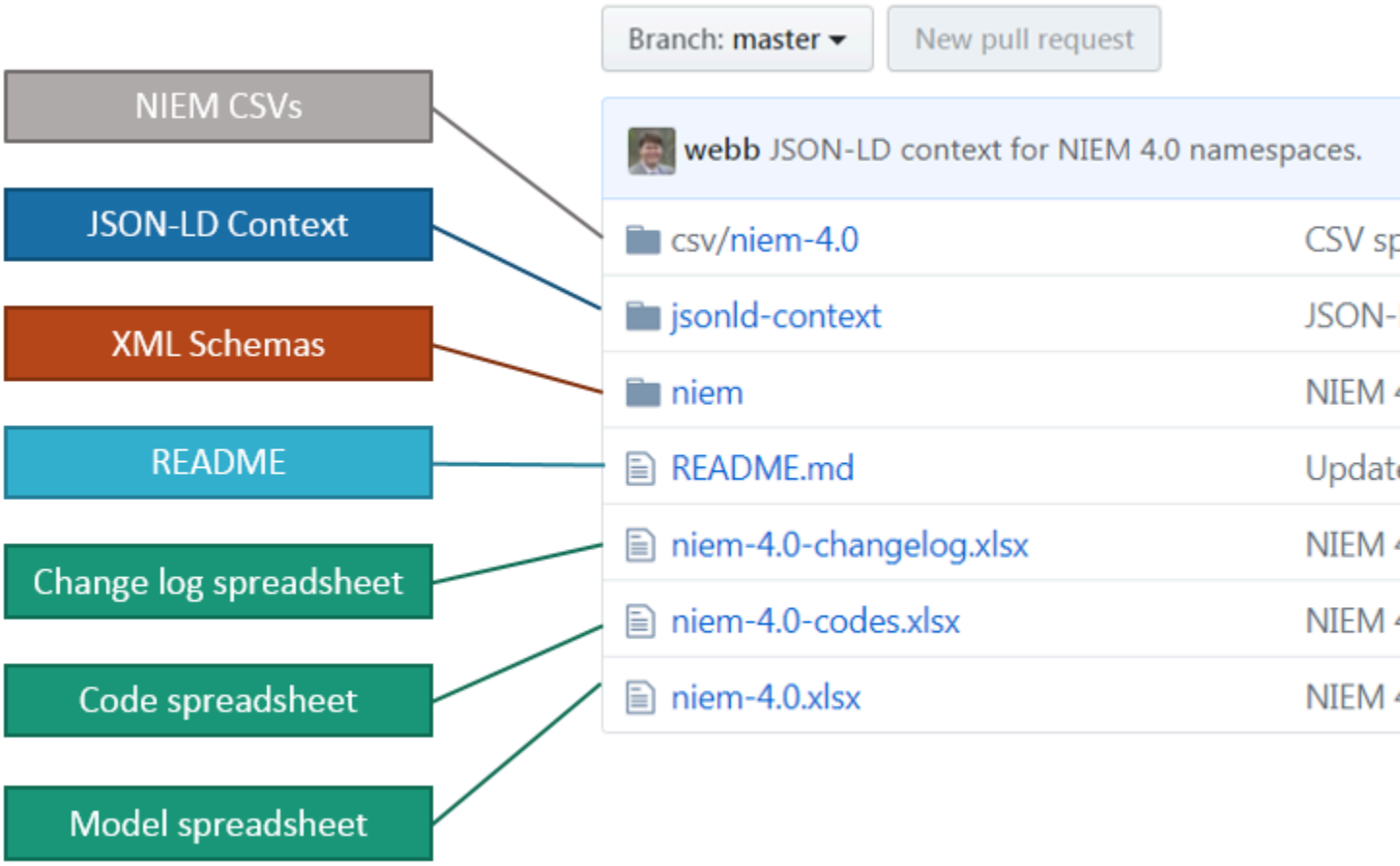
Fore more information, see similar usage information for a Core Supplement.

Release Artifacts

Each NIEM release comes bundled as a zip file containing schemas and other supporting files.

- XML Schemas
- Readme
- Model spreadsheet
- Code spreadsheet
- Change log spreadsheet
- CSVs
- Other artifacts

The image below shows the NIEM 4.0 release package on GitHub, with labels superimposed to point out the various artifacts.

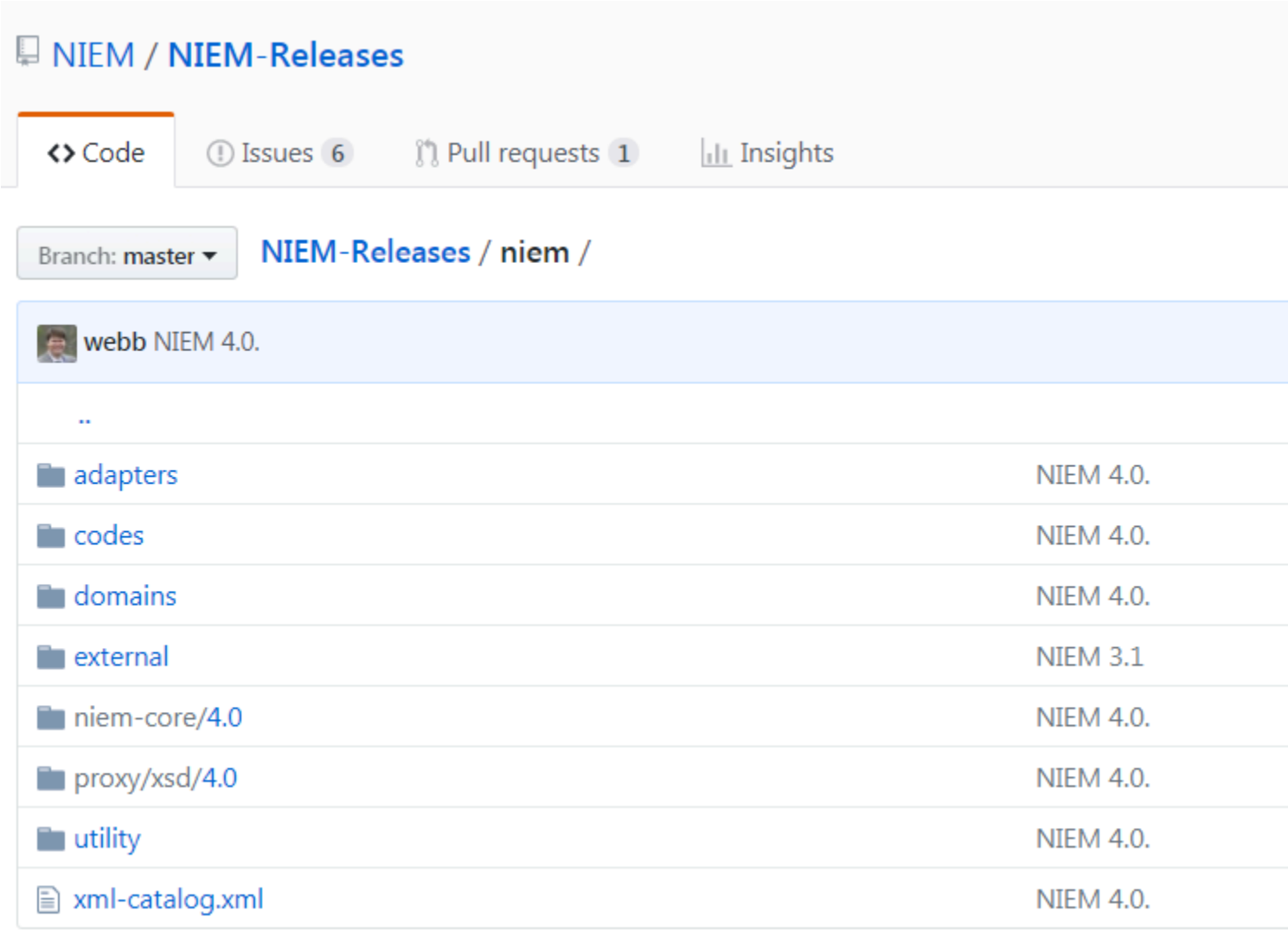


XML Schemas

The NIEM release XML Schemas define the model content and the internal constructs needed to support the model. Schemas include Core, domains, codes, external standards, structures and appinfo.

Schemas are located in the `niem` subdirectory of the package or repo.

The image below shows the schema layout of NIEM 4.0. The XML catalog is also present.



Readme

The release README file is a text file that provides a high-level overview of the changes made in the release, and in particular any changes that might affect Core.

This file is located in the root of the package or repo. The file name is `README.md`.

The format is “Markdown”, which provides simple syntax for basic HTML formatting. Markdown files can be viewed as plain text in any text editor, but the HTML formatting is rendered for a better display on GitHub or a markdown viewer.

NIEM / NIEM-Releases

Code | Issues 6 | Pull requests 1 | Insights

Branch: master ▾ NIEM-Releases / README.md

CB2 Update README.md

2 contributors

146 lines (122 sloc) | 5.93 KB

NIEM 4.0

This is the NIEM 4.0 release. NIEM 4.0 is the current release.

This release reflects updates to the NIEM Naming and Design Rules, as well as the new

NDR 4.0

This release of NIEM reflects updates to the NIEM Naming and Design Rules, including

- The structures schema incorporates a new attribute, uri, that supports cross-docu
  in the style of Linked Data.
- The vocabulary for local terminology, formerly in its own namespace, has been mo
- Abstract elements now have names that end in "Abstract", other than augmentatio

Code Lists Specification 4.0

This release reflects the new Code Lists Specification, including:

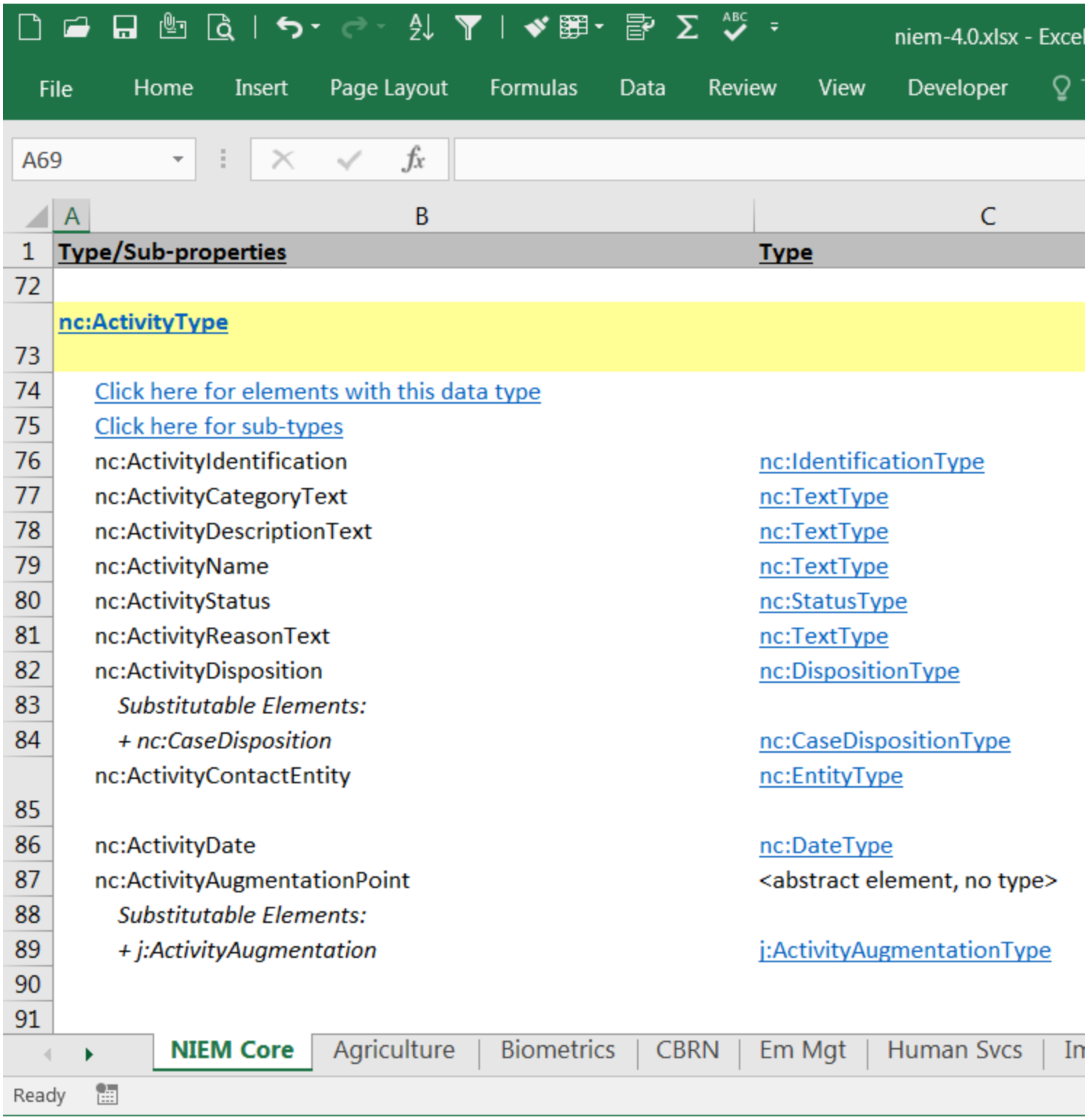
Model spreadsheet

The NIEM model spreadsheet provides user documentation for the properties and types of Core and the domain namespaces.

Each of these namespaces is represented in its own tab, with several supporting tabs at the end of the spreadsheet.

The main tabs are organized alphabetically by types (highlighted in yellow) and their sub-elements. Related components are hyperlinked for easier navigation.

This file is located in the root of the package or repo. The file name is “niem-version.xlsx”, with “version” representing the release number. For the 4.0 release, the spreadsheet is named `niem-4.0.xlsx`.



	A	B	C
1		Type/Sub-properties	Type
72			
73		nc:ActivityType	
74		Click here for elements with this data type	
75		Click here for sub-types	
76		nc:ActivityIdentification	nc:IdentificationType
77		nc:ActivityCategoryText	nc:TextType
78		nc:ActivityDescriptionText	nc:TextType
79		nc:ActivityName	nc:TextType
80		nc:ActivityStatus	nc:StatusType
81		nc:ActivityReasonText	nc:TextType
82		nc:ActivityDisposition	nc:DispositionType
83		*Substitutable Elements:*	
84		*+ nc:CaseDisposition*	nc:CaseDispositionType
		nc:ActivityContactEntity	nc:EntityType
85			
86		nc:ActivityDate	nc:DateType
87		nc:ActivityAugmentationPoint	<abstract element, no type>
88		*Substitutable Elements:*	
89		*+ j:ActivityAugmentation*	j:ActivityAugmentationType
90			
91			

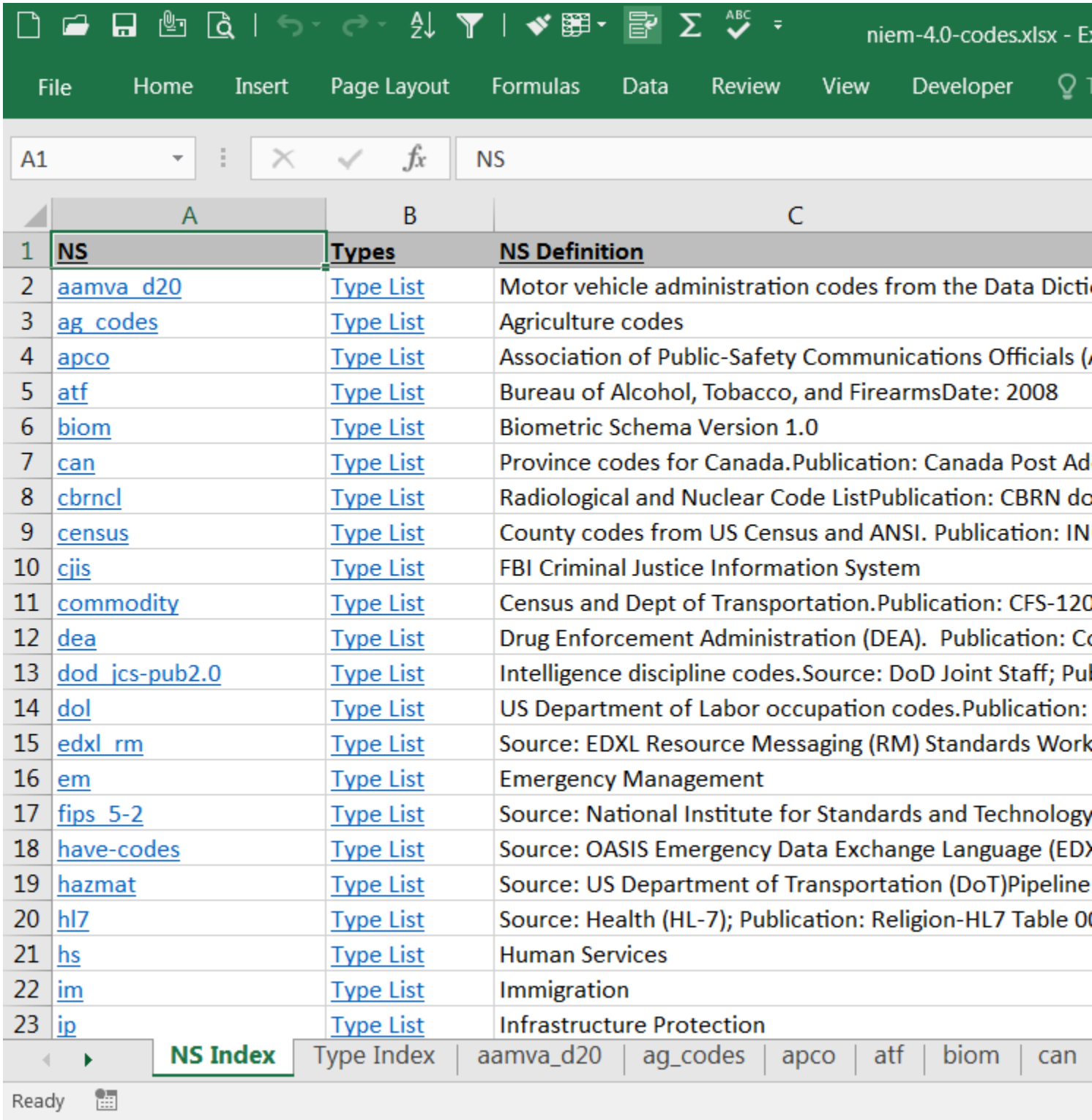
Code spreadsheet

The NIEM Code spreadsheet provides user documentation for the codes defined in a NIEM release.

This file is located in the root of the package or repo. The file name is `niem-[version]-codes.xlsx`.

Namespace index tab

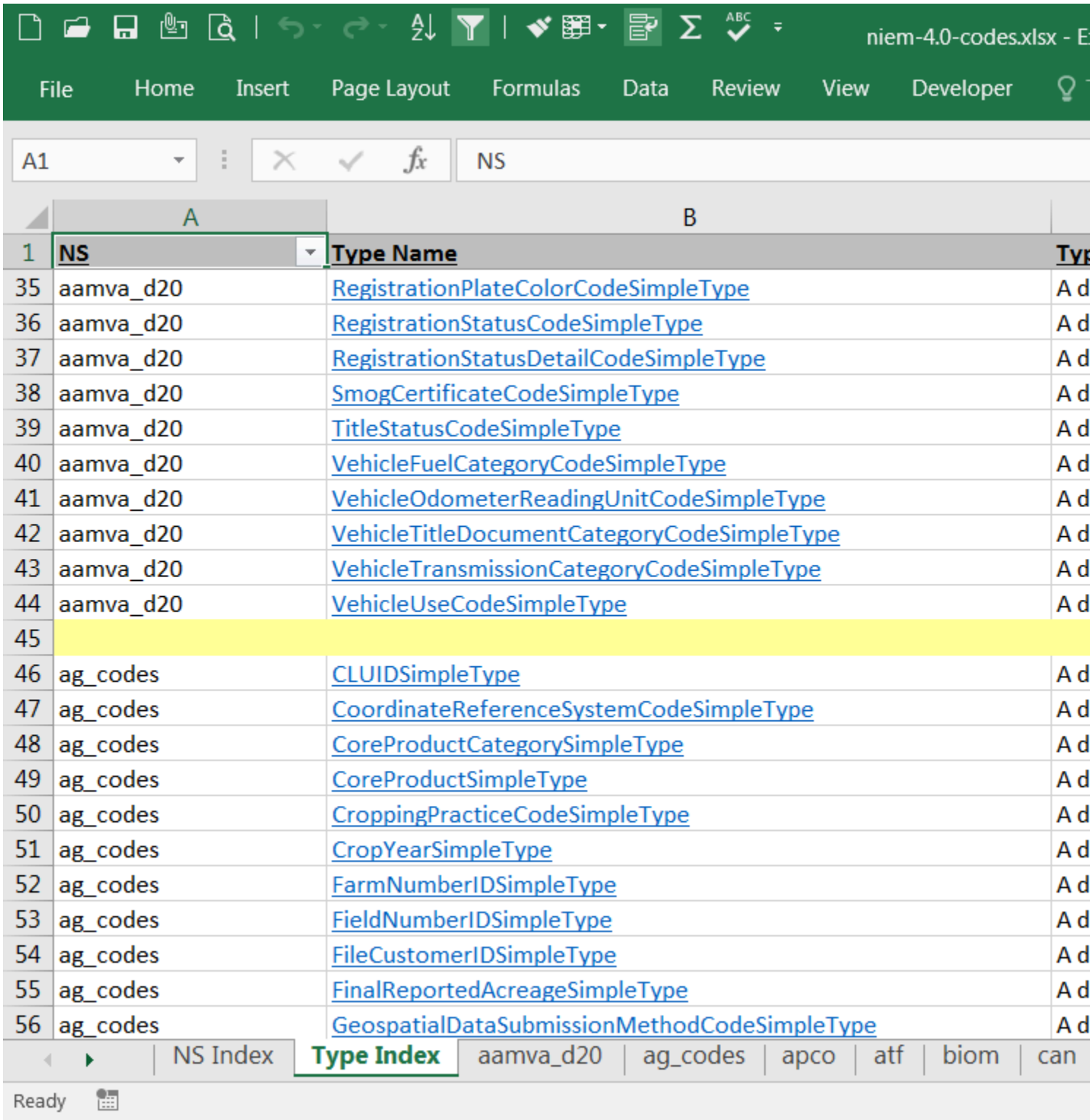
The first tab is an index of the code namespaces. Click on the link in the first column to go directly to the tab corresponding to that namespace. Click on the link in the second column to go to an index of code types for that namespace.



	A	B	C
1	NS	Types	NS Definition
2	aamva_d20	Type List	Motor vehicle administration codes from the Data Dicti
3	ag_codes	Type List	Agriculture codes
4	apco	Type List	Association of Public-Safety Communications Officials (
5	atf	Type List	Bureau of Alcohol, Tobacco, and FirearmsDate: 2008
6	biom	Type List	Biometric Schema Version 1.0
7	can	Type List	Province codes for Canada.Publication: Canada Post Ad
8	cbrncl	Type List	Radiological and Nuclear Code ListPublication: CBRN do
9	census	Type List	County codes from US Census and ANSI. Publication: IN
10	cjis	Type List	FBI Criminal Justice Information System
11	commodity	Type List	Census and Dept of Transportation.Publication: CFS-120
12	dea	Type List	Drug Enforcement Administration (DEA). Publication: Co
13	dod_jcs-pub2.0	Type List	Intelligence discipline codes.Source: DoD Joint Staff; Pul
14	dol	Type List	US Department of Labor occupation codes.Publication:
15	edxl_rm	Type List	Source: EDXL Resource Messaging (RM) Standards Work
16	em	Type List	Emergency Management
17	fips_5-2	Type List	Source: National Institute for Standards and Technology
18	have-codes	Type List	Source: OASIS Emergency Data Exchange Language (EDX
19	hazmat	Type List	Source: US Department of Transportation (DoT)Pipeline
20	hl7	Type List	Source: Health (HL-7); Publication: Religion-HL7 Table 00
21	hs	Type List	Human Services
22	im	Type List	Immigration
23	ip	Type List	Infrastructure Protection

Type index tab

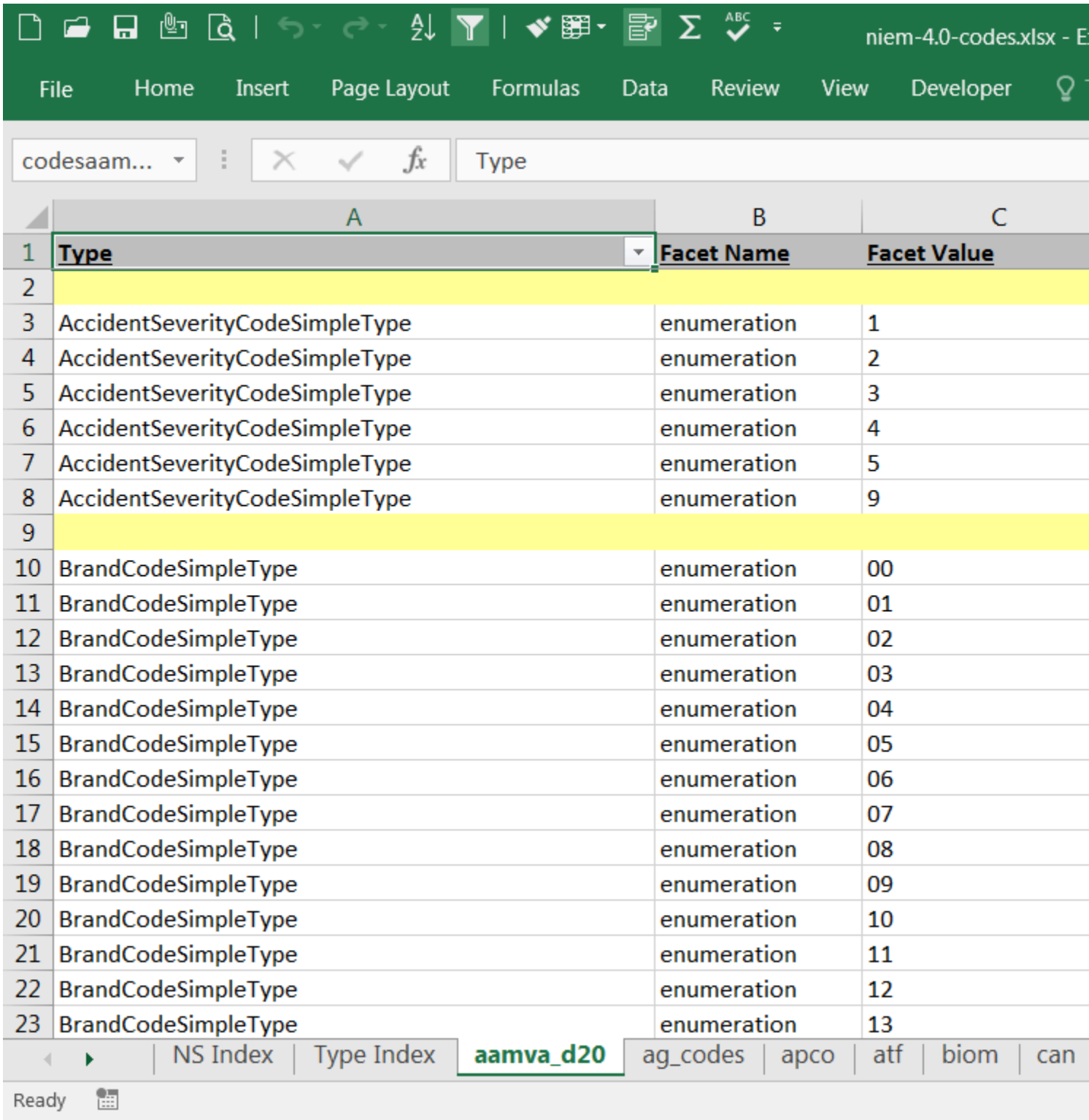
The second tab is a full index of the code types for each namespace. Click on the type name to go directly to the codes for that type.



	NS	Type Name	Typ
35	aamva_d20	RegistrationPlateColorCodeSimpleType	A d
36	aamva_d20	RegistrationStatusCodeSimpleType	A d
37	aamva_d20	RegistrationStatusDetailCodeSimpleType	A d
38	aamva_d20	SmogCertificateCodeSimpleType	A d
39	aamva_d20	TitleStatusCodeSimpleType	A d
40	aamva_d20	VehicleFuelCategoryCodeSimpleType	A d
41	aamva_d20	VehicleOdometerReadingUnitCodeSimpleType	A d
42	aamva_d20	VehicleTitleDocumentCategoryCodeSimpleType	A d
43	aamva_d20	VehicleTransmissionCategoryCodeSimpleType	A d
44	aamva_d20	VehicleUseCodeSimpleType	A d
45			
46	ag_codes	CLUIDSimpleType	A d
47	ag_codes	CoordinateReferenceSystemCodeSimpleType	A d
48	ag_codes	CoreProductCategorySimpleType	A d
49	ag_codes	CoreProductSimpleType	A d
50	ag_codes	CroppingPracticeCodeSimpleType	A d
51	ag_codes	CropYearSimpleType	A d
52	ag_codes	FarmNumberIDSimpleType	A d
53	ag_codes	FieldNumberIDSimpleType	A d
54	ag_codes	FileCustomerIDSimpleType	A d
55	ag_codes	FinalReportedAcreageSimpleType	A d
56	ag_codes	GeospatialDataSubmissionMethodCodeSimpleType	A d

Code tabs

The remaining tabs represent each NIEM namespace that defines codes.



Type	Facet Name	Facet Value
AccidentSeverityCodeSimpleType	enumeration	1
AccidentSeverityCodeSimpleType	enumeration	2
AccidentSeverityCodeSimpleType	enumeration	3
AccidentSeverityCodeSimpleType	enumeration	4
AccidentSeverityCodeSimpleType	enumeration	5
AccidentSeverityCodeSimpleType	enumeration	9
BrandCodeSimpleType	enumeration	00
BrandCodeSimpleType	enumeration	01
BrandCodeSimpleType	enumeration	02
BrandCodeSimpleType	enumeration	03
BrandCodeSimpleType	enumeration	04
BrandCodeSimpleType	enumeration	05
BrandCodeSimpleType	enumeration	06
BrandCodeSimpleType	enumeration	07
BrandCodeSimpleType	enumeration	08
BrandCodeSimpleType	enumeration	09
BrandCodeSimpleType	enumeration	10
BrandCodeSimpleType	enumeration	11
BrandCodeSimpleType	enumeration	12
BrandCodeSimpleType	enumeration	13

Change log spreadsheet

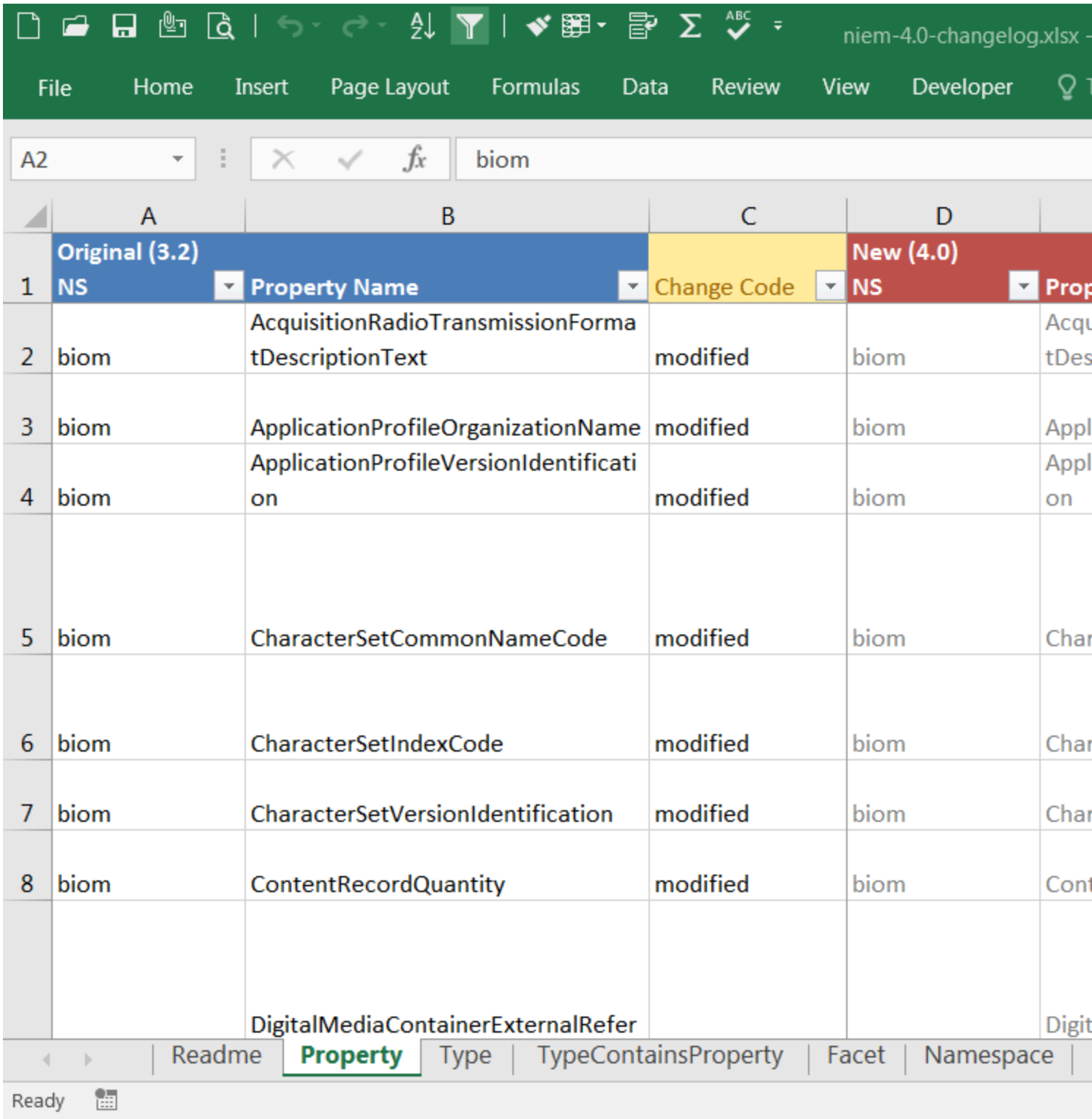
The NIEM change log spreadsheet documents the changes between the given release and the previous one.

Items from the previous release are identified in the left-most columns, with blue column headers. Change information is provided under the yellow column header. And items from the current release are provided in the right-most columns, under the red column headers.

For items marked as modified, the fields that have changed are in the standard black font. Fields that remain the same as in the previous release are in a lighter gray font.

Note that not all changes are recorded with the same amount of details. There are cases where changes to a component do not show up as an edit, but as separate delete and add operations.

This file is located in the root of the package or repo. The file name is `niem-version-changelog.xlsx`.



	Original (3.2)			New (4.0)	
1	NS	Property Name	Change Code	NS	Prop
2	biom	AcquisitionRadioTransmissionFormatDescriptionText	modified	biom	Acqu tDes
3	biom	ApplicationProfileOrganizationName	modified	biom	Appl
4	biom	ApplicationProfileVersionIdentification	modified	biom	Appl on
5	biom	CharacterSetCommonNameCode	modified	biom	Char
6	biom	CharacterSetIndexCode	modified	biom	Char
7	biom	CharacterSetVersionIdentification	modified	biom	Char
8	biom	ContentRecordQuantity	modified	biom	Cont
		DigitalMediaContainerExternalRefer			Digit

CSVs

NIEM CSVs provide an alternate, non-authoritative representation of the NIEM schema content in a tabular format. They are made available to support developers who wish to import the model content without having to parse the schemas.

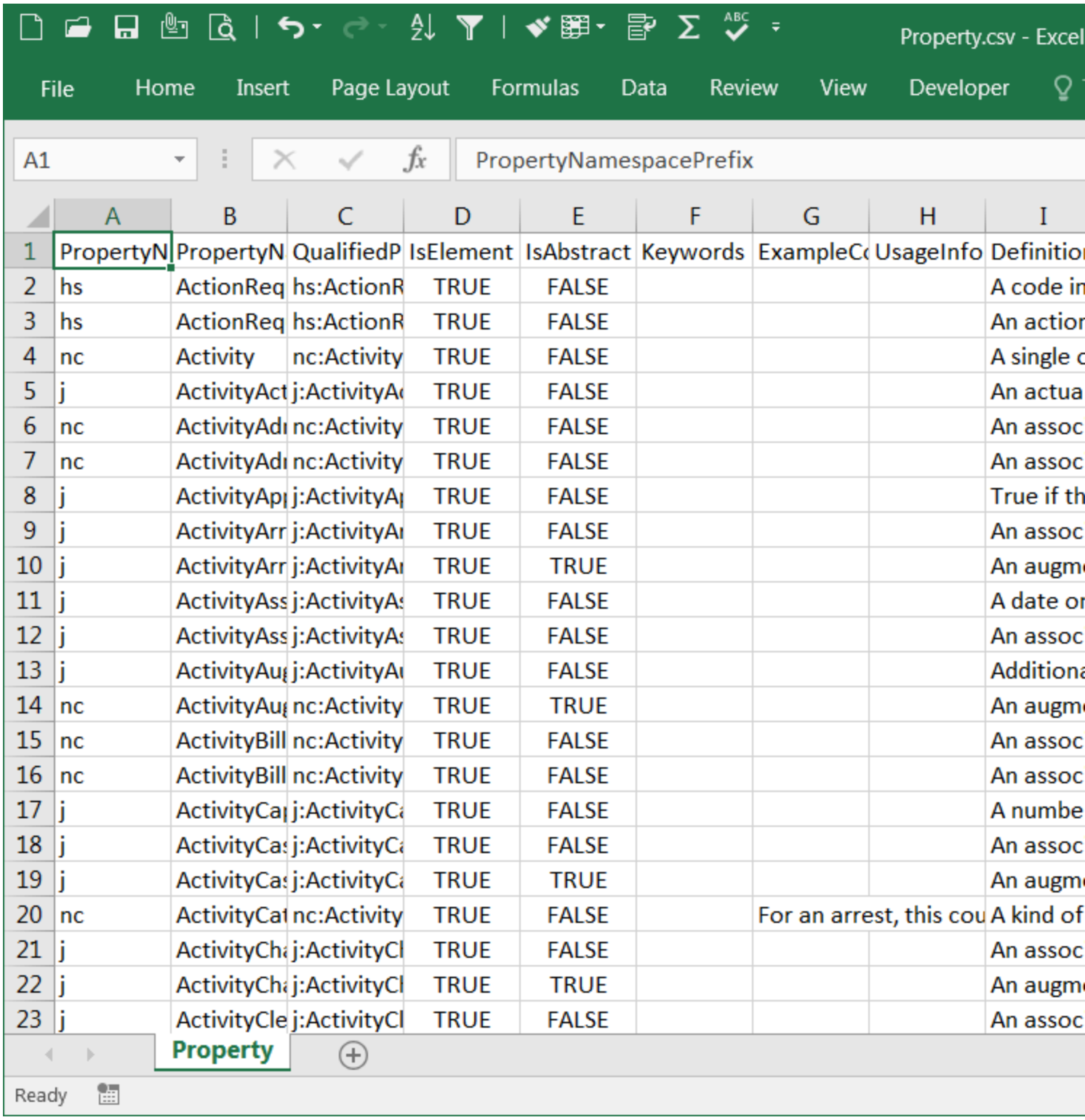
These materials are different from the spreadsheets in the release package. The other spreadsheets are formatted to provide user-readable documentation. These files are formatted to be machine-readable, so that the contents may be **easily imported**.

NIEM CSVs provide information about the content in a release:

- properties
- types
- types with their sub-properties
- type unions
- facets
- namespaces
- local terminology

On GitHub, the CSV formats are located in the `csv` directory off the root. The release page on release.niem.gov contains CSVs, plus two additional formats representing the same information: a spreadsheet and an Access database.

Below is a snapshot from the Property CSV:



	A	B	C	D	E	F	G	H	I
1	PropertyN	PropertyN	QualifiedP	IsElement	IsAbstract	Keywords	ExampleCo	UsageInfo	Definitio
2	hs	ActionReq	hs:ActionR	TRUE	FALSE				A code in
3	hs	ActionReq	hs:ActionR	TRUE	FALSE				An action
4	nc	Activity	nc:Activity	TRUE	FALSE				A single c
5	j	ActivityAct	j:ActivityA	TRUE	FALSE				An actua
6	nc	ActivityAd	nc:Activity	TRUE	FALSE				An assoc
7	nc	ActivityAd	nc:Activity	TRUE	FALSE				An assoc
8	j	ActivityAp	j:ActivityA	TRUE	FALSE				True if th
9	j	ActivityArr	j:ActivityA	TRUE	FALSE				An assoc
10	j	ActivityArr	j:ActivityA	TRUE	TRUE				An augm
11	j	ActivityAss	j:ActivityA	TRUE	FALSE				A date or
12	j	ActivityAss	j:ActivityA	TRUE	FALSE				An assoc
13	j	ActivityAu	j:ActivityA	TRUE	FALSE				Additiona
14	nc	ActivityAu	nc:Activity	TRUE	TRUE				An augm
15	nc	ActivityBill	nc:Activity	TRUE	FALSE				An assoc
16	nc	ActivityBill	nc:Activity	TRUE	FALSE				An assoc
17	j	ActivityCa	j:ActivityC	TRUE	FALSE				A numbe
18	j	ActivityCa	j:ActivityC	TRUE	FALSE				An assoc
19	j	ActivityCa	j:ActivityC	TRUE	TRUE				An augm
20	nc	ActivityCat	nc:Activity	TRUE	FALSE		For an arrest, this cou		A kind of
21	j	ActivityCh	j:ActivityC	TRUE	FALSE				An assoc
22	j	ActivityCh	j:ActivityC	TRUE	TRUE				An augm
23	j	ActivityCle	j:ActivityC	TRUE	FALSE				An assoc

Other artifacts

Other standard artifacts in a NIEM release are:

- Code Lists

NIEM Code Lists are special files that support the use of enhanced code lists in NIEM, beyond the basic capabilities provided by XML and JSON schema enumerations.

- XML Catalog

  An XML catalog is a XML document that assigns locations to files. This can be used to override the file locations assigned by NIEM XML schema import statements without having to modify the original schema itself.

- JSON-LD Context

  A JSON-LD context file is a file that can be used to associate namespace prefixes with the full URIs that they represent.

Specifications

NIEM specifications provide rules and guidance in order to design consistent and well-defined information exchanges. Specifications are managed by the NIEM Technical Architecture Committee (NTAC).

View and search rules from the specifications below in the **NIEM Specification Rule Browser**.

Announcements

The final 5.0 version of the NIEM JSON specification has just been released.

- NIEM Naming and Design Rules (NDR) Specification

  The NIEM Naming and Design Rules (NDR) describe the architecture of the NIEM data model and its representation in XML. It specifies principles and enforceable rules for NIEM data components and schemas.

- NIEM Information Exchange Package Documentation (IEPD) Specification

  The NIEM Information Exchange Package Documentation (IEPD) Specification specifies normative rules and non-normative guidance for building NIEM information exchange messages. It defines IEPD artifacts like subset schemas, extension schemas, and IEPD catalogs; and recommends how the package should be structured.

- NIEM JSON Specification

  The NIEM JSON Specification establishes the technical basis for using JavaScript Object Notation (JSON) as a data format for exchange of information consistent with NIEM-conformant schemas and message specifications.

- NIEM Conformance Specification

  The Conformance Specification specifies general conformance guidance, principles, and rules for NIEM.

- NIEM Conformance Targets Attribute Specification

  The Conformance Targets Attribute Specification (CTAS) defines how NIEM XML documents classify what kind of artifact they are through the use of a conformance targets attribute.

- NIEM Code Lists Specification

  The Code Lists Specification adds support for new capabilities of NIEM code lists beyond the basic enumeration representations provided by XML and JSON schema. Key features include the definition of codes in CSV or Genericode files, dynamic

code lists via run-time binding, and multi-column code table support.

- NIEM High-Level Version Architecture (HLVA) Specification

  The NIEM High-Level Version Architecture (HLVA) Specification identifies the processes, artifacts, and responsibilities required to produce new releases of the NIEM model. It also establishes a regular release cycle for predictable and manageable NIEM updates.

NIEM Naming and Design Rules (NDR) Specification

The **NIEM Naming and Design Rules (NDR)** describes the architecture of the NIEM data model and its representation in XML. It specifies principles and enforceable rules for NIEM data components and schemas.

- Overview
- Specification Resources
- Conformance Targets
  - REF
  - EXT
  - SET
  - INS
- Conformance Validation
- Example Rules
- Additional Resources

Overview

NDR rules are intended to establish and, more importantly, enforce a degree of standardization across a broad set of users in order to improve precision and interoperability across the NIEM community.

- A description of 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

Specification Resources

Submit a new issue, view all issues, track issue status, or go to the GitHub repo for this specification.

Available versions

- **Version 5.0**, published 2020. Use with NIEM 5.0 - NIEM 5.2. See changes. **[Current version]**
- **Version 4.0**, published 2017. Use with NIEM 4.0 - NIEM 4.2.
- **Version 3.0**, published 2014. Use with NIEM 3.0 - NIEM 3.2.
- **Version 1.3**, published 2008. Use with NIEM 2.0 - NIEM 2.1.

Conformance Targets

The NDR describes and defines rules for four different kinds of documents, each identified by its own

NDR conformance target.

Refer to the section about the Conformance Targets Attribute Specification for more information, including how to specify a conformance target in a NIEM schema.

Code	Conformance Target	Description	Spec
REF	Reference Schema Document	A NIEM schema that provides authoritative definitions of broadly reusable schema components and follows a stricter syntax. NIEM release schemas are REFs.	Link
EXT	Extension Schema Document	A NIEM schema for special-purpose content with little reuse expected. EXTs may thus follow a slightly more relaxed syntax. IEPD extension schemas are often EXTs.	Link
SET	Conformant Schema Document Set	A single conceptual XML schema that represents the sum of all rules and definitions within a collection of NIEM REF and EXT schemas, such as an IEPD.	Link
INS	Conformant Instance Document	An XML instance document which is valid against a conformant schema document set (SET).	Link

REF

Reference schema documents (REFs) are NIEM schemas that provide authoritative definitions of broadly reusable schema components and follow a stricter syntax, as specified by the NDR. NIEM release schemas are REFs.

The conformance rules for reference schema documents are generally stricter than those for extension schema documents (EXTs). This is because REFs are intended for reuse and must be defined in a way that supports a broad variety of requirements.

- Has a uniform document structure.
- Does not restrict other data definitions.
- Does not use XML Schema's restriction mechanism `xs:restriction`.
- Does not use schema groups such as `xs:choice` or `xs:any`.
- Requires augmentation points to support additional content from other namespaces
- Typically contains data components with relaxed cardinality (0 to unbounded)
- Made as regular and simple as possible.

EXT

Extension Schema Documents (EXTs) are NIEM schemas for special-purpose content that is not expected to be reused outside of the specific package in which it is defined. EXTs may thus follow a slightly more relaxed syntax, as specified by the NDR. IEPD extension schemas are usually EXTs, but may be designated as REFs instead if needed.

Extension schema documents (EXTs), like reference schema documents (REFs), are a fundamental part of NIEM model design. They contain components that use, extend, or limit the components defined in REFs and express additional vocabulary above and beyond the vocabulary available from REFs.

- Defines package-specific data elements and models.
- Extends vocabulary above and beyond that available to REFs.
- Can augment existing NIEM types with additional content.
- Can limit the behavior of existing NIEM data types through schema `xs:restriction`.
- Can constrain cardinality for specific requirements
- Can serve as the core schematic definitions of an IEPD.

SET

A **Conformant Schema Document Set** (SET) is a single conceptual XML schema that represents the sum of the rules and definitions defined within a collection of NIEM REF and EXT XML schemas, such as an IEPD.

- Conceptual artifact for collection of schemas
- Resolves to exactly one Schema
- Capable of validating Schema Instance Documents
- Self-contained, with no unresolved references

Because schema document sets are composed of NIEM-conformant schema documents (either REF or EXT), each member document must adhere to the rules designated by their individual conformance targets. Furthermore, the document set must also adhere to the rules applicable to the conformance target SET.

A SET defines a schema that may be used to validate an instance document. This rule ensures that a SET under consideration contains definitions for everything that it references; it has everything necessary to do a complete validation of instance documents without any unresolved references.

INS

An **Conformance Instance XML Document** (INS) is an XML instance document which is valid against a conformant schema document set (SET).

- Document element is locally schema-valid
- Each element of the document is defined by a conformant REF or EXT schema

Conformance Validation

While some rules in the NDR require subjective judgment (like determining if component names and definitions are semantically meaningful), other rules are capable of being automated. Starting with NDR version 3.0, such rules have been written as Schematron rules. Schematron is an ISO specification that describes a validation language for assertions about XML documents.

Schematron representations of NDR rules provide several key benefits. There is a single, normative interpretation of each rule. This eliminates the possibility that different users might read read a free-text rule description and apply it in different ways.

Schematron rules can also be used to validate XML documents. This makes it *substantially* easier for users to test NIEM REF and EXT XML schemas against the NDR to see if those schemas conform to

the given NDR conformance target.

> Two common ways to test NDR conformance are to upload schemas to **ConTesA** (online tool) or use the NDR rules with the **Schematron validation** functionality locally within the Oxygen XML Editor.

Example Rules

Refer to the **NDR** for the full set of rules. Descriptions, principles, and/or examples often accompany each rule. The NDR Table of Contents and the rules index in the Appendix can be used to jump directly to particular topics.

See the `Modeling` and `XML` sub-pages in the Concepts section to a collection of rules relating to particular topics. For example, the Namespaces XML page contains a list of NDR rules related to a namespace and its representation in NIEM XML (a XML schema).

Example Rule

The image below shows a small excerpt from the NDR.

- Each rule shows which conformance targets they apply to. In these cases, both rules apply to both REF and EXT schemas because they are labeled as `(REF, EXT)`
- `Rule 10-47` is a descriptive, free-text rule. Conformance for this rule must be checked manually.
- `Rule 10-48` is written in Schematron and can be tested via ConTesA or Oxygen.

10.8.1. Character case

Names of conformant components use *camel case* formattin
an initial lowercase or uppercase letter, with each remaining
initial uppercase letter, and *lowerCamelCase* is written with

Rule 10-47. Names use camel case

[Rule 10-47] (REF, EXT) (Constraint)

The name of any XML Schema component defined b

Rule 10-48. Attribute name begins with lower case letter

[Rule 10-48] (REF, EXT) (Constraint)

```
<sch:pattern>
  <sch:rule context="xs:attribute[exists(@na
    <sch:assert test="matches(@name, '^[a-z]
      >Within the schema, any attribute decl
      ('a'-'z').</sch:assert>
  </sch:rule>
</sch:pattern>
```

Additional Resources

See more information about the Naming and Design Rules Specification:

- Naming and Design Rules Change History

  The NIEM Naming and Design Rules Specification is updated with major NIEM releases, occurring every three years. The following highlights the changes that have been made.

- Reference vs. Extension Schema

  The NIEM Naming and Design Rules defines two classes of conformant XML Schema documents: reference schema documents (REF) and extension schema documents (EXT). View the key rule differences between the two conformance targets.

Naming and Design Rules Change History

The NIEM Naming and Design Rules Specification is updated with major NIEM releases, occurring every three years. The following highlights the changes that have been made.

- NDR 5.0 Changes
  - New rules
  - Modified rules
- NDR 4.0 Changes
- NDR 3.0 Changes

NDR 5.0 Changes

The NIEM Naming and Design Rules (NDR) specification was updated from version 4.0 to version 5.0 in 2020, corresponding with the NIEM 5.0 major release.

Key changes include

- Simplified the folder layout in the release package, removing version folders and unnecessary nesting (#140)
- Changed the character encoding of the XML schemas from “US-ASCII” to “UTF-8” for better international support (#125)
- Added attribute xml:lang to reference schemas (#153)
- Updated structures and appinfo utility schemas to version 5.0 based on NDR updates (#157)
- Reestablished the structures:sequenceID attribute (#154)
- Updated the conformance targets to correspond with the new 5.0 version of the NDR

See highlighted changes of the specification text.

New rules

Several new rules have been added to the NDR for 5.0:

Rule 5-1 (INS): structures:uri denotes resource identifier

> The interpretation of a conformant instance XML document 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.

Rule 10-45 (REF, EXT): Schema component name has xml:lang

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.

Rule 11-2 (REF, EXT): Only types have name ending in Type or SimpleType

A schema component with a name that ends in 'SimpleType' MUST be a simple type definition.

Rule 11-30 (REF, EXT): xs:documentation has xml:lang

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.

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

Given two properties of object $Object,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.

Modified rules

Modified rule conformance targets

In NDR 4.0, the following rule only applied to REF. In 5.0, it is now applied to EXT as well.

Rule 10-70 (REF, EXT): External adapter type indicator annotates complex type

The attribute {http://release.niem.gov/niem/appinfo/5.0/}externalImportIndicator MUST be owned by an element xs:import.

Modified rule numbers

Due to the addition of the new rules outlined above, inserted throughout the specification, some rule numbers from the NDR 4.0 version have shifted in the 5.0 version.

Expand to view the 93 adjusted rule numbers

NDR 4.0	NDR 5.0	Applicability	Title
10-45	10-46	REF, EXT	Schema component names have only specific characters

10-46	10-47	REF, EXT	Punctuation in component name is a separator
10-47	10-48	REF, EXT	Names use camel case
10-48	10-49	REF, EXT	Attribute name begins with lower case letter
10-49	10-50	REF, EXT	Name of schema component other than attribute and proxy type begins with upper case letter
10-50	10-51	REF, EXT	Names use common abbreviations
10-51	10-52	REF, EXT	Local term declaration is local to its schema document
10-52	10-53	REF, EXT	Local terminology interpretation
10-53	10-54	REF, EXT	Singular form is preferred in name
10-54	10-55	REF, EXT	Present tense is preferred in name
10-55	10-56	REF, EXT	Name does not have nonessential words
10-56	10-57	REF, EXT	Element or attribute name follows pattern
10-57	10-58	REF, EXT	Object-class term identifies concrete category
10-58	10-59	REF, EXT	Property term describes characteristic or subpart
10-59	10-60	REF, EXT	Name may have multiple qualifier terms
10-60	10-61	REF, EXT	Name has minimum necessary number of qualifier terms
10-61	10-62	REF, EXT	Order of qualifiers is not significant
10-62	10-63	REF, EXT	Redundant term in name is omitted
10-63	10-64	REF, EXT	Element with simple content has representation term
10-64	10-65	REF, EXT	Element with complex content has representation term when appropriate
10-65	10-66	REF, EXT	Element with complex content has representation term only when appropriate
10-66	10-67	REF, EXT	Machine-readable annotations are valid
10-67	10-68	REF, EXT	Component marked as deprecated is deprecated component
10-68	10-69	REF, EXT	Deprecated annotates schema component
10-69	10-70	REF, EXT	External import indicator annotates import
10-70	10-71	REF, EXT	External adapter type indicator annotates complex type
10-71	10-72	REF, EXT	appinfo:appliesToTypes annotates metadata element
10-72	10-73	SET	appinfo:appliesToTypes references types
10-73	10-74	REF, EXT	appinfo:appliesToElements annotates metadata element
10-74	10-75	SET	appinfo:appliesToElements references elements
10-75	10-76	REF, EXT	appinfo:LocalTerm annotates schema
10-76	10-77	REF, EXT	appinfo:LocalTerm has literal or definition
10-77	10-78	REF, EXT, INS, SET	Use structures consistent with specification
11-2	11-3	REF, EXT	Base type definition defined by conformant schema
11-3	11-4	REF, EXT	Name of simple type ends in SimpleType
11-4	11-5	REF, EXT	Use lists only when data is uniform
11-5	11-6	REF, EXT	List item type defined by conformant schemas
11-6	11-7	REF, EXT	Union member types defined by conformant schemas
11-7	11-8	REF, EXT	Name of a code simple type ends in CodeSimpleType
11-8	11-9	REF, EXT	Code simple type corresponds to a code list
11-9	11-10	REF, EXT	Attribute of code simple type has code representation term
11-10	11-11	REF, EXT	Complex type with simple content has

			structures:SimpleObjectAttributeGroup
11-11	11-12	REF, EXT	Element type does not have a simple type name
11-12	11-13	REF, EXT	Element type is from conformant namespace
11-13	11-14	REF, EXT	Name of element that ends in Abstract is abstract
11-14	11-15	REF, EXT	Name of element declaration with simple content has representation term
11-15	11-16	SET	Name of element declaration with simple content has representation term
11-16	11-17	REF, EXT	Element substitution group defined by conformant schema
11-17	11-18	REF, EXT	Attribute type defined by conformant schema
11-18	11-19	REF, EXT	Attribute name uses representation term
11-19	11-20	REF, EXT	Element or attribute declaration introduced only once into a type
11-20	11-21	REF, EXT	Element reference defined by conformant schema
11-21	11-22	REF, EXT	Referenced attribute defined by conformant schemas
11-22	11-23	REF, EXT	Schema uses only known attribute groups
11-23	11-24	REF, EXT	Data definition does not introduce ambiguity
11-24	11-25	REF, EXT	Object class has only one meaning
11-25	11-26	REF, EXT	Data definition of a part does not redefine the whole
11-26	11-27	REF, EXT	Do not leak representation into data definition
11-27	11-28	REF, EXT	Data definition follows 11179-4 requirements
11-28	11-29	REF, EXT	Data definition follows 11179-4 recommendations
11-29	11-31	REF, EXT	Standard opening phrase for augmentation point element data definition
11-30	11-32	REF, EXT	Standard opening phrase for augmentation element data definition
11-31	11-33	REF, EXT	Standard opening phrase for metadata element data definition
11-32	11-34	REF, EXT	Standard opening phrase for association element data definition
11-33	11-35	REF, EXT	Standard opening phrase for abstract element data definition
11-34	11-36	REF, EXT	Standard opening phrase for date element or attribute data definition
11-35	11-37	REF, EXT	Standard opening phrase for quantity element or attribute data definition
11-36	11-38	REF, EXT	Standard opening phrase for picture element or attribute data definition
11-37	11-39	REF, EXT	Standard opening phrase for indicator element or attribute data definition
11-38	11-40	REF, EXT	Standard opening phrase for identification element or attribute data definition
11-39	11-41	REF, EXT	Standard opening phrase for name element or attribute data definition
11-40	11-42	REF, EXT	Standard opening phrase for element or attribute data definition
11-41	11-43	REF, EXT	Standard opening phrase for association type data definition
11-42	11-44	REF, EXT	Standard opening phrase for augmentation type data definition
11-43	11-45	REF, EXT	Standard opening phrase for metadata type data definition
11-44	11-46	REF, EXT	Standard opening phrase for complex type data definition
11-45	11-47	REF, EXT	Standard opening phrase for simple type data definition
11-46	11-48	REF, EXT	Same namespace means same components

11-47	11-49	REF, EXT	Different version means different view
11-48	11-50	SET	Reference schema document imports reference schema document
11-49	11-51	SET	Extension schema document imports reference or extension schema document
11-50	11-52	REF, EXT	Structures imported as conformant
11-51	11-53	REF, EXT	XML namespace imported as conformant
11-52	11-54	SET	Each namespace may have only a single root schema in a schema set
11-53	11-55	REF, EXT	Consistently marked namespace imports
12-10	12-11	INS	Metadata applies to referring entity
12-11	12-12	INS	Referent of structures:relationshipMetadata annotates relationship
12-12	12-13	INS	Values of structures:metadata refer to values of structures:id
12-13	12-14	INS	Values of structures:relationshipMetadata refer to values of structures:id
12-14	12-15	INS	structures:metadata and structures:relationshipMetadata refer to metadata elements
12-15	12-16	INS	Attribute structures:metadata references metadata element
12-16	12-17	INS	Attribute structures:relationshipMetadata references metadata element
12-17	12-18	INS	Metadata is applicable to element

NDR 4.0 Changes

The NIEM Naming and Design Rules (NDR) specification was updated from version 3.0 to version 4.0 in 2017, corresponding with the NIEM 4.0 major release.

Updated version from 3.0 to 4.0, affecting:

- Namespaces: structures, appinfo, NDR functions namespace
- Conformance target identifiers: REF, EXT

Now managing specification on GitHub (https://github.com/NIEM/NIEM-NDR/) for:

- Revisions/updates/errata: posted to the document repository
- Comments/issues: posted as GitHub issues

NDR Section 2.4.2 - Schematron updates

- Eliminated warning rules (`sch:report`) that always fire
- Added attribute (`role="warning"`) to warning rules
- Broke apart overly-large rule on standard opening phrases into smaller rules

NDR Section 5 - RDF updates

- Simplified RDF representation
  - RDF is much simpler than v3, with direct properties instead of reification
  - Allows for more direct JSON representation via JSON-LD
- Removed RDF representation for metadata types

NDR Section 6 & 8 - XML processing and defaults

- Improved discussion of infoset augmentation, `fixed`, and `default`
- Allow use of fixed on attribute uses that are required in an instance

NDR Section 10 & 11 - Naming

- Relaxed rules on component naming from MUST to SHOULD (USMTF)
- Allow all valid ASCII characters in component names
    - Allowed: “A”-“Z”, “a”-“z”, “0”-“9”, “-“, “_”, “.”

NDR Section 10 & 11 - Codes

- Allow code elements and code types to be represented by methods other than enumerations
- Described as correspondence to a list of conceptual entities
- Relaxed rules for structure/content of code elements and types
- Relaxed naming rules for code elements and types

NDR Section 12 - Linked data approach

- Added attribute `structures:uri`
- Defined `structures:id` and `structures:ref` in terms of `structures:uri` (Appendix B)
- Allow properties of objects to be distributed across multiple objects with the same identifier
    - Removed requirement that elements with `structures:ref` have no properties
- Major rewrite to section 12.1, 12.2, addressing data’s meaning, identity, and references

Miscellaneous

- Grammar, spelling, and narrative improvements
- Moved content of the *Local Terminology* namespace (`term`) into the *Application Information* namespace (`appinfo`)
- Explicitly allow structures namespace to be subset

(See full diffs)

NDR 3.0 Changes

The NIEM Naming and Design Rules (NDR) specification was updated from version 1.3 to version 3.0 in 2014, corresponding with the NIEM 3.0 major release.

Changes include:

- Replaced conformant indicator with conformance targets for reference schema, extension schema, conformant schema document set and conformance instance XML document.

- Updated the approach used for augmentations. Replaced the use of type substitution with

element substitution, and added augmentation point elements to object and association types.

- Added support for local vocabularies.
- Added structures:ref and merged content and reference elements.
- Added NDR Schematron rules to normatively define and validate conformance where possible.
- Replaced ComplexObjectType with ObjectType and AssociationType.
- Added multiple representations of code lists (code, literal).
- Removed appinfo base.
- Removed sequenceID, augmentation and metadata elements.
- Replaced external adaptor type indicator with an attribute.
- Added external import indicator.
- Replaced applies-to element with substitution groups for augmentations and applies-to types and applies-to elements for metadata.
- Replaced link metadata with relationship metadata.
- Replaced appinfo annotation elements with attributes.
- Replaced reference target annotations with XML schema type attribute.

Reference vs. Extension Schema

The NIEM Naming and Design Rules version 5.0 defines two classes of conformant XML Schema documents: reference schema documents (REF) and extension schema documents (EXT). The following outlines the differences in NDR rules between REF and EXT schema documents.

- Background
- Conformance targets
- Extension and restriction
- Block and final
- Nillable
- Choice
- Wildcards
- Augmentation points
- External adapters

Background

Beginning with NDR 3.0, NIEM XML schemas must declare themselves as either a reference schema document or an extension schema document through the use of the schema-level conformance targets attribute.

- Reference schema document (REF): A schema document that is intended to provide the authoritative definitions of broadly reusable schema components. NIEM release schemas are REFs because they are intended for wide reuse.

- Extension schema document (EXT): A schema document that is intended to provide definitions of schema components that are intended for reuse within a more narrow scope than those defined by a reference schema document. IEPD extension schemas are commonly EXTs so they can meet specific exchange requirements as needed, such as precise cardinality.

Conformance targets

REF and EXT schema documents have different conformance target identifiers. All REF and EXT schema documents must have a conformance target identifier, which tells readers of the schema what class of schema document it is. The conformance target identifiers are different for REF and EXT. These two rules identify the conformance target for each type of schema document.

- Rule 4-5, Schema claims reference schema conformance target (REF)
- Rule 4-6, Schema claims extension conformance target (EXT)

Extension and restriction

REF schema documents must use `xs:extension` to derive complex types. EXT schema documents are allowed to use `xs:restriction` for complex types. These rules require `xs:extension` for REF schemas. EXT schema documents are allowed to use `xs:extension` and `xs:restriction`.

- Rule 9-30, Complex content uses extension (REF)
- Rule 9-33, Simple content uses extension (REF)

These rules ensure that REF and EXT have `xs:sequence` as the top-level model compositors; they are different rules because REF requires `xs:extension`, while EXT also allows `xs:restriction`

- Rule 9-62, xs:sequence must be child of xs:extension (REF)
- Rule 9-63, xs:sequence must be child of xs:extension or xs:restriction (EXT)

Block and final

REF schema documents are not allowed to use `block` and `final`, which restrict extension and reuse of elements and complex types. EXT schemas are allowed to use these. These restrictions are placed on elements, complex types and on the schema, where defaults are set.

- Rule 9-34, No complex type disallowed substitutions (REF)
- Rule 9-35, No complex type disallowed derivation (REF)
- Rule 9-43, No element disallowed substitutions (REF)
- Rule 9-44, No element disallowed derivation (REF)
- Rule 9-86, No disallowed substitutions (REF)
- Rule 9-87, No disallowed derivations (REF)

Nillable

REF elements must be nillable.

- Rule 9-47, Element declaration is nillable (REF)

Choice

EXT schemas are allowed to use `xs:choice`. REF schemas are not allowed to use `xs:choice`. (Neither type of schema can use `xs:all`).

- Rule 9-64, No xs:choice (REF)

Since REF schemas are not allowed to use `xs:choice`, these rules are applied only to EXT schemas.

- Rule 9-60, Model group does not affect meaning (EXT)
- Rule 9-65, xs:choice must be child of xs:sequence (EXT)

These rules limit cardinality on `xs:choice`, applicable only to EXT schema documents.

- Rule 9-68, Choice has minimum cardinality 1 (EXT)
- Rule 9-69, Choice has maximum cardinality 1 (EXT)

Wildcards

REF schema documents cannot use wildcards.

- Rule 9-70, No use of xs:any (REF)
- Rule 9-71, No use of xs:anyAttribute (REF)

Augmentation points

REF schema documents must include augmentations on complex types.

- Rule 10-23, Augmentable type has augmentation point element (REF)
- Rule 10-29, Augmentation point element use is optional (REF)
- Rule 10-30, Augmentation point element use is unbounded (REF)

External adapters

REF schemas are not allowed to use attributes from external (not NIEM-conformant) schemas. EXT schemas are allowed to reference external attributes.

- Rule 10-13, External attribute use only in external adapter type (REF)

Prior to NDR 5.0, the following rule applied only to REF. It has since been adjusted to apply to EXT as well, and is no longer a difference between the two targets.

- Rule 10-70, External adapter type indicator annotates complex type (REF)

NIEM Information Exchange Package Documentation (IEPD) Specification

The **NIEM Information Exchange Package Documentation (IEPD) Specification** specifies normative rules and non-normative guidance for building NIEM information exchange messages.

> This specification was previously known as the **NIEM Model Package Description (MPD) Specification**. For 5.0, the MPD specification was simplified to focus specifically on IEPDs (one of the specialized kinds of model packages) and renamed.

- Specification Resources
- Conformance Targets
- Artifacts
- Additional Resources

Specification Resources

Submit a new issue, view all issues, track issue status, or go to the GitHub repo for this specification.

Available versions

- **Version 3.0.1**, published 2015. Use with NIEM 3.0 - NIEM 4.2.
- **Version 5.0**, published 2021. Use with NIEM 5.0 - NIEM 5.2. See changes. **[Current version]**

Conformance Targets

The IEPD Specification includes the following conformance targets:

Code	Conformance Target	Description	Spec
IEPD	Information Exchange Package Documentation	A set of artifacts that define the content, structure, and meaning of an information exchange message. The schemas contained by the IEPD must conform to the NIEM Naming and Design Rules (NDR).	Link
WF-IEPD	Well-formed IEPD	An IEPD with a less-strict set of requirements. A well-formed IEPD focuses on the format and structure of the package, includes a valid IEPD catalog with no broken links, and includes and uses other general IEPD artifacts correctly; however, schema conformance to the NIEM Naming and Design Rules (NDR) is not required.	Link
IEP	Information Exchange Package	An information exchange message, as an XML instance document. The use of NIEM-defined elements is not required.	Link
FN-IEP	Full NIEM IEP	An information exchange message, as an XML instance document. Ths use of elements defined by	Link

		NIEM reference or extension schemas is required.	
Schema-subset	Schema document subset	A set of schemas, corresponding to a set of full reference schemas, that have been reduced and constrained to include only the files and components needed for an information exchange.	Link
IEPD-catalog	IEPD catalog document	A XML file that contains metadata about the IEPD, including descriptive information and an index of the files it contains.	Link
XML-catalog	XML catalog document	A XML file that maps namespaces to file locations.	Link

Artifacts

The IEPD Specification provides information about required and optional artifacts for an IEPD. These are described in the IEPD Artifacts page of the Reference / IEPD section.

Additional Resources

See more information about the IEPD Specification:

- IEPD Specification Change History

  The NIEM Information Exchange Package Documentation (IEPD) Specification can be updated with major NIEM releases, occurring every three years. The following highlights the changes that have been made.

IEPD Specification Change History

The NIEM Information Exchange Package Documentation (IEPD) Specification can be updated with major NIEM releases, occurring every three years. The following highlights the changes that have been made.

- Key changes
- IEPD catalog changes
- Conformance targets
- Removed rules
- Modified rules

Along with the NIEM 5.0 major release, the Model Package Description (MPD) Specification, version 3.0.1, was simplified, renamed, and updated to the IEPD Specification, version 5.0.

The MPD Specification was originally written to support multiple kinds of models, including releases, IEPDs, domain updates, and Enterprise Information Exchange Models (EIEMs). In practice, however, IEPDs were almost the sole use case so the specification in 5.0 was tailored specifically to IEPDs and the potentially confusing *model package description (MPD)* terminology was removed.

> There was no update to the MPD Specification for the NIEM 4.0 major release, so the 3.0.1 version applies to all IEPDs using a release from NIEM 3.0 to NIEM 4.2.

Key changes

Key changes to the 5.0 version of the specification include:

- Updated the name of the specification from “Model Package Description (MPD) Specification to Information Exchange Package Documentation (IEPD) Specification
- Replaced “MPD” terminology throughout the spec with “IEPD”.
- Renamed the `mpd-catalog.xml` and `mpd-catalog.xsd` files as `iepd-catalog.xml` and `iepd-catalog.xsd`.
- Updated the IEPD catalog element and attribute names to replace `mpd` with `iepd`
- Updated the example IEPD for 5.0

IEPD catalog changes

In the renamed `iepd-catalog.xsd` schema, the following elements and attributes have been renamed:

MPD 3.0.1	IEPD 5.0
c:Catalog/c:MPD	c:IEPDCatalog - *(renamed and flattened)*
c:MPDInformation	c:IEPDInformation

MPD 3.0.1	IEPD 5.0
c:mpdURI	c:iepdURI
c:mpdClassURIList	c:iepdConformanceTargetIdentifierURIList
c:mpdName	c:iepdName
c:mpdVersionID	c:iepdVersionID
c:MPDChangeLog	c:IEPDChangeLog

Please see the new IEPD catalog schema and IEPD catalog example for more.

Conformance targets

The following conformance target applicability codes have been updated in version 5.0:

- **WF-IEPD** (Well-formed IEPD) - *was **MPD** (Model Package Description)*
- **IEPD-catalog** (IEPD catalog document) - *was **MPD-catalog** (MPD catalog document)*

The following applicability codes remained unchanged:

- **IEPD** (Information Exchange Package Documentation)
- **IEP** (Information Exchange Package)
- **FN-IEP** (Full-NIEM IEP)
- **Schema-subset** (Schema document subset)
- **XML-catalog** (XML catalog document)

Removed rules

Two rules have been removed from the IEPD specification for version 5.0:

Rule 3-2 (MPD): MPD with MPD class of IEPD is an IEPD

> A model package description with an MPD class of http://reference.niem.gov/niem/specification/model-package-description/3.0/#IEPD MUST be an information exchange package documentation.

Rule 5-9 (WF-MPD): MPD Class Determined by Conformance Target Identifier in c:mpdClassURIList

> An MPD MUST have an MPD class of a conformance target identifier if and only if that conformance target identifier appears in the c:mpdClassURIList attribute within its MPD catalog document.

Modified rules

Added missing rule titles

Two rule titles were left blank in the MPD 3.0 specification, which has been corrected in the IEPD 5.0 specification:

MPD 3.0	IEPD 5.0	Applicability	Title
5-31	5-30	WF-IEPD	Schema document set interpreted as constraint schema document set
5-41	5-40	IEPD-Catalog	Validity contraint context is value of c:xPathText

Modified rule numbers

Due to the removal of the two 3.0 rules outlined above, some rule numbers from the MPD 3.0 version have shifted in the IEPD 5.0 version.

Expand to view the 39 adjusted rule numbers

MPD 3.0	IEPD 5.0	Applicability	Title
3-1	3-2	WF-IEPD	IEPD is well-formed
3-3	3-1	IEPD	IEPD conformance target identifier
5-10	5-9	WF-IEPD	IEPD Version Number Syntax
5-11	5-10	WF-IEPD	IEPD URI Is Absolute
5-12	5-11	WF-IEPD	IEPD URI Supports Fragment
5-13	5-12	WF-IEPD	IEPD URI Has No Fragment
5-14	5-13	WF-IEPD	IEPD Artifact URI Syntax
5-15	5-14	WF-IEPD	c:pathURI Resolves to a Resource
5-16	5-15	WF-IEPD	c:pathURI for c:XMLCatalog
5-17	5-16	WF-IEPD	c:pathURI for c:IEPDChangeLog
5-18	5-17	WF-IEPD	c:pathURI for c:ReadMe
5-19	5-18	WF-IEPD	c:pathURI for c:IEPSampleXMLDocument
5-20	5-19	WF-IEPD	c:pathURI for c:BusinessRulesArtifact
5-21	5-20	WF-IEPD	c:pathURI for c:XMLSchemaDocument
5-22	5-21	WF-IEPD	c:pathURI for c:ExternalSchemaDocument
5-23	5-22	WF-IEPD	c:pathURI for c:ReferenceSchemaDocument
5-24	5-23	WF-IEPD	c:pathURI for c:ExtensionSchemaDocument
5-25	5-24	WF-IEPD	c:pathURI for c:SubsetSchemaDocument
5-26	5-25	WF-IEPD	c:pathURI for c:Wantlist
5-27	5-26	WF-IEPD	c:pathURI for c:SchematronSchema
5-28	5-27	WF-IEPD	c:pathURI for c:RelaxNGSchema
5-29	5-28	WF-IEPD	c:pathURI for c:SchemaDocumentSet
5-30	5-29	WF-IEPD	c:pathURI for c:ConstraintSchemaDocumentSet
5-31	5-30	WF-IEPD	Schema document set interpreted as constraint schema document set
5-32	5-31	WF-IEPD	Resolve IEPD URI with Fragment
5-33	5-32	XML-catalog	XML Catalog uri Value Resolves to Resource
5-34	5-33	XML-catalog	XML Catalog uri Value Resolves to Resource with Correct Target Namespace
5-35	5-34	IEPD	IEPD Has a Change Log
5-36	5-35	WF-IEPD	Readme Describes Purpose, Scope, Business Value, etc.
5-37	5-36	IEPD	IEPD Has a ReadMe Artifact
5-38	5-37	IEPD-catalog	Conformance Target Identifier
5-39	5-38	IEPD-catalog	IEP Conformance Target Has a structures:id
5-40	5-39	IEPD	IEPD Declares One or More IEP Conformance Targets

5-41	5-40	IEPD-catalog	Validity contraint context is value of c:xPathText
5-42	5-41	IEP	IEP has Document Element
5-43	5-42	IEP	Validating an XPath Expression
5-44	5-43	IEPD	IEPD Has an IEP Sample for Each c:IEPConformanceTarget
5-45	5-44	IEP	Validating an IEP Sample XML Document
5-46	5-45	IEPD	IEPD Has Conformance Assertion

NIEM JSON Specification

The **NIEM JSON Specification** establishes the technical basis for using JavaScript Object Notation (JSON) as a data format for exchange of information consistent with NIEM-conformant schemas and message specifications.

- Specification Resources
- Conformance Targets
- Additional Resources

Specification Resources

Submit a new issue, view all issues, track issue status, or go to the GitHub repo for this specification.

Available versions

- **Version 5.0**, published 2021. Use with NIEM 5.0 - NIEM 5.2. See changes. **[Current version]**
- **Version 4.0**, published 2020. Use with NIEM 4.0 - NIEM 4.2.

Conformance Targets

NIEM-conformance of JSON data is primarily focused on the relationship between the data in the JSON file and the definitions established by a NIEM-conformant schema, e.g., the schema defined by a NIEM IEPD.

There are two forms of conformance of a NIEM JSON document to a NIEM-conformant schema:

Code	Conformance Target	Description	Spec
STRICT	NIEM JSON document strictly conformant to a schema	A NIEM JSON document strictly conformant to a schema is a JSON-LD document that may be assigned a one-to-one correspondence to a conformant instance XML document valid against a conformant schema document set.	Link
LAX	NIEM JSON document laxly conformant to a schema	A NIEM JSON document laxly conformant to a schema is a JSON-LD document that may be interpreted using the RDF vocabulary defined by a conformant schema document set.	Link

Additional Resources

- NIEM JSON normalization - Describes how a full NIEM JSON-LD document can be translated to a much more simple JSON representation.

- From the NIEM JSON section on this site, see especially:

    - Guidance in Using NIEM with JSON

- NIEM JSON tutorial

NIEM Conformance Specification

The **NIEM Conformance Specification** introduces NIEM conformance and provides a general normative definition for its meaning, how it applies, and to what it does and does not apply. It describes the benefits of conformance, and refers to sources of information, tools, and help.

- Specification Resources
- Conformance vs Compliance
- Conformance Targets
- NIEM Artifact Conformance
- NIEM Conformance Practices
- Support for NIEM-Conformance

The term *conformance* is often used very broadly, but it has a very specific meaning within NIEM. Artifacts, like schemas and IEPDs, can be built so that they conform to a specific conformance target. Conformance targets and their associated rules are defined by NIEM specifications.

System, tools, or databases may have capabilities designed to specifically support the development of NIEM-conformant artifacts, but those systems art not, themselves, NIEM-conformant. Such tools or systems are considered NIEM-aware or NIEM-supporting.

Specification Resources

Submit a new issue, view all issues, or go to the GitHub repo for this specification.

Available versions

- **Version 5.0**, published 2021. Use with NIEM 5.0 - NIEM 5.2. See changes. **[Current version]**
- **Version 3.0**, published 2014. Use with NIEM 3.0 - NIEM 4.2.
- **Version 1.0**, published 2008. Use with NIEM 1.0 - NIEM 2.1.

Conformance vs Compliance

NIEM-conformance does not imply NIEM compliance. Compliance indicates enforcement and the existence of an official certification process that verifies conformance or level of conformance. NIEM does not currently define compliance.

Early in the development of NIEM (and its predecessor), governance committees discussed the language to be used in solicitations (such grants, RFPs, etc.) that require use of NIEM. They considered both terms, and concluded that the term compliance does not convey appropriate intent; it is too regulatory (for example, U.S. citizens and residents must comply with the U.S. tax code). The objectives of NIEM are to facilitate and support interoperability, NOT to force or regulate it.

Conformance Targets

A `conformance target` is a special class of artifact, defined by a NIEM specification, with an

applicable set of normative rules.

- The NDR 5.0 specification defines 4 conformance targets, including
  - **Reference schema document (REF)**
  - **Extension schema document (EXT)**

- The IEPD 5.0 specification defines 7 conformance targets, including
  - **Information Exchange Package Documentation (IEPD)**
  - **IEPD catalog document (IEPD-catalog)**

- The Code Lists 4.0 specification defines 6 conformance targets, including
  - **CSV code list document (CSV-CLD)**

In addition to describing what a conformance target is, NIEM specifications assign unique IDs (conformance target identifiers) and define normative rules for each.

The images below show information about conformance target `reference schema document (REF)` from the NDR 4.0 specification.

- The first image shows the definition of the conformance target and its conformance target identifier.
- The second image shows two of the normative rules for the REF conformance target. Note that the first rule is free text and must be checked manually; the second rule is written in Schematron and can be validated using ConTesA.

[Definition: reference schema document]

A **reference schema document** is a ·schema docume
definitions of broadly reusable ·schema components·.
specification. A reference schema document MUST c
apply to this conformance target. An ·XML document
`http://reference.niem.gov/niem/specification/`
`rules/4.0/#ReferenceSchemaDocument` MUST be a

Rule 10-47. Names use camel case

[Rule 10-47] (REF, EXT) (Constraint)

The name of any XML Schema component defined by the
formatting convention.

Rule 10-48. Attribute name begins with lower case letter

[Rule 10-48] (REF, EXT) (Constraint)

```
<sch:pattern>
  <sch:rule context="xs:attribute[exists(@name)]"
    <sch:assert test="matches(@name, '^[a-z]')"
      >Within the schema, any attribute declarati
with a lowercase letter
      ('a'-'z').</sch:assert>
  </sch:rule>
</sch:pattern>
```

A NIEM artifact will declare its applicable conformance targets by referencing the associated conformance target identifier(s).

- An XML schema can declare the `Extension schema document` conformance target from the NDR 5.0 specification.
- An IEPD can declare the `IEPD` conformance target from the IEPD 5.0 specification.
- A CSV code list can declare the `CSV Code List Document` conformance target from the Code Lists 4.0 specification.

See the Conformance Targets Attribute Specification for more information on how this is done.

NIEM Artifact Conformance

An artifact is NIEM-conformant if and only if it:

1. has a named Conformance Target
2. adheres to all design and structural rules applicable to its Conformance Target
3. references the namespaces of any NIEM components used in its definition

NIEM Conformance Practices

NIEM developed a set of practices for information exchanges, which include:

1. Reuse existing NIEM data components when appropriate

Reuse NIEM elements and types whenever possible rather than creating new ones.

In an IEPD extension schema, this means reference NIEM elements and types that are declared in a full NIEM release or in a NIEM subset. Do not recreate elements in the extension schema with NIEM names and definitions.

2. Extend and augment NIEM data components as needed

If a type like `nc:PersonType` does not contain all of the properties that are needed for an exchange, create extensions or augmentations.

3. Construct new components when necessary

Construct new data components to express any concepts specific to an exchange or community which the NIEM model does not cover.

4. Use NIEM techniques for data modeling

Use NIEM techniques for common data modeling needs. For example, NIEM describes:

- references - create links between data items
- associations - express complex relationships that carry their own data
- augmentations - add characteristics to already-defined classes of objects
- adapters - carry non-NIEM data in a NIEM message
- roles - express a role or function of an entity

5. Make IEPDs precise

Make the specifications for information exchanges precise, so that tools and systems can support the use and comprehension of messages, as well as ensuring each messages obeys the rules of their specifications

6. Make IEPDs consistent and unambiguous

Construct messages so tools and systems can ensure they follow their specifications and reuse NIEM terms (namespaces and local names), so the identity and expected content of messages are consistent and unambiguous

Support for NIEM-Conformance

NIEM has practices, methods, and formats for the pieces that compose an information exchange. These are expressed as a set of specifications. Specifications define the rules and procedures for creating content which is NIEM-conformant.

We encourage developers to use tools provided by the NIEM program to construct and validate their information exchanges as NIEM-conformant, including:

- The NIEM Schema Subset Generation Tool (SSGT) to find and assemble NIEM data components into subset schemas.
- A schema development environment or editor to construct Extension Schemas and express additional concepts needed for an exchange. See the Concepts section for NIEM modeling techniques.
- Ensure IEPDs and schemas follow the NIEM rules by using the Conformance Testing Assistant (ConTesA).

NIEM Conformance Targets Attribute Specification

The **NIEM Conformance Targets Attribute Specification (CTAS)** defines how NIEM XML documents classify what kind of artifact they are. This is not always obvious by inspection, so making this information explicit is important; it also supports automated conformance validation by NIEM tools.

- Specification Resources
- Background
- The conformance targets schema
- Using the conformance targets attribute
- Common Conformance Targets

Specification Resources

Submit a new issue, view all issues, or go to the GitHub repo for this specification.

Available versions

- **Version 3.0**, published 2014. Use with NIEM 3.0 and later releases. **[Current version]**

Background

Certain NIEM specifications define rules for specific kinds of artifacts. The NDR defines rules for Reference Schema Documents and Extension Schema Documents. The Code List Specification defines rules for a Code List Document, a CSV Code List Document, and a Genericode Code List Document. In addition to defining the artifacts and the rules, these specifications also define conformance target identifiers for each kind of artifact. These are unique string values, formatted as a URL, and include information about the specification, the version, and the kind of artifact.

- The NDR, version 3.0, defines the following conformance target identifier for a Reference Schema Document: `http://reference.niem.gov/niem/specification/naming-and-design-rules/3.0/#ReferenceSchemaDocument`

- The Code Lists Specification, version 4.0, defines the following conformance target identifier for a CSV Code List Document: `http://reference.niem.gov/niem/specification/code-lists/4.0/#CSVCodeListDocument`

A NIEM XML document can describe itself as a certain kind of artifact by including special attribute `ct:conformanceTargets` with a value of one or more of the conformance target identifiers described above.

> All conformant NIEM 4.0 release schemas include the `ct:conformanceTargets` attribute with the value of the conformance target identifier specified by the NDR, version 4.0, for Reference Schema Documents:

```
ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-
```

If an XML document represents more than one kind of artifact, then all applicable conformance target identifiers can be listed together as the `ct:conformanceTargets` attribute value, with each identifier separated by whitespace.

The conformance targets schema

The `ct:conformanceTargets` attribute is defined by a published schema, conformanceTargets.xsd. This schema is included with each NIEM release starting with 3.0, and in each NIEM subset generated by the SSGT. Starting with the NIEM 4.0 release, the schema is located in its own folder under the niem/utility subdirectory.

The `conformanceTargets.xsd` schema defines a single attribute: `conformanceTargets`. Its data type is a simple list type for URI values.

Using the conformance targets attribute

In order for a NIEM XML schema to specify its conformance targets, it is necessary to:

- identify the `conformanceTargets.xsd` schema
- assign the schema a file location where it may be found
- include the `conformanceTargets` attribute with all applicable values

This is demonstrated in the example below from a 4.0 release schema:

```
<xs:schema
  xmlns:ct="http://release.niem.gov/niem/conformanceTargets/3.0/"

  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  xsi:schemaLocation="http://release.niem.gov/niem/conformanceTargets/3.0/
      ../../../utility/conformanceTargets/3.0/conformanceTargets.xsd"

  ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-a

<xs:schema>
```

- The namespace prefix `ct` is assigned to the identifier (target namespace) of the conformance targets schema.
- The namespace prefix `xsi` is represents the XML Schema instance namespace.
- The `xsi:schemaLocation` attribute assigned the target namespace of the conformance targets schema to a relative location of that file. The file location may also be defined outside of this schema using a XML Catalog.
- The `ct:conformanceTargets` attribute contains all applicable conformance target identifiers.

Common Conformance Targets

NDR 3.0 Reference Schema rules

Conformance targets attribute used by:

- any conformant schema from the 3.0-series of releases
- IEPD extension schema that follows the stricter NDR 3.0 rule set

```
ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-and
```

NDR 3.0 Extension Schema rules

Conformance targets attribute used by:

- IEPD extension schema that follows the more relaxed NDR 3.0 rule set

```
ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-and
```

NDR 4.0 Reference Schema rules

Conformance targets attribute used by:

- any conformant schema from the 4.0-series of releases
- IEPD extension schema that follows the stricter NDR 4.0 rule set

```
ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-and
```

NDR 4.0 Extension Schema rules

Conformance targets attribute used by:

- IEPD extension schema that follows the more relaxed NDR 4.0 rule set

```
ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-and
```

NDR 5.0 Reference Schema rules

Conformance targets attribute used by:

- any conformant schema from the 5.0-series of releases
- IEPD extension schema that follows the stricter NDR 5.0 rule set

```
ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-and
```

NDR 5.0 Extension Schema rules

Conformance targets attribute used by:

- IEPD extension schema that follows the more relaxed NDR 5.0 rule set

```
ct:conformanceTargets="http://reference.niem.gov/niem/specification/naming-and
```

NIEM Code Lists Specification

The **Code Lists Specification** adds support for new capabilities of NIEM code lists beyond the basic enumeration representations provided by XML and JSON schema (simple sets of fixed-schema code values and definitions).

- Specification Resources
- Overview
- Examples
  - Code CSV
  - Run-Time (Dynamic) Binding
  - Schema-Time (Static) Binding
- Conformance Targets
- FAQ
- Additional Resources

Specification Resources

Submit a new issue, view all issues, track issue status, or go to the GitHub repo for this specification.

Available versions

- **Version 4.0**, published 2017. Use with NIEM 4.0 - NIEM 5.2. **[Current version]**

Overview

The Code Lists Specification establishes methods for using code list artifacts with NIEM information exchange specifications.

New code formats

This specification adds support for the use of Genericode documents, as well as for CSV code lists.

Static and dynamic code list support

NIEM has traditionally supported static code lists, with code values and definitions defined in fixed schemas. Code updates require schema updates. This specification supports this traditional method of defining code lists during the development of the the schemas, via annotations for XML Schema documents that bind to code lists.

This specification also introduces a feature requested by the NIEM user community - support for specifying code lists at run time. This is done via binding code lists as additional data in the XML instance documents rather than as part of the fixed definitions in the schemas.

Multi-column support

Additionally, this specification introduces multi-column support for NIEM codes. This enables

additional information to be provided with codes beyond only the standard definition information supported by NIEM schemas. Examples include code sets that capture both literals and definitions; code sets that provide definitions in multiple languages; and linked code sets, like state and county codes, or vehicle make and model codes.

> The Code Lists Specification does not replace the existing XML and JSON schema enumerations, which still represent the majority of code lists in NIEM. This specification is additive, providing additional code list formats and support for new features, like dynamic code lists and linked columns.

Examples

Code CSV

An example of a code CSV is provided below. CSVs may be simple, providing basic information like code value and definition; or complex, providing not only code values and definitions, but also additional representations or information as well.

Make code	Make description	Model code	Model description	Class
FORD	Ford	FUS	Fusion	Auto
FORD	Ford	F15	F-150	Pickup
HOND	Honda	CIV	Civic	Auto
HOND	Honda	CRV	CRV	SUV
HOND	Honda	ACC	Accord	Auto
TOYT	Toyota	COA	Corolla	Auto
TOYT	Toyota	CAM	Camry	Auto

CSVs may be linked to a URI via an XML catalog file.

Run-Time (Dynamic) Binding

The example below shows an XML schema snippet that creates two code elements of type `nc:CodeType`. This type enables code list information to be provided in instance messages.

```
<xs:element name="VehicleMakeCode" type="nc:CodeType"
            substitutionGroup="nc:VehicleMakeAbstract">
  <xs:annotation>
    <xs:documentation>A code for a manufacturer of a vehicle.</xs:documentat
  </xs:annotation>
</xs:element>

<xs:element name="VehicleModelCode" type="nc:CodeType"
            substitutionGroup="nc:VehicleModelAbstract">
  <xs:annotation>
    <xs:documentation>A code for a model of a vehicle.</xs:documentation>
  </xs:annotation>
</xs:element>
```

The example below is an XML instance based on the schema snippet above:

> Because the schema does not identify a URI or column name of a code set, this information must be provided by the instance.

```
<?xml version="1.0" encoding="US-ASCII" standalone="yes"?>
<nc:Vehicle
    xmlns:cli="http://reference.niem.gov/niem/specification/code-lists/4.0/coc
    xmlns:ext="http://example.org/namespace/extension-run-time"
    xmlns:nc="http://release.niem.gov/niem/niem-core/4.0/">
  <ext:VehicleMakeCode
      cli:codeListURI="http://example.org/code-list/vehicle-make-model"
      cli:codeListColumnName="make"
    >DODG</ext:VehicleMakeCode>
  <ext:VehicleModelCode
      cli:codeListURI="http://example.org/code-list/vehicle-make-model"
      cli:codeListColumnName="model"
    >R15</ext:VehicleModelCode>
</nc:Vehicle>
```

Schema-Time (Static) Binding

The XML schema example below creates static, schema-time bindings between the elements and and CSVs. URIs and column names are used to indicate the particular set of valid code values.

```
  <xs:element name="Vehicle" type="nc:VehicleType"
              substitutionGroup="nc:Vehicle">
    <xs:annotation>
      <xs:appinfo>
        <clsa:ComplexCodeListBinding
          codeListURI="http://example.org/code-list/vehicle-make-model">
          <clsa:ElementCodeListBinding
              elementName="ext:VehicleMakeCode"
              columnName="make"/>
          <clsa:ElementCodeListBinding
              elementName="ext:VehicleModelCode"
              columnName="model"/>
        </clsa:ComplexCodeListBinding>
      </xs:appinfo>
    </xs:annotation>
  </xs:element>

  <xs:element name="VehicleMakeCode" type="niem-xs:token"
              substitutionGroup="nc:VehicleMakeAbstract">
    <xs:annotation>
      <xs:documentation>A code for a manufacturer of a vehicle.</xs:documentat
      <xs:appinfo>
        <clsa:SimpleCodeListBinding
            codeListURI="http://example.org/code-list/vehicle-make-model"
            columnName="make"/>
```

```
      </xs:appinfo>
    </xs:annotation>
  </xs:element>

  <xs:element name="VehicleModelCode" type="niem-xs:token"
              substitutionGroup="nc:VehicleModelAbstract">
    <xs:annotation>
      <xs:documentation>A code for a model of a vehicle.</xs:documentation>
      <xs:appinfo>
        <clsa:SimpleCodeListBinding
            codeListURI="http://example.org/code-list/vehicle-make-model"
            columnName="model"/>
      </xs:appinfo>
    </xs:annotation>
  </xs:element>
```

The example below is an XML instance based on the schema snippet above:

> Because bindings to CSVs have been made in the schema (URI and column name), they do not appear in the instance.

```
<?xml version="1.0" encoding="US-ASCII"?>
<ext:Vehicle xmlns:ext="http://example.org/namespace/extension-schema-time">
  <ext:VehicleMakeCode>DODG</ext:VehicleMakeCode>
  <ext:VehicleModelCode>R15</ext:VehicleModelCode>
</ext:Vehicle>
```

Conformance Targets

The Code Lists Specification includes the following conformance targets:

Code	Conformance Target	Description	Spec
CLD	code list document	A file or resource that contains one or more code lists.	Link
GC-CLD	Genericode code list document	A code list document with codes represented as Genericode.	Link
CSV-CLD	CSV code list document	A code list document with codes represented as Comma-separated values (CSV).	Link
XSD	code list-enabled schema document	An XML Schema document that supports the use of code list documents for validation and meaning.	Link
INS	code list-enabled instance document	An XML document that leverages the Code Lists Specification to connect data values with code lists.	Link
VSET	code list validation set	An abstract concept that brings together the necessary components to define validity of XML documents with respect to code lists, and to identify correspondences between XML data and code list distinct entries.	Link

FAQ

These are some questions that have been frequently asked about code lists, Genericode, and the NIEM Code Lists Specification.

1. What if simple type enumerations don't match code list values?

Any code lists constraint (e.g., an element value must match a code list) is layered on top of any other constraints, from XML Schema, Schematron, or even other code lists. So, if a simple type enumeration allowed codes "red", "green", and "blue", while the Genericode code list allowed only "blue" and "purple", then the only valid value for the element would be "blue".

2. Do you have to create a new code list type for each different code list you use?

No. You have lots of options. If you're binding the code lists in schema (rather than run-time binding), you can encode each different code list in a separate element.

If you're doing run-time binding, you can use `cli:CodeType`, or you can hang the `cli:codeListURI` attribute on each type for which you want *run-time code list binding*.

3. Can an authoritative source update a code list without updating NIEM?

Absolutely. That's one of the main driving factors in the way the code lists specification is built and used. We expect authoritative sources to update code lists as they need to. It will be up to the code list publishers and users to make sure they are using the same versions of the code lists.

4. Can an authoritative source update a code list without updating an IEPD?

Yes, but you need to make sure that all the partner implementations of the IEPD are using the same strategy for updating their code lists. You probably don't want some systems using v1 of a code list and others using v13 of the code list.

5. What if my Genericode file has no unique key column?

It is a requirement of Genericode that every Genericode file have at least one key constraint. The NIEM Code Lists Specification respects that, and it means that people who create Genericode files need to make sure that their code lists files are valid and maintain key constraints.

6. Does a referenced column have to be a key column?

No. A reference to non-key columns might mean that you get multiple distinct entries that match your values. That's OK; that's how it is designed, but use caution.

7. Can I only use Genericode code lists with NIEM 3.2 and later releases?

No. You can use it with any NIEM 3-based IEPD. This includes NIEM 3.0, 3.1, and 3.2. We're not even telling you that you **can't** use it with other XML Schemas; you're just likely going to

run into difficulties.

8. Does a Genericode file have multiple code lists in it?

No. The NIEM Code Lists Specification uses Genericode documents with document element `gc:CodeList`, which allow only one code list per document. If you need multiple code lists, use multiple code list documents.

9. Can you convert Excel files to Genericode?

Maybe. There are existing tools for working with Genericode files, as well as tools under development within the NIEM community. See our Genericode page for more details.

10. When will I get a Genericode file from the NIEM Schema Subset Generation Tool (SSGT)?

The SSGT will yield a Genericode file when one is referenced by a component that is selected for a wantlist. If an XML Schema type references a code list, that code list will be included in your subset.

11. Where should I publish my code list documents?

You should consider your code list documents to be part of a schema set, like XML Schema documents. If you are building an IEPD, then code list documents should be provided alongside the schemas in your IEPD. If you are publishing a domain update, code list documents should be part of your domain update.

The NTAC will be providing guidelines on how to provide updates to code lists.

12. How do I validate a Genericode file?

See our Genericode page for code list tools that can validate Genericode documents.

13. How do I validate a message against a Genericode file?

Software to validate messages against code list documents is under development.

14. How do I describe use of code lists in an IEPD?

The IEPD specification uses XML Catalogs to define XML Schemas for validation. The NIEM Code Lists Specification describes how to use XML Catalogs to resolve code lists. Add your code lists to your IEPD's XML catalog. Additional guidance will be provided in the future.

Additional Resources

See more information about the NIEM Code Lists Specification:

- Country Codes Example

This example shows different ways to define and represent country codes in NIEM -

via free text, XML schema enumerations, and CSV codes via the Code Lists Specification.

- Genericode

  A list of known tools and resources available for Genericode.

Country Codes Example

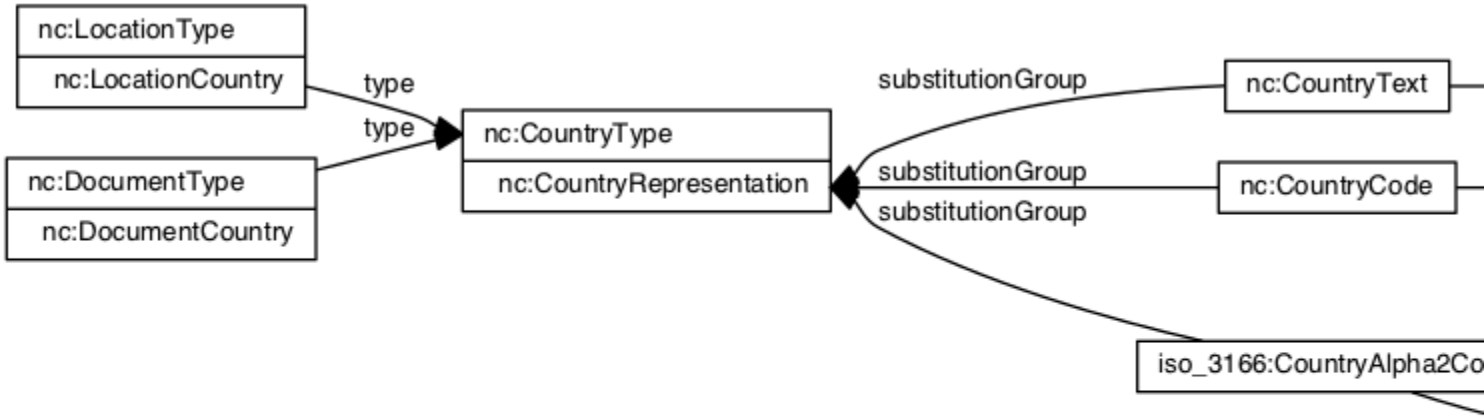
- Representations diagram
- Free-text
- Enumerations
- CSV codes

Representations diagram

Country codes in NIEM 4.0 provide a good example of different representations of code lists in NIEM. ISO 3166 codes are represented as schema enumerations, and GENC codes are represented by CSV files as described by the Code Lists Specification.

> In addition to using a CSV representation, country codes in NIEM 4.0 also follow the representation pattern to enable better reuse.

Country representations in NIEM:



Free-text

Representing country by a text value is the simplest method, but provides minimal interoperability, as sender and receiver must somehow agree on what any given value means.

An element `nc:CountryText` carries a text string that identifies a country. It is one of the representations for a country.

`nc:CountryText` is defined in XML Schema as an element of type `nc:TextType`, which is substitutable for `nc:CountryRepresentation` using `substitutionGroup`.

```
<xs:element name="CountryText" type="nc:TextType"  substitutionGroup="nc:Count
```

A text representation of the United States of America may appear in an XML message:

```
<nc:Document>
  <nc:DocumentOriginCountry>
    <nc:CountryText>United States of America</nc:CountryText>
  </nc:DocumentOriginCountry>
</nc:Document>
```

Enumerations

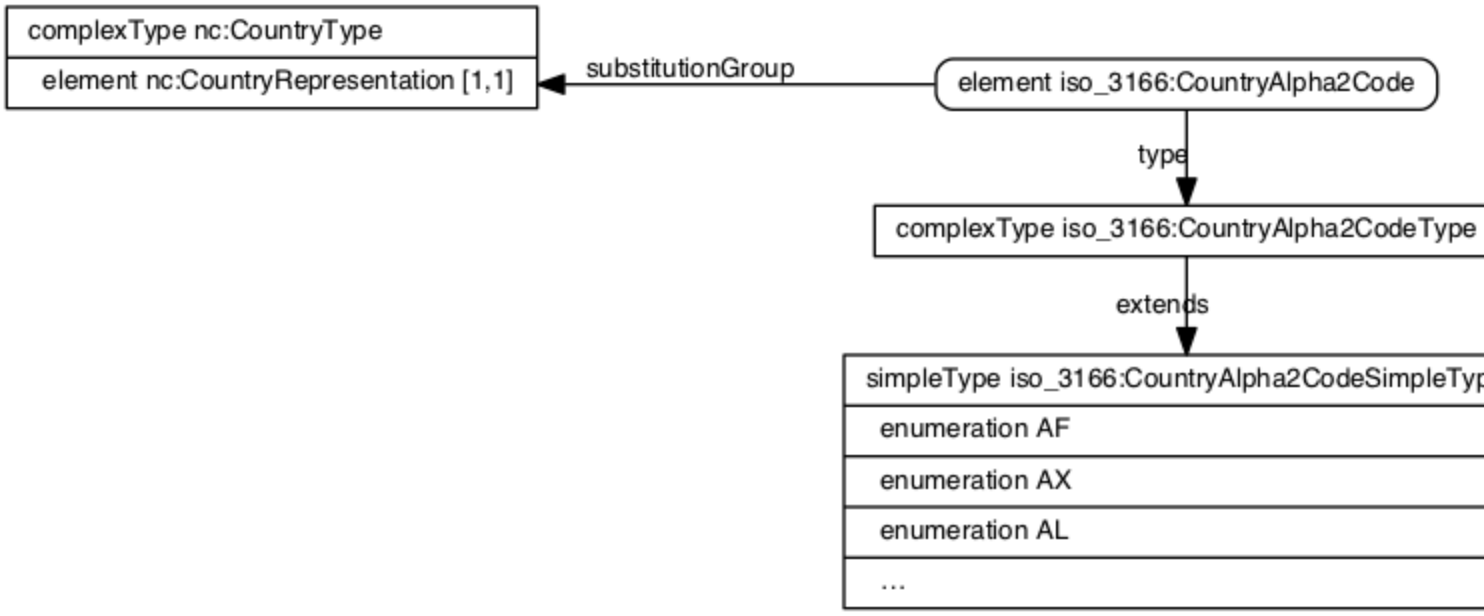
ISO represents country codes via the ISO 3166-1 standard, which incorporates three different representations for each country:

- 2 character (“Alpha-2”)
- 3 character (“Alpha-3”)
- Numeric

Each of these is represented in NIEM 4 via an element, which is of a type that encodes the available code values using enumerations, as NIEM has done traditionally.

Although this example shows only the Alpha-2 encoding, this pattern will be followed for ISO-3166-1’s Alpha-2 encoding, its Alpha-3 encoding, and its Numeric encoding.

First, the diagram below shows the alpha2 codes, represented by element `iso_3166:CountryAlpha2Code`, which is of a type based on a simple type with enumerations.



An XML message instance using these components appears as follows:

```
<nc:Document>
  <nc:DocumentOriginCountry>
    <iso_3166:CountryAlpha2Code>US</iso_3166:CountryAlpha2Code>
  </nc:DocumentOriginCountry>
</nc:Document>
```

An encoding similar to the way we represent GENC, below, was considered for representing ISO 3166-1 in NIEM 4, but was rejected because of issues specific to ISO 3166-1, which include:

1. ISO does not publish URIs identifying the ISO 3166 code lists. NIEM would have to invent NIEM-specific URIs for ISO's code lists, and NIEM prefers not to do so.
2. ISO does not publicly publish identifiers for specific versions of the ISO 3166 code lists. It is unclear how exchange developers and implementers would distinguish between versions of ISO 3166.
3. NIEM 3 used simple type enumerations for ISO 3166-1 code lists, and no issues have been submitted that indicate a problem with that representation.

CSV codes

The following pattern is used to represent the GENC codes in NIEM. This includes the representations of country codes shown below, but is also extended for Geopolitical subdivisions (e.g., states) described by GENC. All versions of GENC use code lists as described by The NIEM Code Lists Specification.

This is made more convenient by:

1. GENC provides an identifying URI for every version of every code list
2. GENC provides machine-readable forms of their code lists
3. GENC documents describe messages that use the code list URI along with code values.

GENC encodings of country codes use the generic code type `nc:CodeType` described by The NIEM Code Lists Specification. This allows for run-time-specified code list URIs. A GENC website identifies the following URIs that identify encodings of the current GENC Geopolitical entities code list:

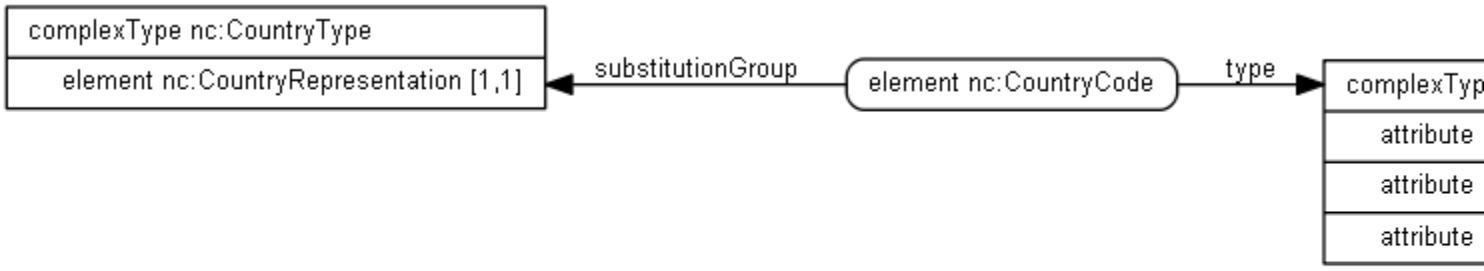
Version	Encoding	URI
3-3	2-character encoding	http://api.nsgreg.nga.mil/geo-political/GENC/2/3-3
3-3	3-character encoding	http://api.nsgreg.nga.mil/geo-political/GENC/3/3-3
3-3	numeric encoding	http://api.nsgreg.nga.mil/geo-political/GENC/n/3-3

These encodings incorporate not only the code list (the list of entities) but particular encodings of codes for the code list (2-character, 3-character, numeric). As a result, for binding to the code list, a different code list URI is used for each encoding, and the column name is not used.

A sample instance, using a 3-character encoding for the United States of America, appears as follows:

```
<nc:Document>
  <nc:DocumentOriginCountry>
    <nc:CountryCode cli:codeListURI="http://api.nsgreg.nga.mil/geo-political/G
        >USA</nc:CountryCode>
  </nc:DocumentOriginCountry>
</nc:Document>
```

This uses the generic code type `nc:CodeType`, provided as part of the NIEM Code Lists Specification, described by the following diagram:



Here we see that `nc:CountryCode` uses the generic `nc:CodeType`, requiring it to have a code list URI and optional column name and constraining indicator.

The code list for GENC Geopolitical entities, 3-character encoding, is provided as a CSV file as follows:

char3 code	name	short name	full name
AFG	AFGHANISTAN	Afghanistan	Islamic Republic of Afghanistan
XQZ	AKROTIRI	Akrotiri	Akrotiri
ALB	ALBANIA	Albania	Republic of Albania
…	…	…	…

This CSV code list file is provided in a sub-directory of the NIEM release, along with CSVs for the other encodings of the Geopolitical Entities code list, as well as the Geopolitical Subdivisions code list (`niem/codes/genc`). These CSV code list files are identified by XML catalog documents provided as part of the NIEM release (`niem/xml-catalog.xml`).

Users and exchange developers who must (1) update to newer versions of GENC code lists, or (2) incorporate exchanges using older versions of GENC code lists, or (3) use code lists other than GENC, in alignment with local exchange requirements, may do so by using other code list URIs, and may optionally incorporate new code list files as CSV files or Genericode files in an XML catalog used with their exchange, as described by the NIEM Code Lists Specification.

Genericode

- Tools
  - Crane Softwrights Code List Resources
  - Webb Roberts's Code List Tools

Genericode is a XML representation for code lists defined by OASIS in 2007. The NIEM Code Lists Specification leverages Genericode as a method of representing code lists in NIEM information exchange specifications.

The OASIS Genericode specification defines the Genericode XML format.

Tools

Since Genericode is an XML syntax, a person may use any standard XML tools to work with Genericode. In addition, some tools provide specific support for Genericode code lists.

Crane Softwrights Code List Resources

Crane Softwrights' page on OASIS UBL and Code List Resources features a package containing XSLT files that can convert a specially-structured CSV file to Genericode.

Webb Roberts's Code List Tools

Webb Roberts provides a set of code list tools on Github that include Unix/Linux/Mac command-line tools that:

1. Convert Excel/spreadsheet CSV files to Genericode files.
2. Validate Genericode files against the Genericode schema and additional rules.

You can download a complete release of Webb's code list tools.

NIEM High-Level Version Architecture (HLVA) Specification

The **NIEM High-Level Version Architecture (HLVA) Specification** identifies the processes, artifacts, and responsibilities required to produce new releases of the NIEM model. It also establishes a regular release cycle for predictable and manageable NIEM updates.

- Specification Resources
- Objectives
- Status

Specification Resources

Available versions

- **Version 3.0**, published 2015. Use with NIEM 3.0 and later releases. **[Current version]**

Objectives

The HLVA was designed to meet the following objectives:

- A domain may issue a domain update as needed to publish content changes outside of the regular release cycle.
- IEPDs may reuse content from NIEM releases and domain updates.
- Domain updates are incorporated into the next NIEM release.
- A clear process is available for domains to participate in Core-related harmonization and issue resolution.
- A reliable schedule exists for releases and release activities.
- All NIEM content changes are visible, via uniquely identified schemas and from change logs.

Status

This specification is being stood down by the NIEM Technical Architecture Committee (NTAC). The content in the specification is being migrated directly into this site, primarily in the Reference / Release section.

Tools

NIEM tools provide support for searching and exploring NIEM content and developing NIEM domains and exchanges.

- NIEM program tools
- NIEM resources for other tools

NIEM program tools

The following is information about the tools that are developed and supported by the NIEM program. Information about community and commercially-available tools can be found in the **NIEM Tools Catalog**.

- Schema Subset Generation Tool (SSGT)

  The Schema Subset Generation Tool (SSGT) enables users to search and view the content of the NIEM model, and build XML Schema subsets for use in exchanges.

- Conformance Testing Assistant (ConTesA)

  The Conformance Testing Assistant (ConTesA) enables users to test NIEM XML schemas against the automated rules from the NIEM Naming and Design Rules (NDR).

- Movement

  Movement provides a user-friendly interface and smarter search results for the latest NIEM release. Movement is also open source so the community who inspired its creation can contribute to it.

- Migration Tool

  The Migration Tool enables you to upgrade a NIEM XML Schema subset generated by the SSGT to the next release.

NIEM resources for other tools

- NIEM resources for the Oxygen XML Editor

  The Oxygen XML Editor is a commercial multi-platform XML editor. NIEM provides some additional information and resources for this tool that may assist with domain and IEPD schema development and conformance testing.

- NIEM resources for XMLSpy

XMLSpy is a proprietary XML editor and integrated development environment developed by Altova. NIEM provides some additional information for this tool that may assist with domain and IEPD schema development and testing.

Schema Subset Generation Tool (SSGT)

The **Schema Subset Generation Tool (SSGT)** enables you to search and explore the content of the NIEM model. Additionally, you have the option of building XML Schema subsets of a NIEM release for use in NIEM XML exchanges. Based on the list of components selected by you for the subset, the tool will calculate dependencies and generate a valid set of schemas that are a subset of a release for download as a zip file.

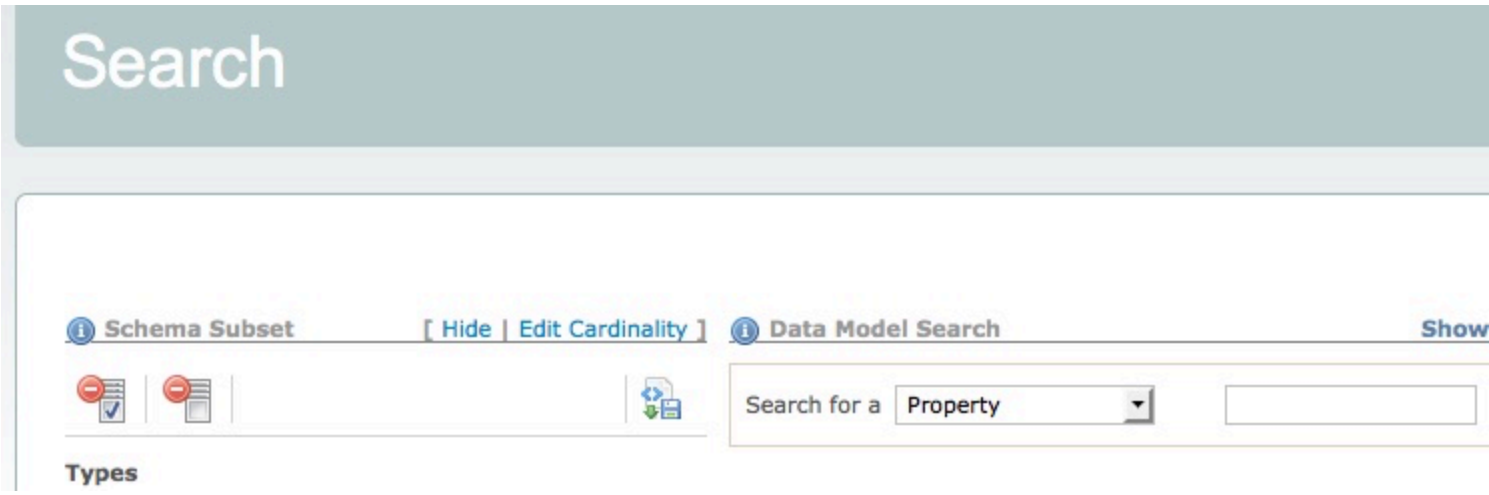
A NIEM subset will contain only the user-selected elements and types from a release, plus any required dependencies. The subset will likely be much smaller than the corresponding full NIEM release (the 5.0 release has 12,000+ elements) and easier to deal with in an IEPD.

- Search and explore the content of NIEM
- Build XML Schema subsets of a release
    - Automatic management of component dependencies
    - Customize cardinality and nillable values
- Multi-release support
- API for selecting NIEM subset components (wantlist)

- Search
    - Basic Search
    - Advanced Search
- Search Results
    - Search Results List
    - Property Info Pages
    - Type Info Pages
- Build a Subset
    - Add Properties
    - Add Types
    - Remove Components
    - Generate Subset
- Options
    - Change Release
    - Resume work later
    - Additional options

Search

The Schema Subset Generation Tool (SSGT) search function allows you search and navigate through the various NIEM properties, data types, namespaces, codes / facets, and their relationships to each other.



Tips for better searches

- Uses spaces between terms.
- Do not include namespace prefixes or information (e.g., `nc:`) in the search.
- Narrow down search results by using the advanced search if too many results are returned.
- Look through the components of any types that match your requirements - results may sometime appear as inherited elements (e.g., `nc:ArrestType` inherits element `nc:ActivityDate` instead of creating a custom arrest date) or augmentations (additional related properties defined in other namespaces).

Basic Search

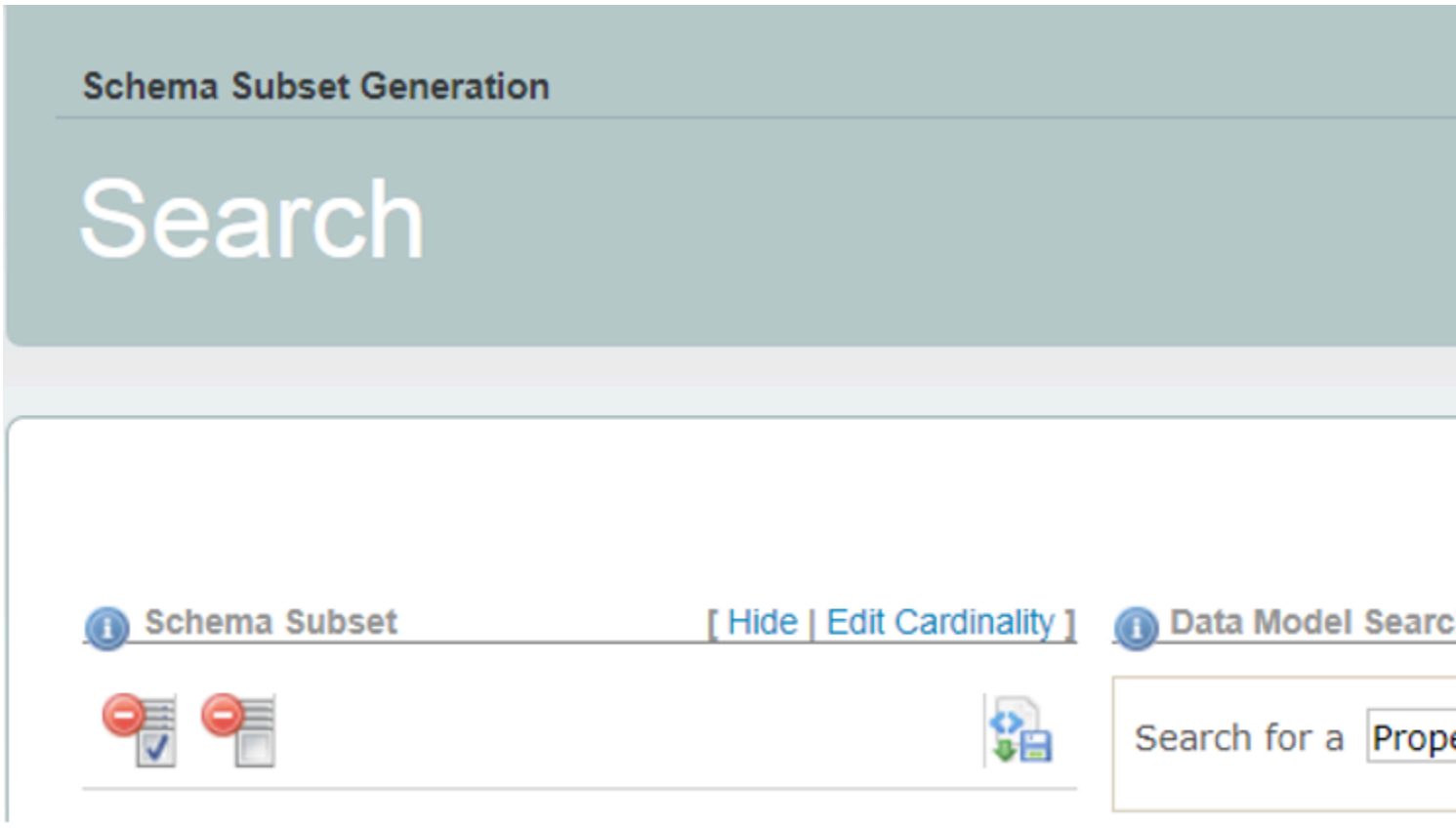
Enter one or more terms into the search box for a default search of NIEM properties.

Change the selection in the drop-down box to change the search to one of the following:

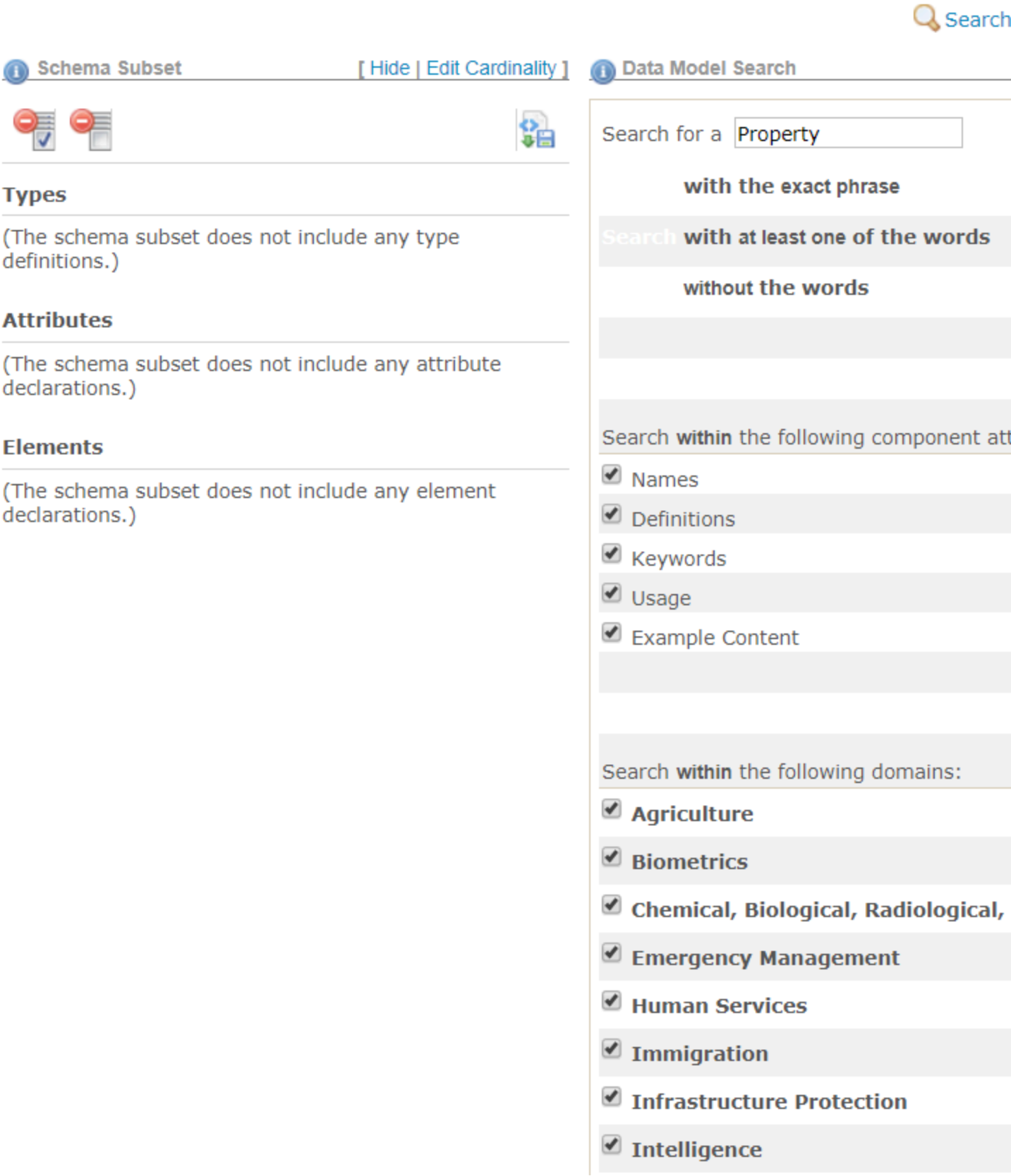
Category	Comments
Property	Search NIEM elements and attributes (default).
Type	Search NIEM types.
Namespace	Search namespace prefixes, target namespace URIs, and definitions.
Facet	Search code values and definitions.
External	Search external element names and definitions.
Association	Search association element names and definitions.

Advanced Search

Click the toggle link **`Show Advanced`** / **`Hide Advanced`** to show or hide the advanced search options.



When you click the **`Show Advanced`** link, the following options will appear:



- Enter search terms in the text-box labeled **`with the exact phrase`** to search multiple

terms.
- Enter search terms in the text-box labeled **`with at least one of the words`** to conduct a logical OR search.
- Enter search terms in the text-box labeled **`without the words`** to exclude terms from the search results.
- Uncheck the check-boxes Names, Definition, Keywords, Usage, and Example Content to exclude the unchecked item from being searched for the search terms entered above.
- Filter the list of namespaces to be searched.
    - The checkbox for **`Other`** should likely remain checked - it includes Core.
    - If no checkboxes are checked, no results will be returned.

Search Results

The properties and types displayed in the search results are hyperlinked to display more detailed information about each component. This helps users navigate through the data model and explore the model hierarchy.

Search

Schema Subset [Hide | Edit Cardinality]

Types

(The schema subset does not include any type definitions.)

Attributes

(The schema subset does not include any attribute declarations.)

Elements

(The schema subset does not include any element declarations.)

Data Model Search Show

Search for a Property unit

Add mo:UnitTypeCodeText (mo : UnitTypeCodeTextType)
Add mo:speedUnitCode (unece : VelocityCodeSimpleType
Add nc:AddressSecondaryUnitText (nc : TextType) details
Add nc:DisciplinaryActionCommunityService (nc : ActivityT
Add nc:LengthUnitCode (unece : LengthCodeType) details
Add nc:LocaleCommunityName (nc : TextType) details
Add nc:MeasureUnit abstract details
Add nc:MeasureUnitText (nc : TextType) details
Add nc:OrganizationSubUnit (nc : OrganizationType) deta
Add nc:OrganizationSubUnitName (nc : TextType) details
Add nc:OrganizationUnit (nc : OrganizationType) details
Add nc:OrganizationUnitAssociationAugmentationPoint abst
Add nc:OrganizationUnitName (nc : TextType) details
Add nc:PersonAssignedUnitAssociation (nc : PersonOrganiz
Add nc:PersonTemporaryAssignedUnitAssociation (nc : Per
Add nc:SpeedUnitCode (unece : VelocityCodeType) details
Add nc:TemperatureUnitCode (unece : TemperatureUnitCo
Add nc:TimeUnitCode (unece : TimeCodeType) details
Add nc:VehicleGrossLadenUnitWeightMeasure (nc : Measu
Add nc:VolumeUnitCode (unece : VolumeUnitCodeType) d
Add nc:WeightUnitCode (unece : MassCodeType) details
Add nc:quantityUnitText (xs : string) details
Add scr:LocationBuildingUnitName (nc : TextType) details
Add scr:LocationBuildingUnitNumberText (nc : TextType)
Add biom:NominalScanningResolutionValue (niem-xs : dec

First Previous Next Last

Displaying 76-100 of 251 results

Page 4 of 11

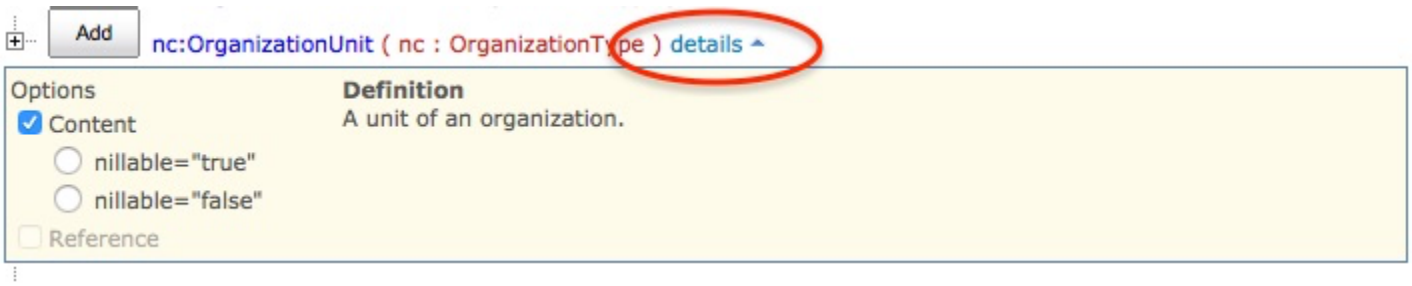
Search Results List

Some additional information about properties and types may be viewed directly from the search results list. Detailed information will be available by clicking the links to open the individual property info pages and type info pages.

Definitions

There are two ways to view the definition of a property or type from the search results page:

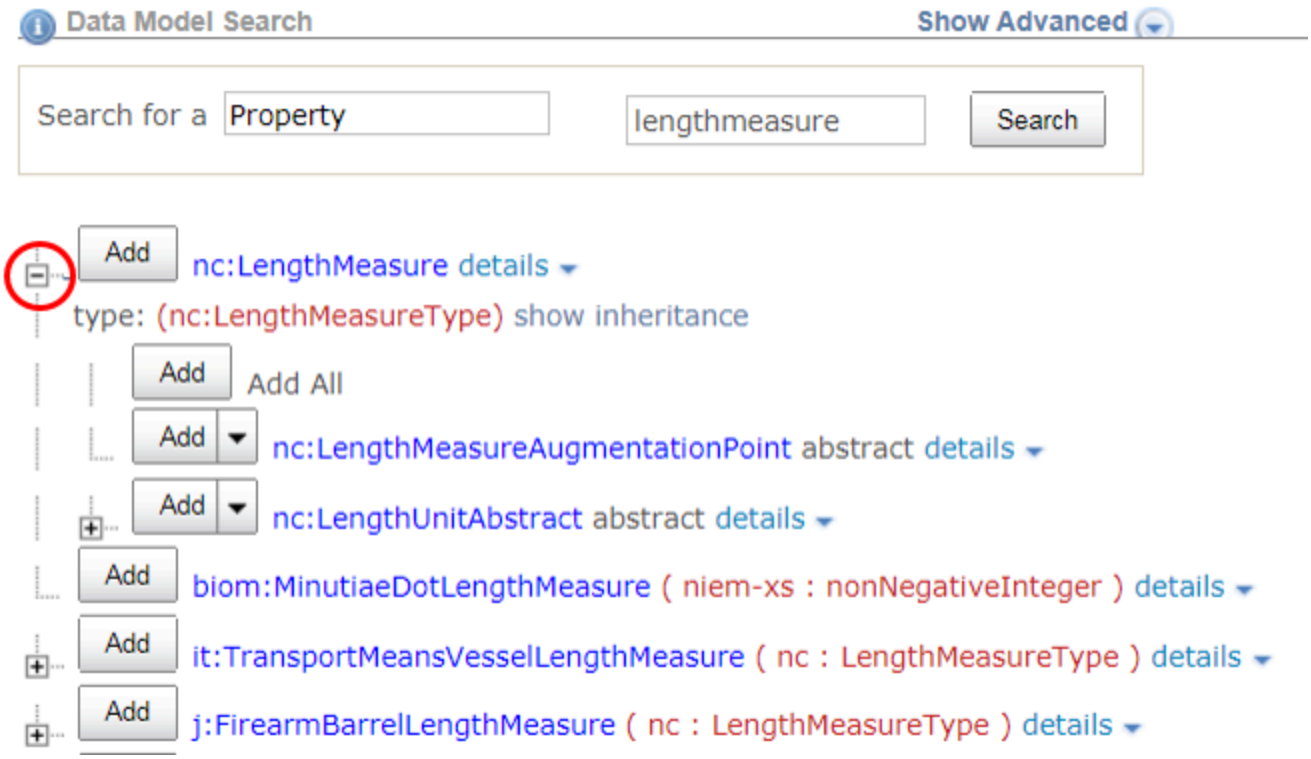
- Hover over a property or type link to see the definition for that component.
- Toggle the **`details`** link to show/hide definition. Multiple details panels may be open at the same time.



Sub-properties

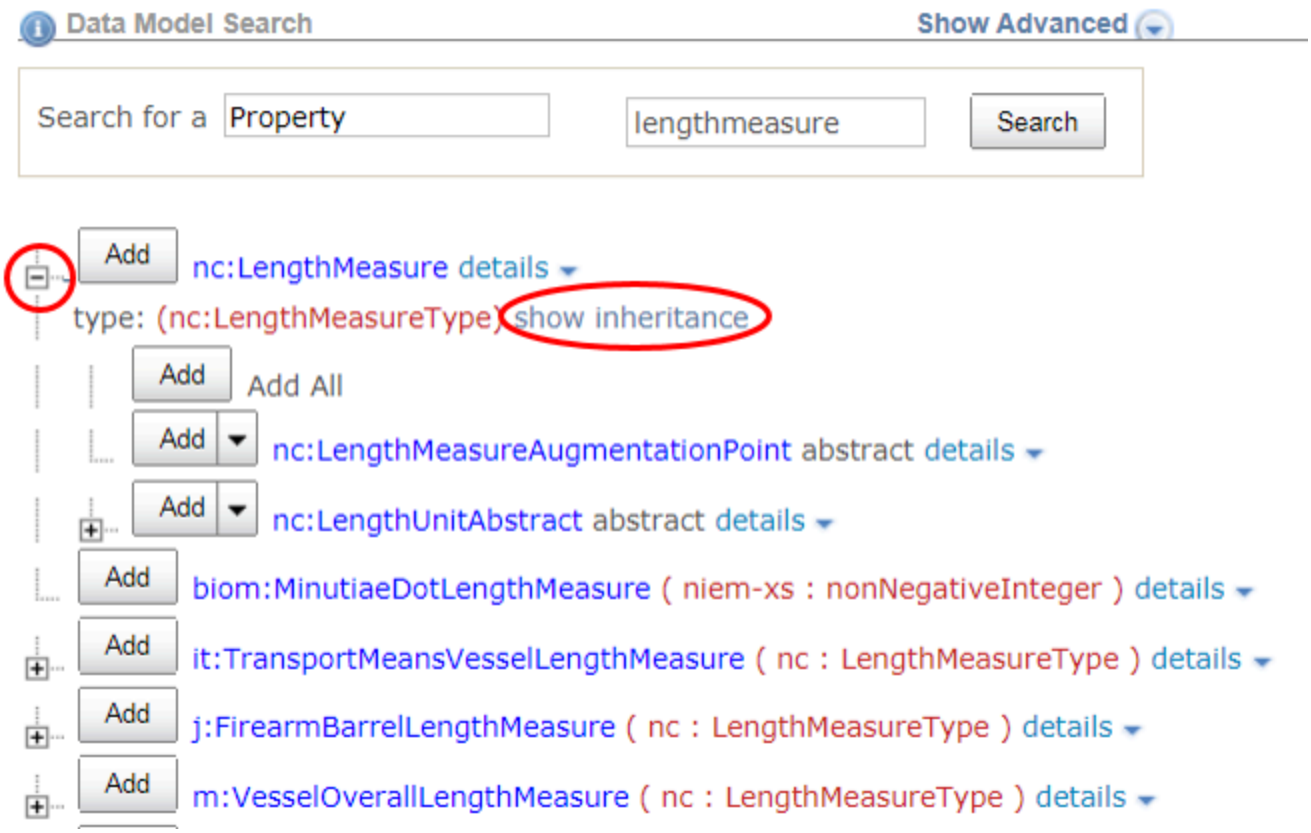
In the list of search results, click the **+** button to the left of the component to see the sub-properties that the type contains.

View the list of sub-properties for **`nc:LengthMeasure`**:



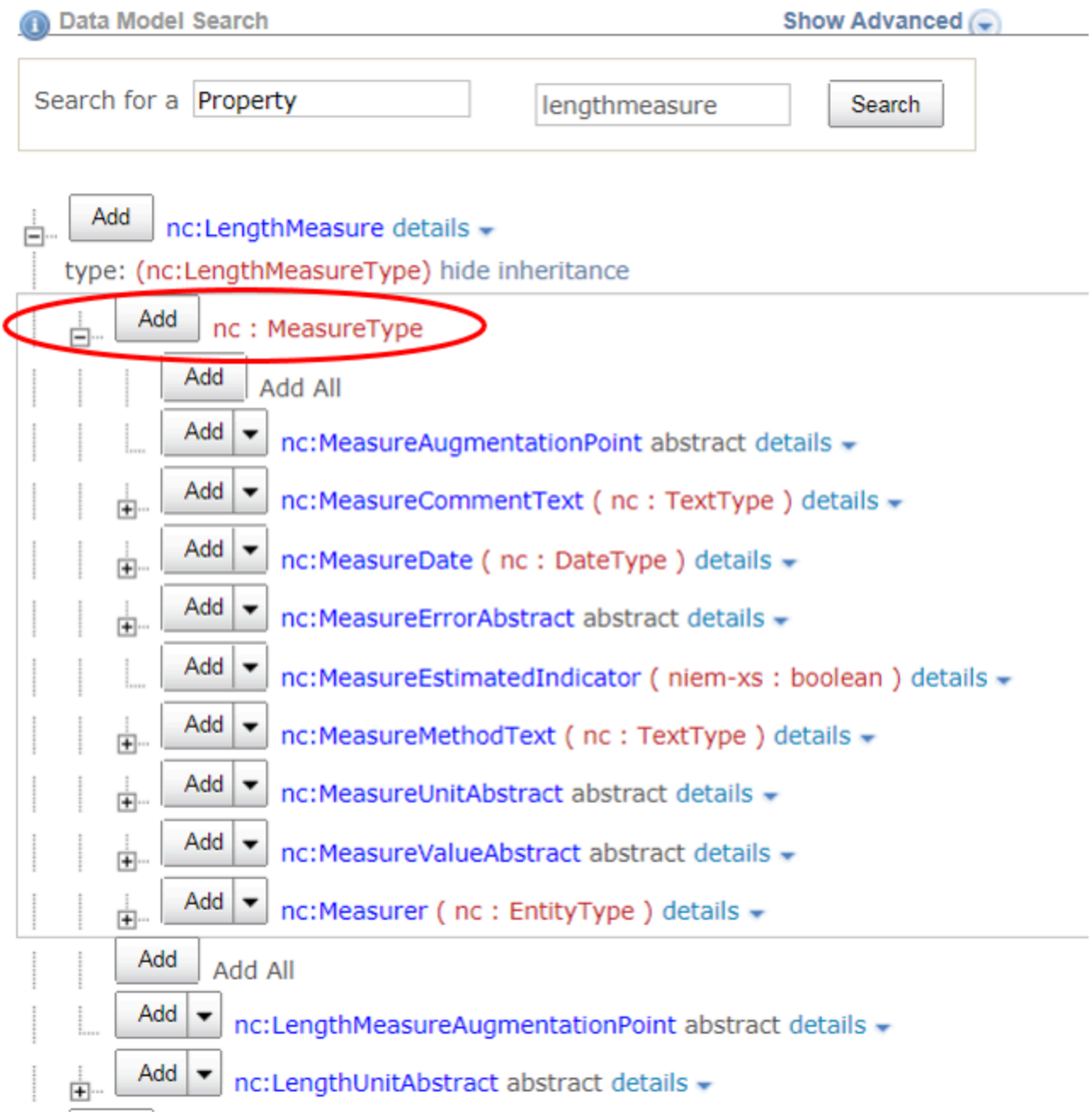
Inheritance

Once the sub-property list is expanded, a **`show inheritance`** link will appear if the type extends another type with sub-properties.



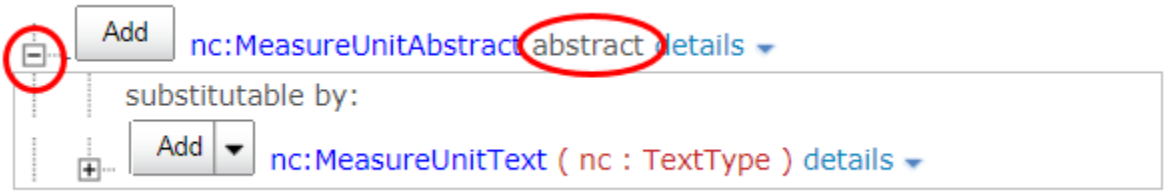
Each parent type may also be expanded to display their sub-properties.

After clicking **`show inheritance`** and the expand button, the list of sub-properties for **`nc:MeasureType`** (the parent type for **`nc:LengthMeasureType`**) is displayed. These properties will be inherited:



Substitutions

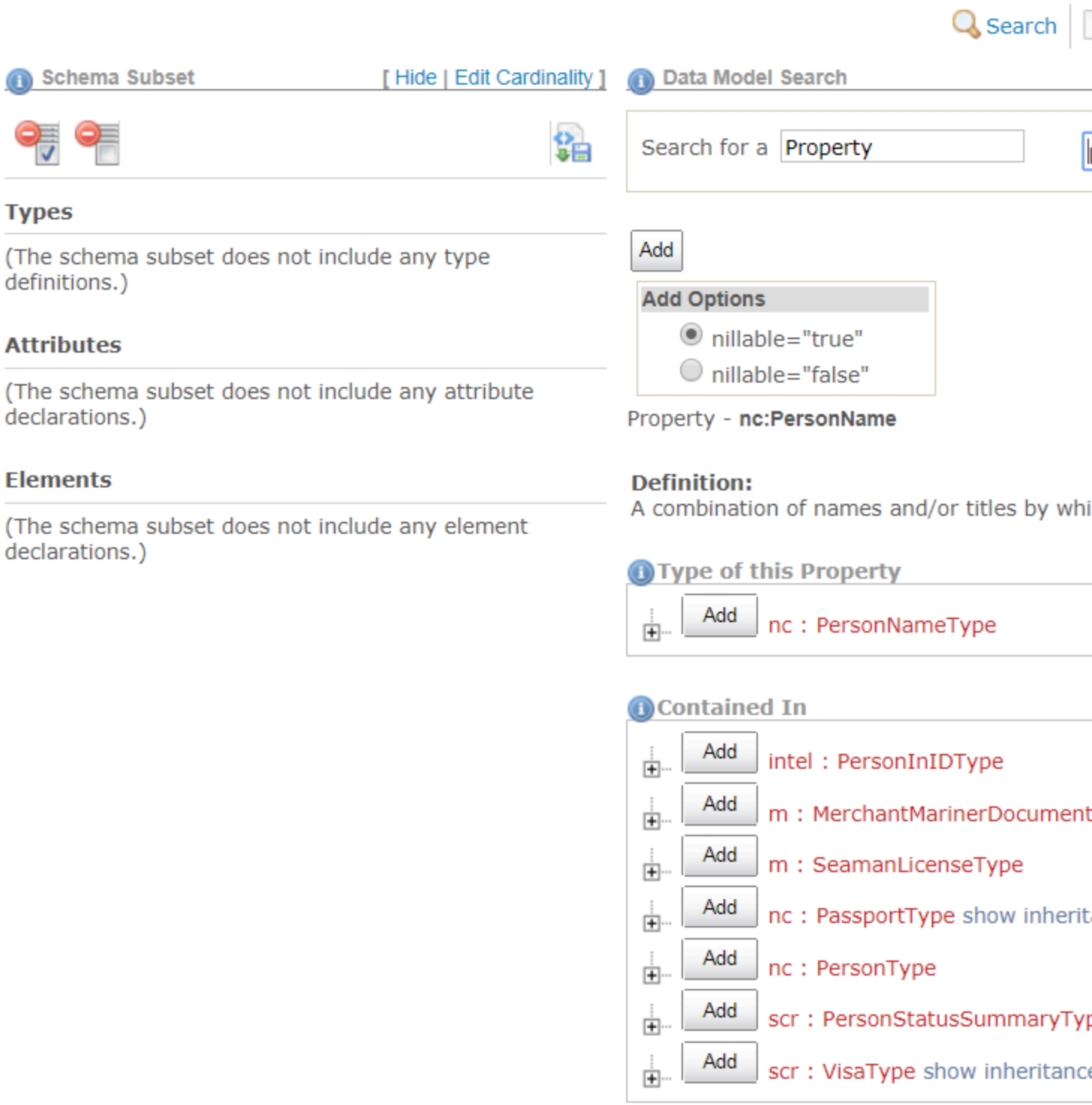
For abstract elements, click the **+** button to view a list of available substitutions.



Property Info Pages

Clicking on a property link will take you to a page with more information about that property. Commonly available information includes the definition, the type of the property, and any types that the property may be contained in.

Page for property **`nc:PersonName`**



Type Info Pages

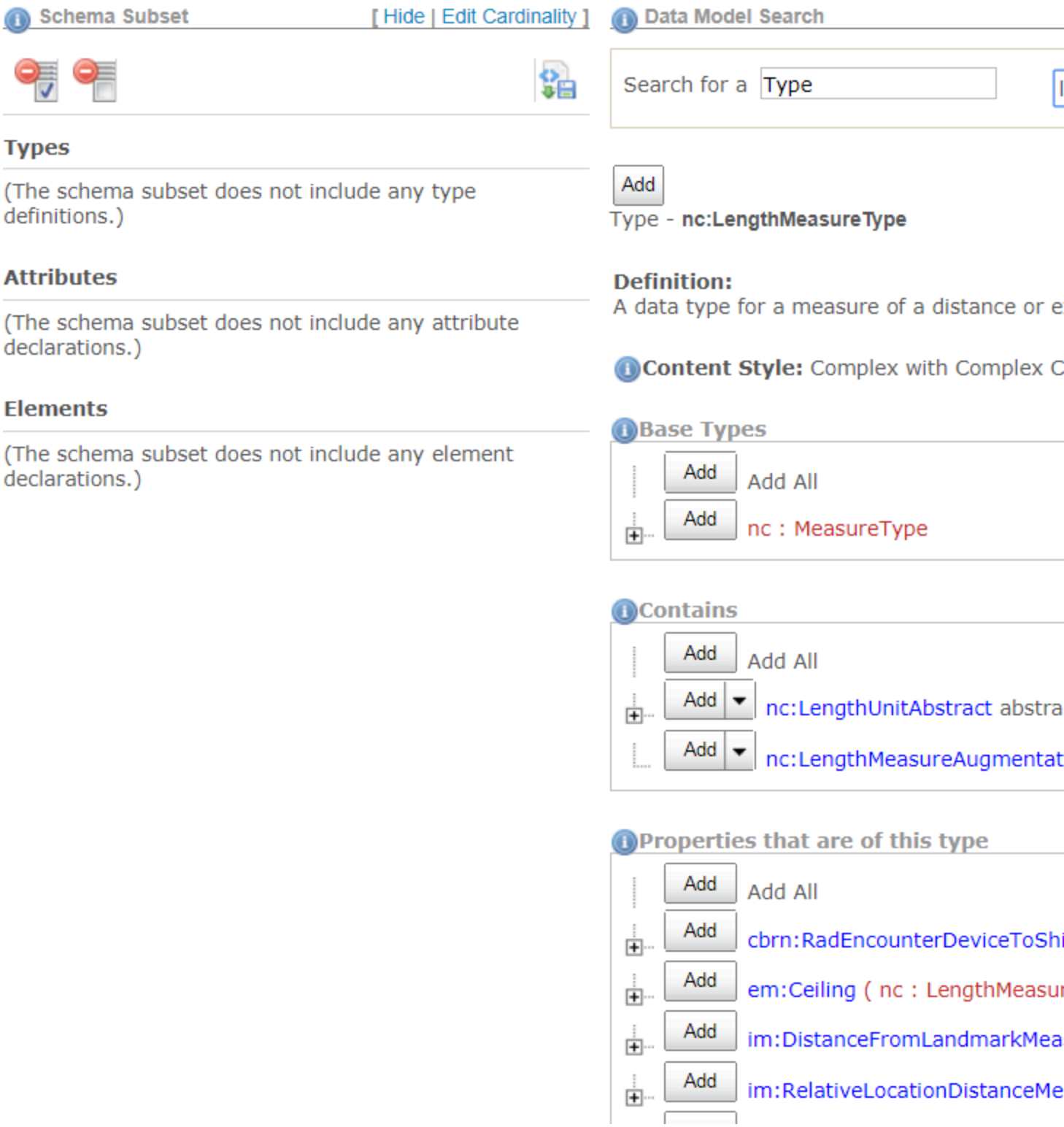
Clicking on a type link will take you to a page with more information about that type, including the definition, the kind of type it is, the parent or base type, and any elements of this type.

Further information on the page will vary based on what kind of type it is. The page may display a list of

sub-properties, codes, or other facets, as applicable.

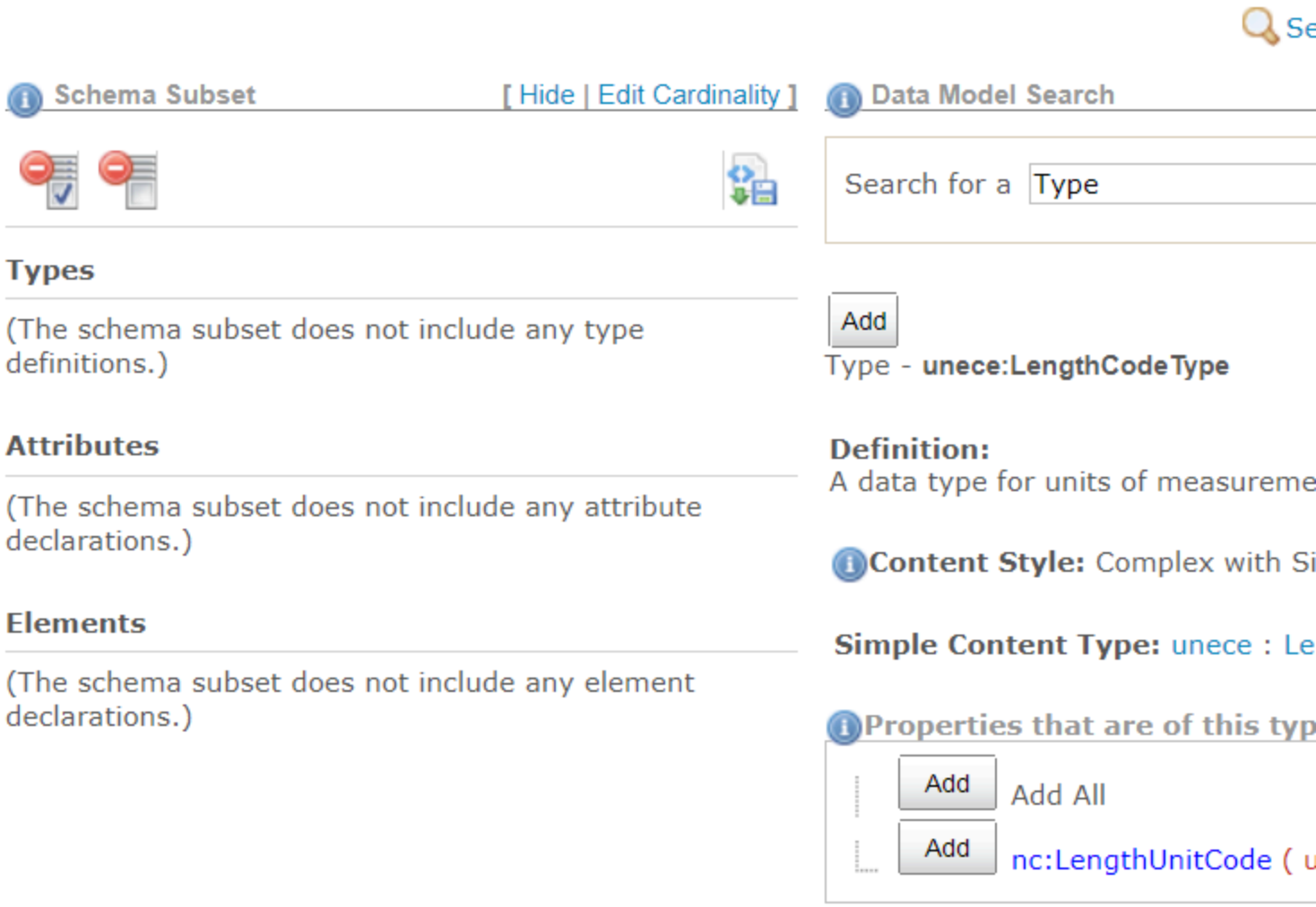
Complex type with sub-properties

Page for type **`nc:LengthMeasureType`**



Complex type with a value

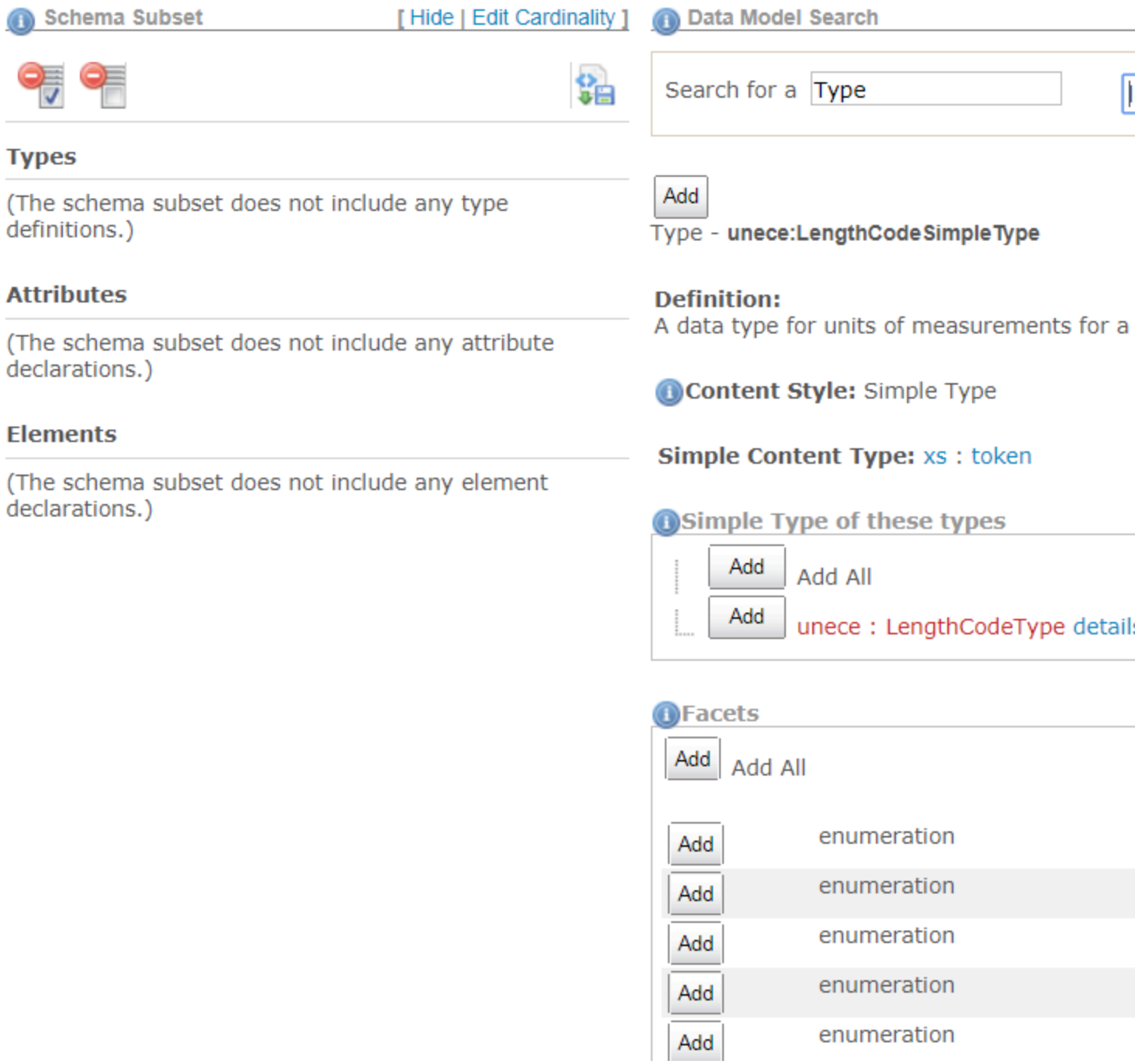
Page for complex type **`unece:LengthCodeType`**



Where are the codes? Click on the link to its simple content type, `unece:LengthCodeSimpleType` (identified in the image with the gold arrow).

Simple type

Page for type **`unece:LengthCodeSimpleType`**



Build a Subset

The SSGT allows you to pick and choose specific parts of a release to include in a subset. This subset, which will likely have many fewer files and components than the full release, can be used in your IEPD. While you can reuse a full release package, it is easier to understand and work with only the schemas and the components that you need.

If you are developing multiple IEPDs, you can build a custom subset for each one or one larger subset that supports the requirements from each IEPD.

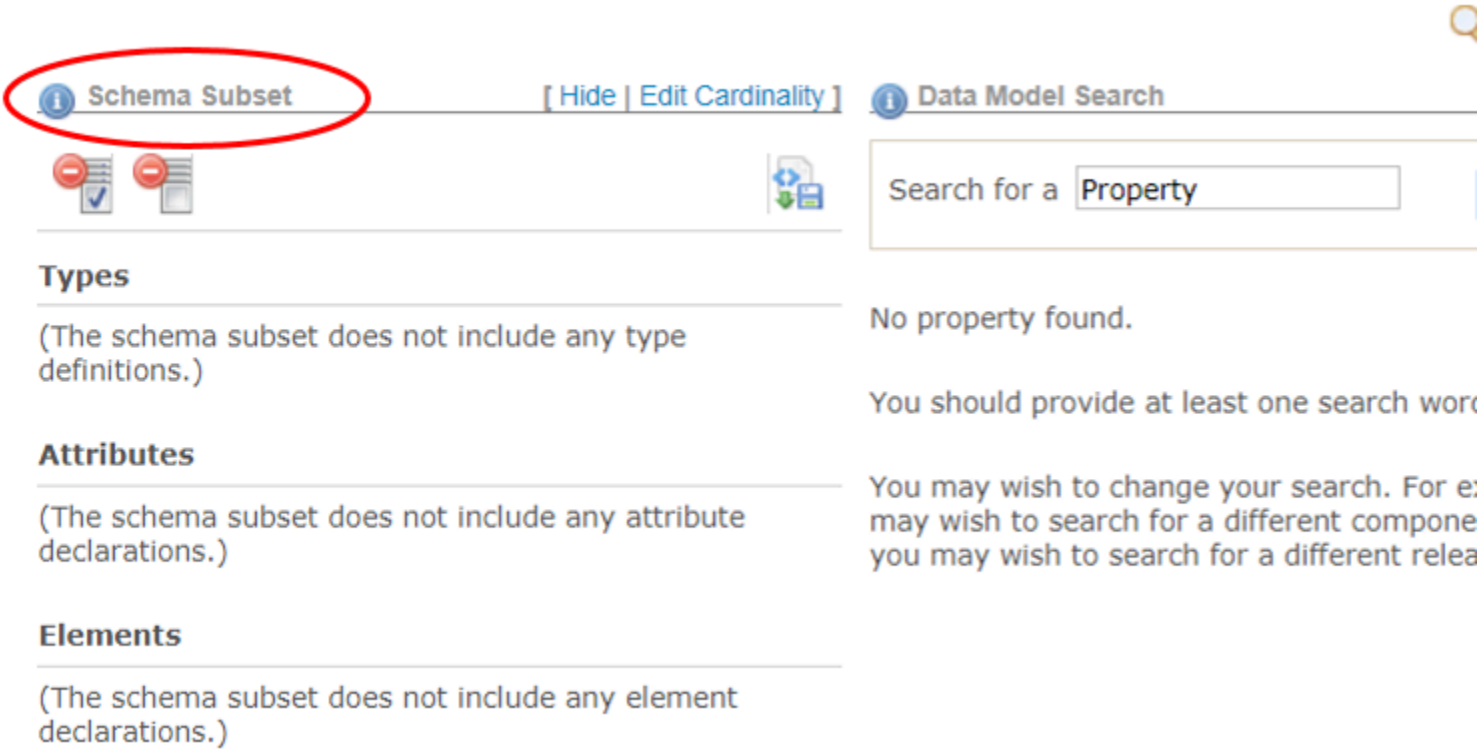
Please see Subset Schemas for more information about what a subset is.

- This tool does not have user accounts. To **save your work**, you must download either the individual **wantlist** file (a SSGT-generated XML file that stores your selections) or the **full subset zip file**, which will contain the latest wantlist.
- The **only way to resume your work** later on without starting over is to keep your latest wantlist file so you can upload it in the SSGT after your session has expired.

Subset Panel

When building a subset, you may add elements, attributes, and types directly from the search results page, or from the property and type pages. All components in your subset will appear in the subset list on the left side of the page.

> The example below shows an empty subset. No types, attributes, or elements have been added yet.



Dependencies

The SSGT will automatically calculate all required dependencies whenever a component is added to the subset list.

- User-selected components will appear in bold font in the subset panel.

- Dependencies will appear in regular font.

If you later remove a selected component, dependencies that are no longer necessary will also be removed.

> If you add property `nc:Person` to your subset, the SSGT will also add type `nc:PersonType` to your subset as well since it is a required dependency of the property. If you later remove `nc:Person` from your subset and no other component needs it, `nc:PersonType` will be removed as well.

Add Properties

There are three ways of adding properties to a subset from the search results page. Choices can be made on a case-by-case basis:

1. Add selected properties independently

Adding properties independently results in a subset that might look like:

- nc:PersonGivenName
- nc:PersonSurName
- nc:PersonBirthDate
- nc:ItemMakeName
- nc:ItemModelName
- nc:IdentificationID

These properties have been added as a flat list.

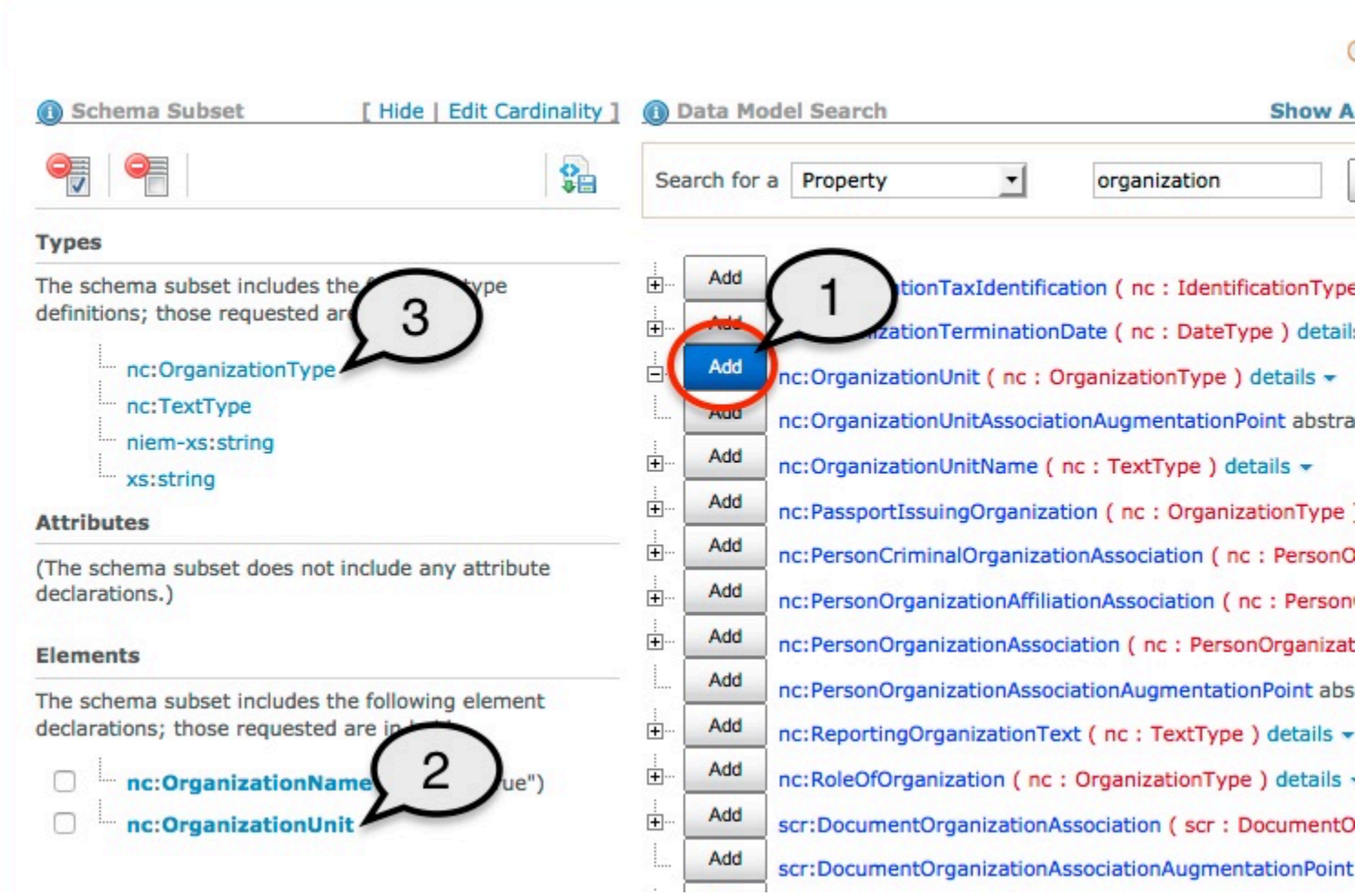
> Without providing a nested structure for these properties via the subset (e.g., NIEM's nc:PersonType, nc:ItemType, etc.) or a custom structure in an IEPD schema, then there will be no way to group related data. If a message with this subset has two given names, two last names, and two birth dates, two separate persons cannot be extracted correctly with any certainty:
>
> - Are values provided alphabetically?
> - Is the first occurrence of a field for person #1 and the second occurrence of a field for person #2?
> - Do multiple values represent a single person with alias information?

Steps

- Click the **`Add`** button next to the element or type name in the search results list.
- The component will be added to the appropriate section in the subset list on the left, in bold font to indicate a user-selection.
- Any dependencies the component may have will also be added to the subset list, in regular font.

> In the example below, element `nc:OrganizationUnit` is added to an existing subset that also contains element `nc:OrganizationName`.

- (1) Click the add button to add the element to the subset.
- (2) The SSGT will add the selected component to the subset list on the left.
- (3) The SSGT will also add dependencies to the subset list (the type of the element: `nc:OrganizationType`).



2. Add selected properties nested under a type

Adding properties nested under a type preserves the relationships defined in the NIEM model. This might look like:

- nc:CrashDriver
  - nc:PersonName
    - nc:PersonGivenName
    - nc:PersonSurName
  - nc:PersonBirthDate
- nc:Vehicle
  - nc:ItemMakeName
  - nc:ItemModelName
  - nc:VehicleIdentification
    - nc:IdentificationID

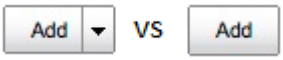
Leveraging the structure that NIEM provides can add a lot more context to your exchanges. Both examples from #1 and #2 include fields to capture first and last name, birth date, and vehicle make, model, and VIN, but #2 is much more informative.

Option #1 is useful when you want to add NIEM properties to your subset so you can build out a custom structure in your IEPD.

In order to add an element or attribute to a subset as a sub-property of a type, this relationship must be captured explicitly. The SSGT will not do this automatically. Many properties belong to more than one type; other times a property may be added to a new type in an exchange.

Any easy way to tell if a property is going to be added as a sub-property is to look at the **`Add`** button.

- For sub-properties, the button also includes a drop-down icon on the right side to allow users to set sub-property options.
- Other **`Add`** buttons do not have these options so they do not have this drop-drown:

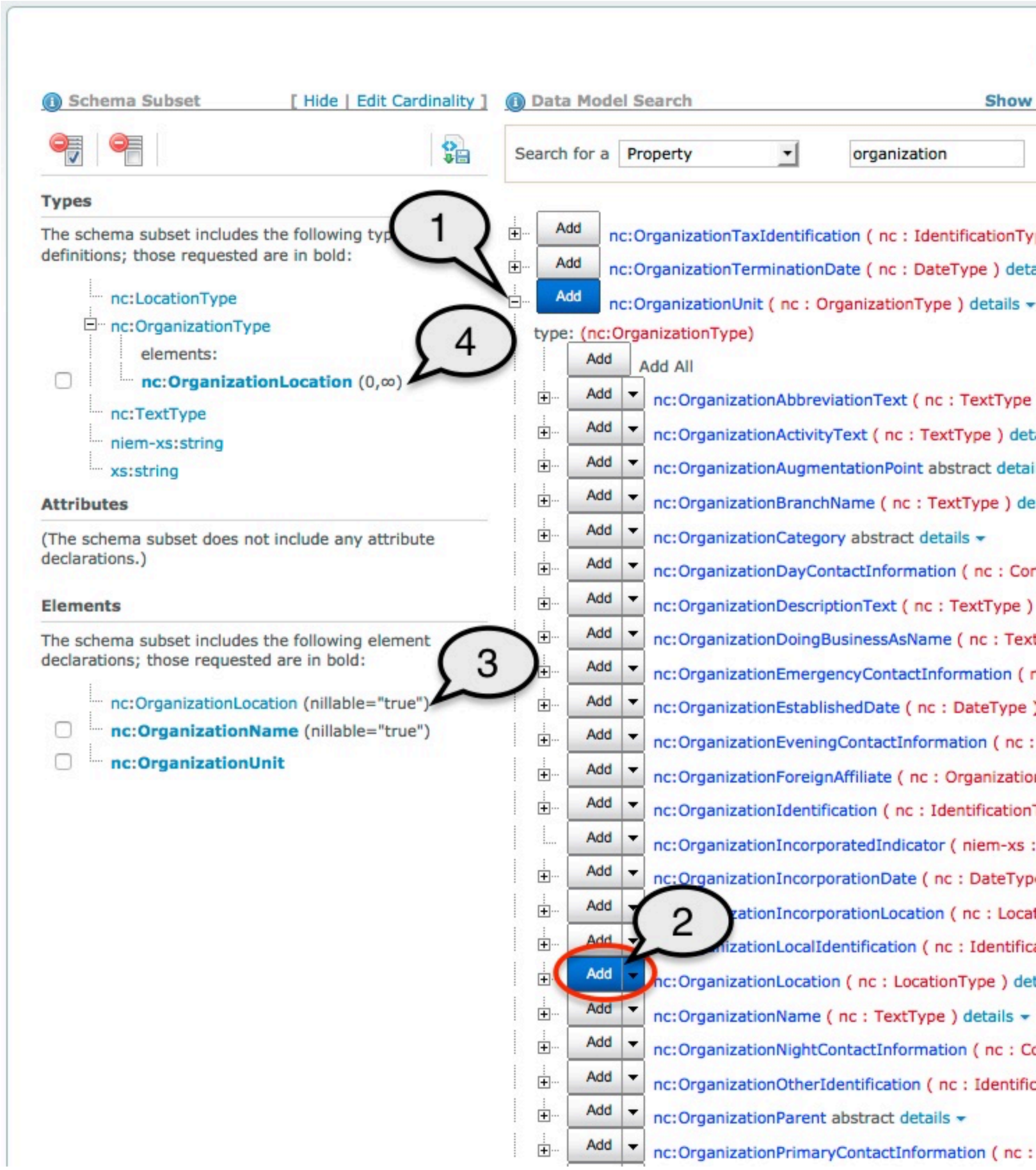


Steps to add nested property from search results

- Click the **+** icon next to a component to expand the list of sub-properties.
- From the sub-property list, click the **`Add`** button next to the property name.
- The property will be added to the appropriate **`Elements`** or **`Attributes`** section of the subset list on the left.
- The type, in not already in the subset, will be added to the **`Types`** section of the subset list on the left.
- Click the **+** icon next to the type in the subset list to view all of its sub-properties in the subset.

In the example below, element `nc:OrganizationLocation` is added to the subset as a sub-property of type `nc:OrganizationType`.

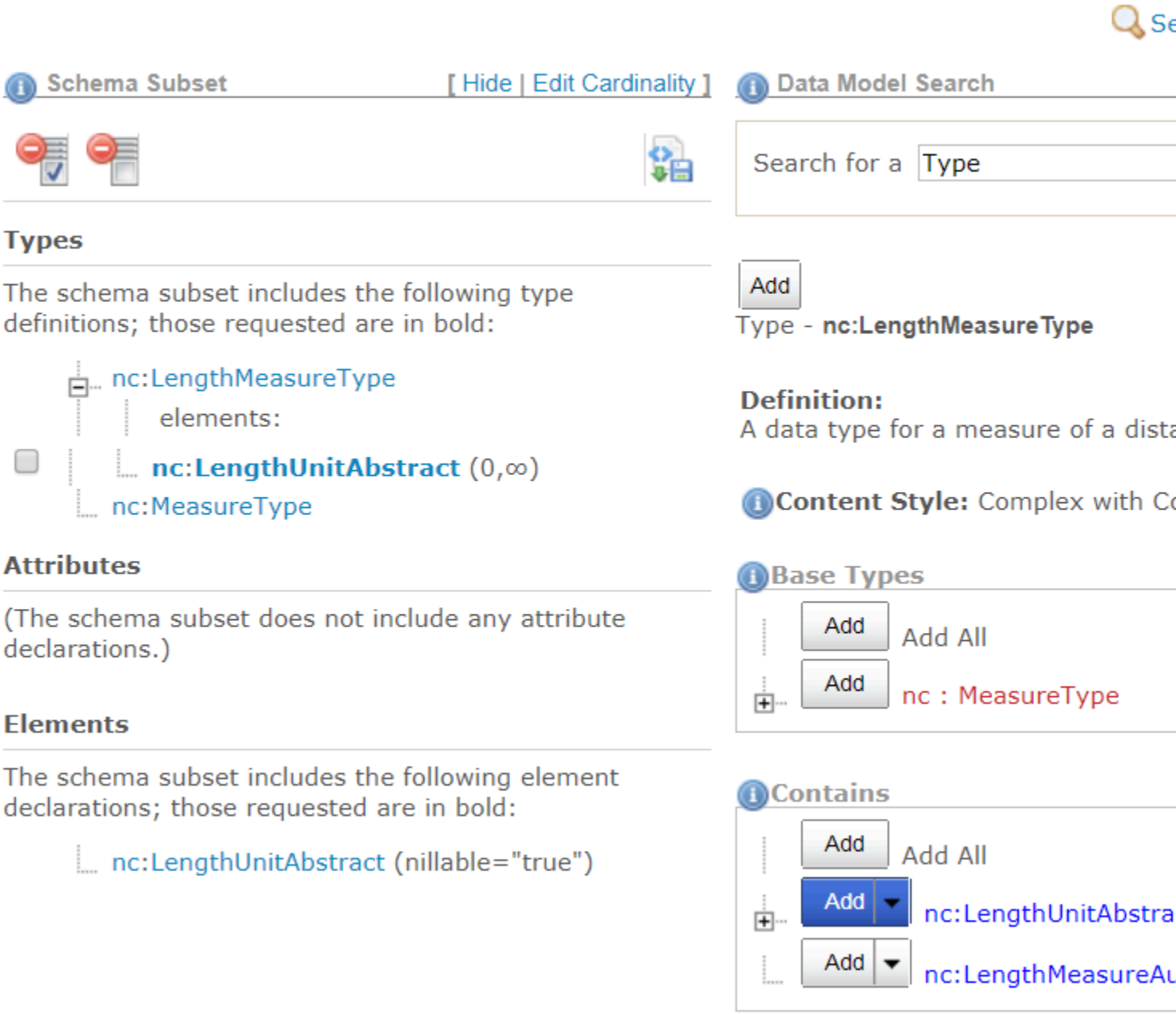
- (1) The user expands the list of sub-properties for type `nc:OrganizationType`.
- (2) The user adds element `nc:OrganizationLocation` to the subset as a sub-property of `nc:OrganizationType`.
- (3) The SSGT adds the element to the subset list.
- (4) The SSGT adds the type to the subset list along with its sub-property.



Steps to add nested property from type pages

Adding sub-properties to a type will happen automatically for any property added to a subset from a type page, in the **`Contains`** section.

> Adding nc:LengthUnitAbstract to the subset from type type page for type nc:LengthMeasureType will automatically add it as nested under the type.

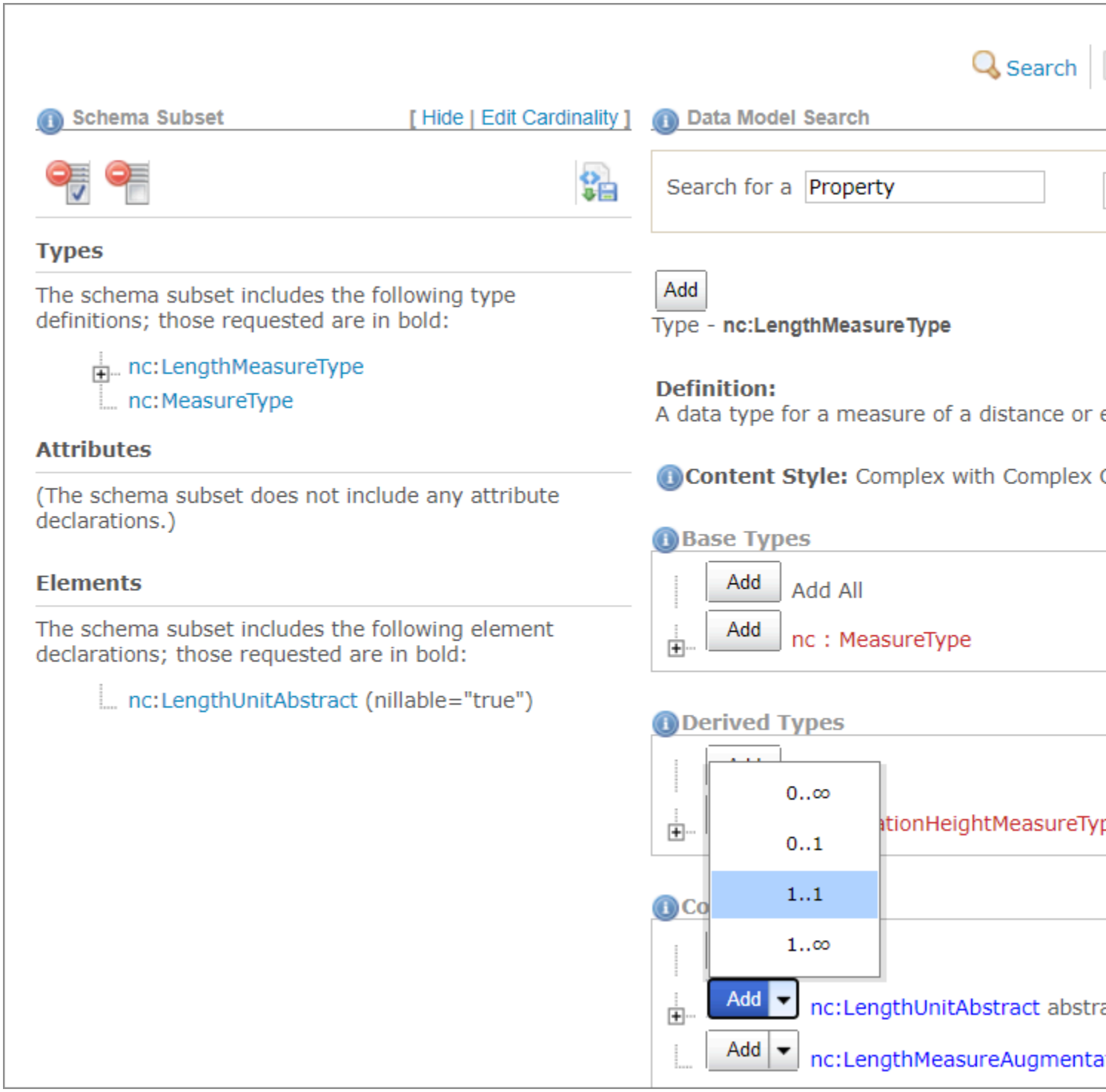


Set cardinality while adding a nested property to a type

Subsets also let you constrain how many times an element may appear nested within a type. As a reference model, NIEM usually sets cardinality to optional and over-inclusive (`minOccurs="0" maxOccurs="unbounded"`) to support a wide variety of requirements.

In an exchange, you can customize the cardinality to your exact requirements. Make elements required, limit occurrences to only once, or set any other cardinality as long as it does not conflict with the original cardinality from NIEM (an element that is required in NIEM cannot be made optional in a subset).

Click the drop-down on the **Add** button to set a custom cardinality for the sub-property:



Cardinality can also be set after properties have been added to the subset by clicking the “Edit

Cardinality" link at the top of the Subset panel.

3. Add all nested properties to their type

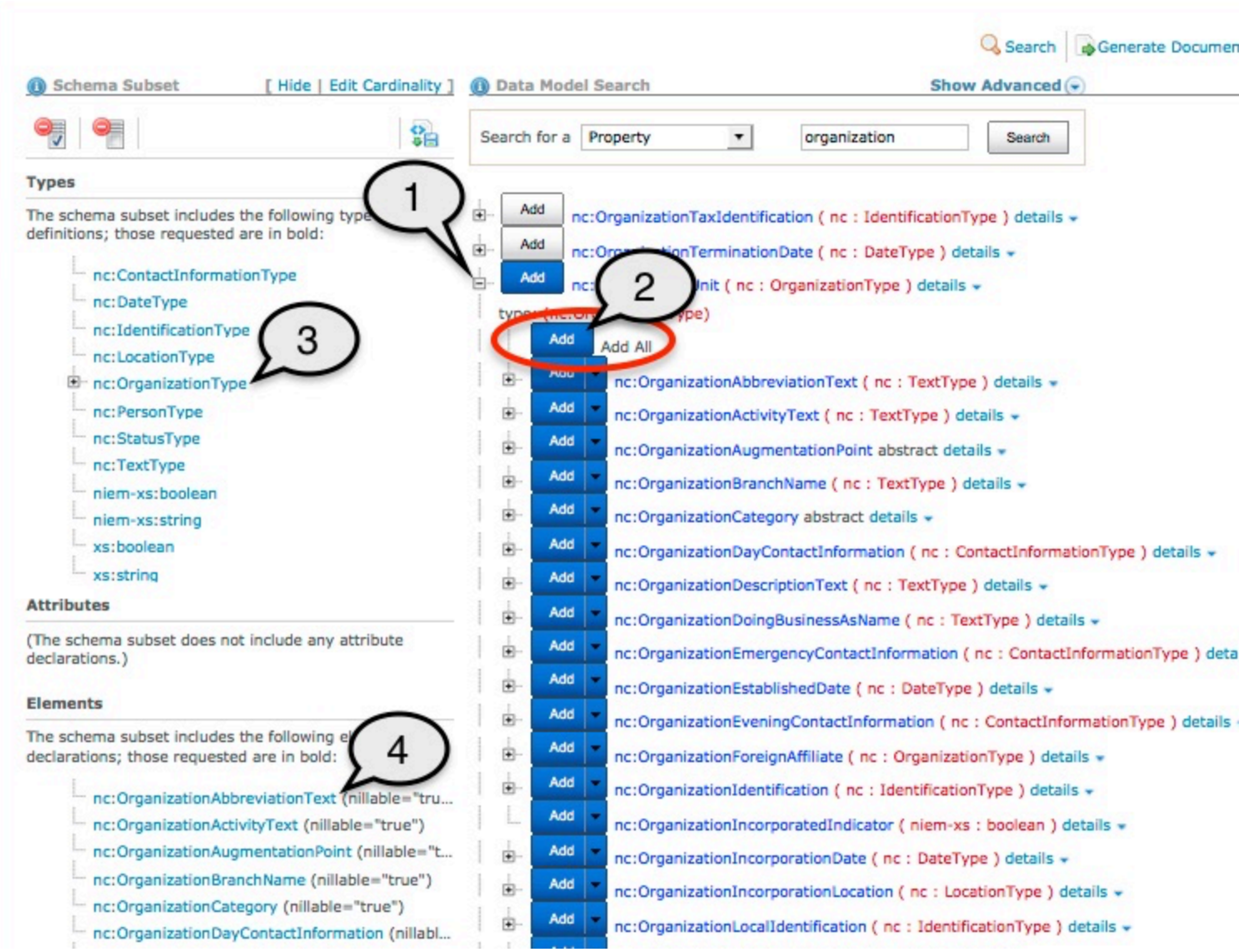
This option is similar to #2, but instead of selecting certain properties to include nested under a type, all available ones can be added in a single step.

- Click the **+** icon next to a component to expand the list of sub-properties.
- From the sub-property list, click the **`Add`** button next to **`Add All`** label. This will appear above the list of sub-properties.
- The property will be added to the appropriate **`Elements`** or **`Attributes`** section of the subset list on the left.
- The type, if not already in the subset, will be added to the **`Types`** section of the subset list on the left.

This action affects only the designated type. It is not recursive. Any new type added to the subset through this action will be added empty. Users will need to add additional sub-properties to as needed.

In the example below, all sub-properties of type `nc:OrganizationType` are added to the subset.

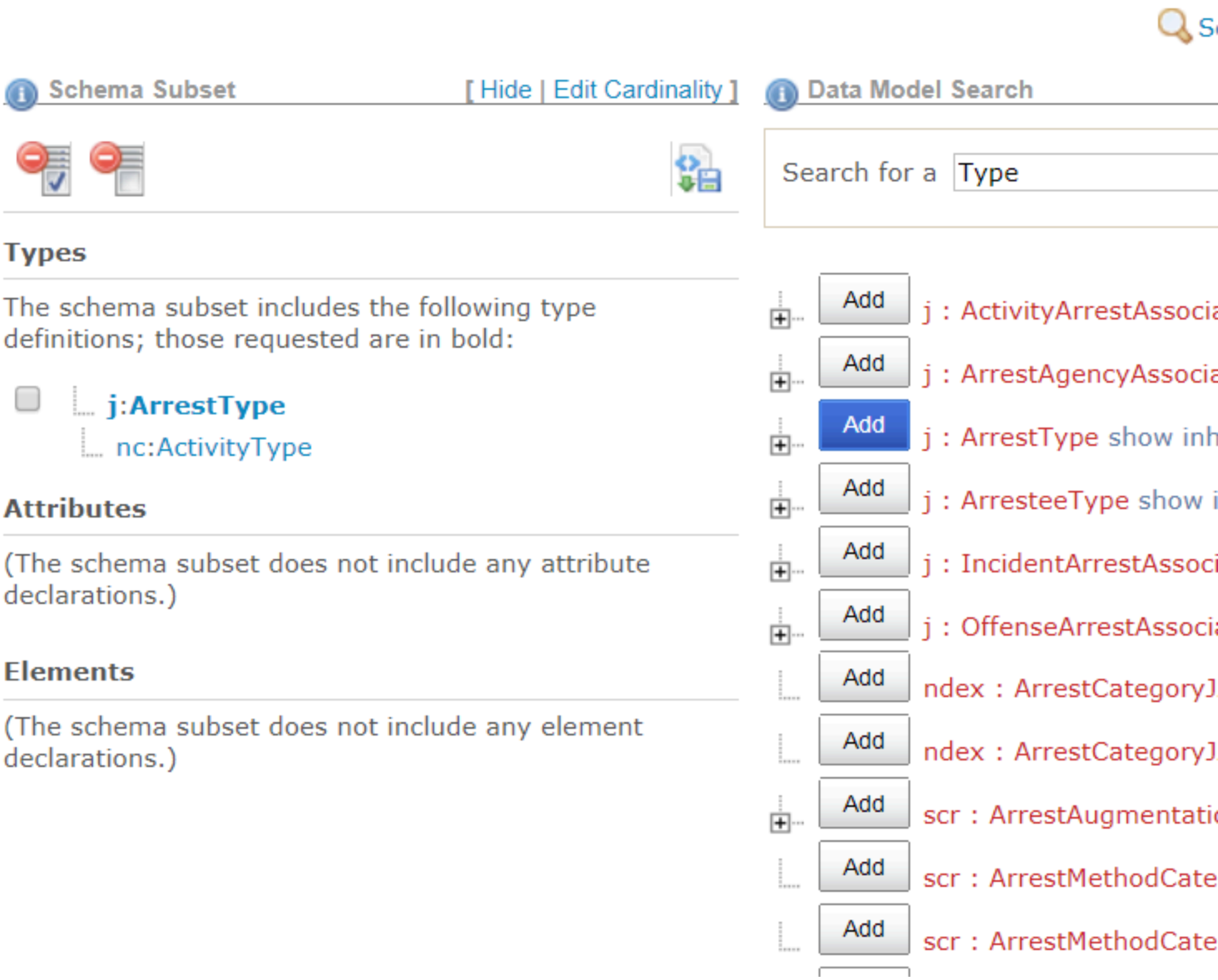
- (1) The user expands the list of sub-properties for type `nc:OrganizationType`.
- (2) The user clicks the **`Add`** button for **`Add All`**.
- (3) The SSGT adds the type and all sub-properties to the subset list on the left.
- (4) The SSGT adds the sub-properties to the Elements section of the subset list.



Add Types

When search results display a list of types, types can be added to the subset directly by clicking their `Add` buttons. This is useful for when NIEM does not have an exact match for a data requirement, but has a type that can be used directly, extended, or augmented in an IEPD.

- Switch the “Search for a” box from “Property” to “Type” or open a type information page to add a type to your subset.

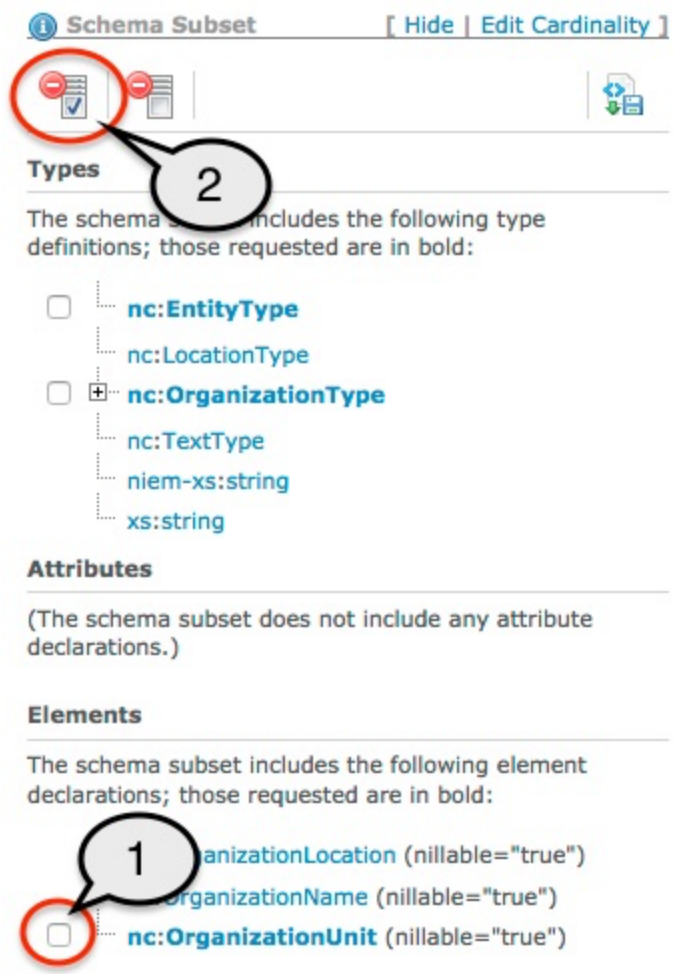


Remove Components

Properties and types may be removed from a subset.

Delete Individual Component

- Mark the checkbox next to the appropriate property, type, or sub-property in the subset list on the left.
- Click the **`Delete Checked`** button to remove all of the selections and dependencies from the subset.

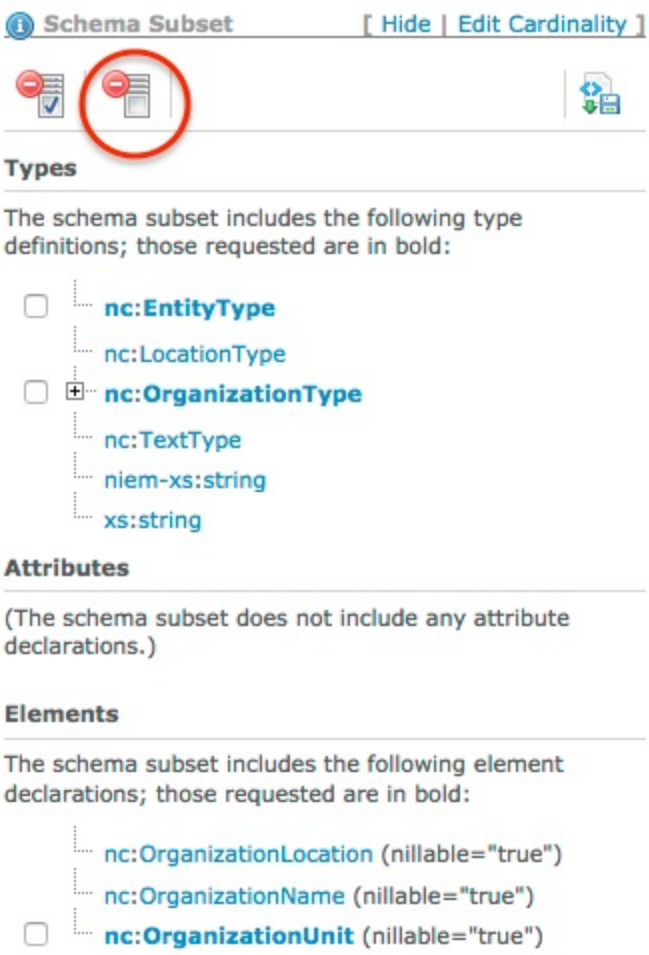


If no checkbox is available next to the component, that is because it is a dependency and cannot be removed manually as long as other components that require it are still in the subset.

Delete All

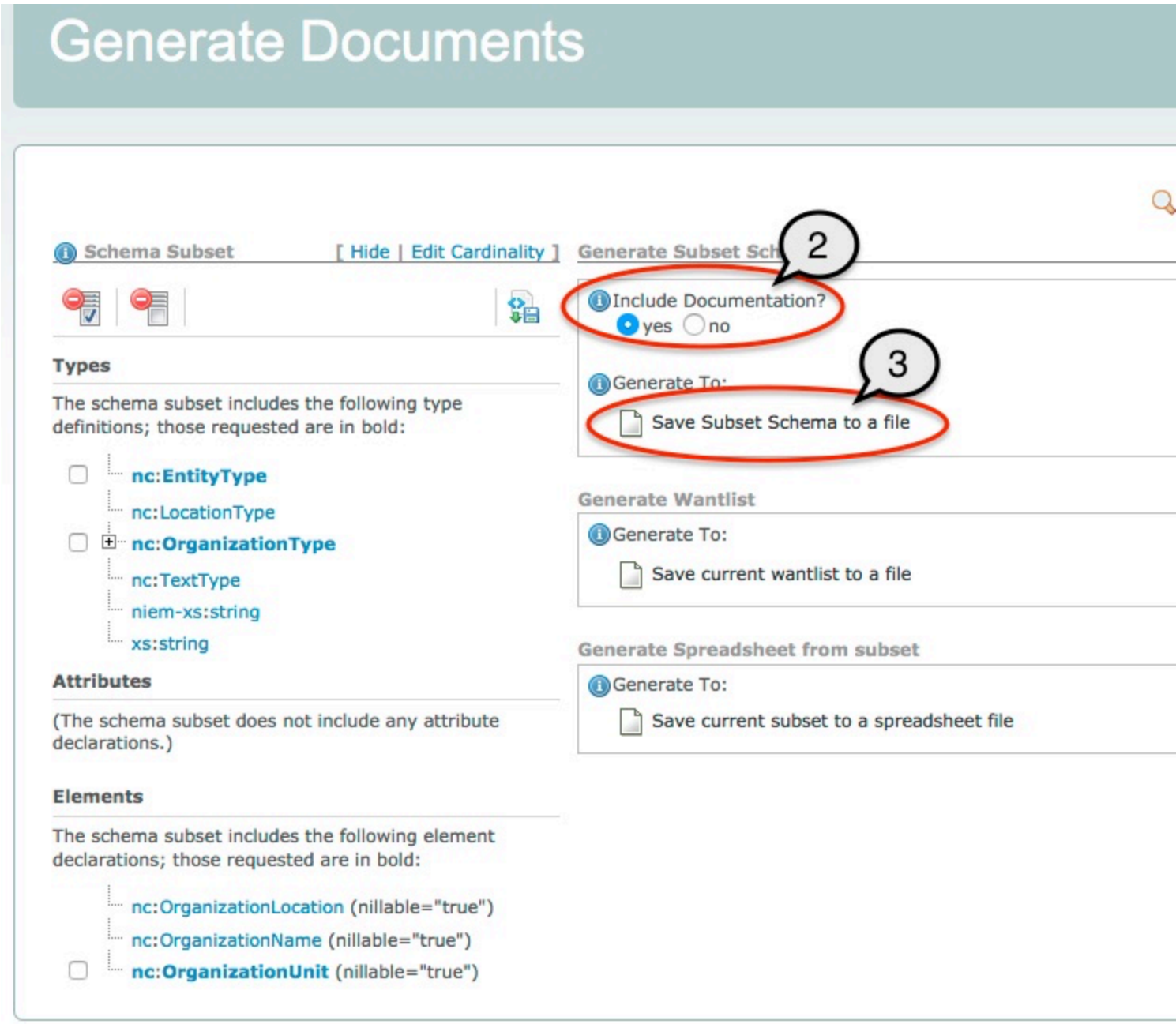
An entire subset may be removed.

- Click the **`Clear Subset`** button.



Generate Subset

- Click **`Generate Documents`**.
- Toggle the **`Include Documentation?`** radio button to either include or remove component definitions.
    - By default, this button will be checked **`yes`** and definitions will be included.
    - Set this to **`no`** if needed to minimize the size of the resulting schemas.
- Click the link to **`Save Subset Schema to a file`** to generate and download a zip file containing the subset schemas.



This subset will include a wantlist (wantlist.xml) in the zip file so that it can be loaded in the SSGT at a later date to resume work on the subset.

Generate Wantlist

A wantlist can be generated without the full subset package as a way to save work.

From the same **`Generate Documents`** page,

1. Click **`Generate Documents`**.
2. Click **`Save current wantlist to file`**.

Generate Spreadsheet

Generate a spreadsheet representation of the subset:

From the same **`Generate Documents`** page,

1. Click **`Generate Documents`**.
2. Click **`Save current subset to a spreadsheet file`**.

Options

The options page allows you to see which release is being used, change the selected release, and reload a subset (load wantlist) in order to make additional changes.

Access the options page by clicking on **`Options`** in the SSGT menu bar:

Options

Schema Subset

[Hide | Edit Cardinality]

Types

(The schema subset does not include any type definitions.)

Attributes

(The schema subset does not include any attribute declarations.)

Elements

(The schema subset does not include any element declarations.)

Wantlist Options

Current Release: NIEM-3.1

Current Nillable Default Value: true

NOTE: If you change the current release or change the current nilla
Save your current subset if you wish to keep it.

Change Release to NIEM-3.1

Change Nillable Default Value to True False

Load Wantlist

Select wantlist XML file to upload and edit:

Filename: Browse... No file se

Replace current Subset

Merge with current Subset

Load Wantlist

Download Wantlist Schema Specification

Download Wantlist Schema Specification

Namespaces

Add Everything from a Namespace to your subset (This can be slov

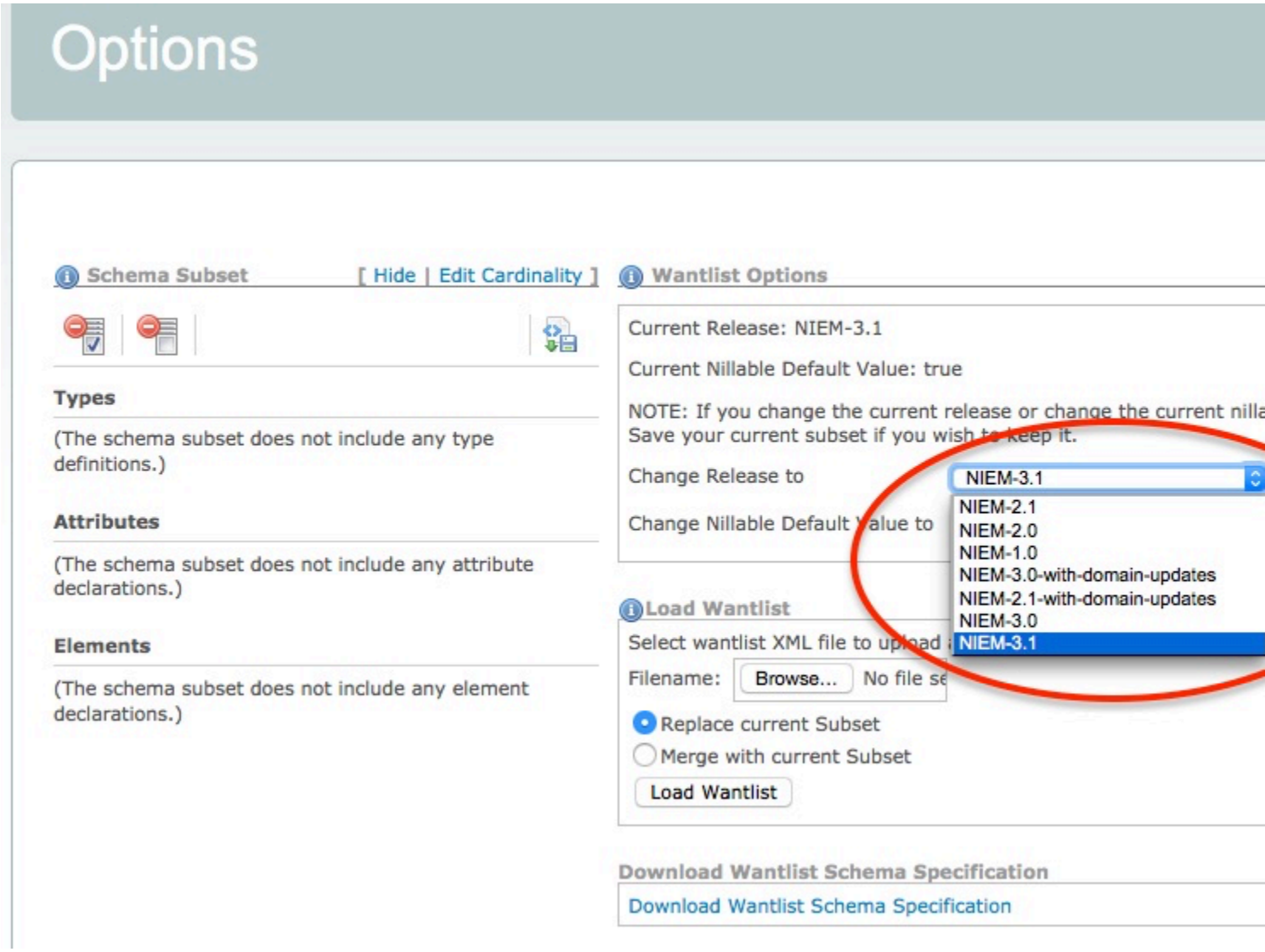
Add to subset	Downlo
Add Biometrics	Downlo
Add Chemical, Biological, Radiological, and Nuclear (CBRN)	Downlo (CBRN)
Add Children, Youth and Family Services	Downlo
Add Emergency Management	Downlo
Add Human Services	Downlo
Add Immigration	Downlo
Add Infrastructure Protection	Downlo
Add Intelligence	Downlo
Add International Trade	Downlo
Add Justice	Downlo
Add Maritime	Downlo
Add MilOps	Downlo
Add Screening	Downlo

Change Release

The top of the Options page displays which release the SSGT is using. The SSGT defaults to using the latest release.

You can switch to a different release using the drop-down list.

> The SSGT does not have the capability to migrate a subset from one release to another. This functionality only exists in the Migration Tool. **Any subset in progress will be automatically deleted if you switch to a different release.**



Resume work later

The SSGT does not provide user accounts. Instead, the SSGT provides a wantlist file as part of a generated subset package that saves your list of selections.

In order to resume work on a subset, this wantlist must be uploaded into the SSGT. Click on **`Choose`**

File to select your most recent “wantlist.xml” file, either generated individually or as part of a subset package (zip file).

If you currently have components in your subset, you can choose to either merge or remove them when uploading a wantlist.

Additional options

Nillable default

In NIEM reference schemas, elements are defined as nillable, which allows for the use of empty tags.

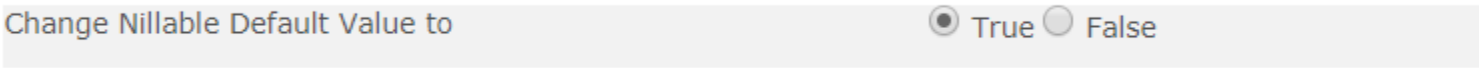
Instance examples:

```
<nc:Person/>

<nc:Person>
</nc:Person>

<nc:Person structures:ref="P1"/>
```

Schema example:

```
<xs:element name="Person" type="nc:PersonType" nillable="true">
  ...
</xs:element>
```

By default, NIEM subsets are also set to allow empty (nil) values for all elements. This default may be changed to **false** under **Wantlist Options** > **Change Nillable Default Value**.



Nillable may also be set to true or false on specific elements, which will override the default value.

Wantlist Schema

Wantlists are XML instances. The schema that defines the structure of a wantlist is available here from the options page.

Download Wantlist Schema Specification

Download Wantlist Schema Specification

> There is typically no need to view or use the wantlist schema. It is provided here for those who choose to build wantlists with another tool and use the SSGT only to calculate the dependencies and generate the subset schemas.

Domain subsets

You can add all of the content from a domain to your subset or download the domain subset directly under the `Namespaces` section.

Namespaces

Add Everything from a Namespace to your subset (This can be slow to load for large namespaces):

Add to subset	Download subset schema archive
Add Agriculture	Download Agriculture
Add Biometrics	Download Biometrics
Add Chemical, Biological, Radiological, and Nuclear (CBRN)	Download Chemical, Biological, Radiological, (CBRN)
Add Emergency Management	Download Emergency Management
Add Human Services	Download Human Services
Add Immigration	Download Immigration
Add Infrastructure Protection	Download Infrastructure Protection
Add Intelligence	Download Intelligence
Add International Trade	Download International Trade
Add Justice	Download Justice
Add Maritime	Download Maritime
Add MilOps	Download MilOps
Add Screening	Download Screening
Add Surface Transportation	Download Surface Transportation

Conformance Testing Assistant (ConTesA)

The **Conformance Testing Assistant (ConTesA)** enables users to test NIEM XML schemas against the automated rules from the NIEM Naming and Design Rules (NDR). This is an important first step in verifying NIEM conformance.

The current version of ConTesA works for NIEM XML schemas based on version 3.0 of the NDR or later.

- Validates XML schemas against the automated rules of the NIEM NDR.
- Generates a conformance report available in spreadsheet, HTML, and XML formats.
- Tests single XML schemas or multiple schemas bundled in a zip file.
- Saves conformance reports to the user accounts.

Do not upload sensitive, classified, or distribution-restricted materials to ConTesA.

- Scope
  - NDR Schematron rules
- Out of scope
  - NDR text rules
  - IEPDs
  - XML validation
  - JSON
- Usage
- Conformance Report
- FAQ

Scope

NDR Schematron rules

Some of the rules that the NDR defines are written in Schematron. ConTesA is able to run automated validation on all such rules.

The image below shows a rule written in Schematron from the NDR:

Rule 10-48. Attribute name begins with lower case letter

[Rule 10-48] (REF, EXT) (Constraint)

```
<sch:pattern>
  <sch:rule context="xs:attribute[exists(@name)]"
    <sch:assert test="matches(@name, '^[a-z]')"
      >Within the schema, any attribute declarati
      ('a'-'z').</sch:assert>
  </sch:rule>
</sch:pattern>
```

Out of scope

NDR text rules

ConTesA cannot verify that a schema is fully NIEM conformant; only that it passes the set of automatable Schematron rules from the NDR. The NDR also has free-text rules that cannot be automated and must be checked manually.

The image belows shows a rule written in free text from the NDR:

Rule 10-47. Names use camel case

[Rule 10-47] (REF, EXT) (Constraint)

The name of any XML Schema component defined by the

IEPDs

ConTesA can check the NDR conformance of the XML schemas within an IEPD. The IEPD specification, however, defines an additional set of conformance rules for IEPDs and for some of the artifacts within. Ensuring that an IEPD meets all applicable rules currently requires manual review.

XML validation

ConTesA does not check for XML schema or instance validation. This is required for NIEM conformance, but there are many tools available that already do this, including XMLSpy, Oxygen XML Editor, and Xerces.

JSON

There is no current automated support for NIEM JSON conformance validation.

Usage

Log in

Register for a new account or log in.

Upload a file

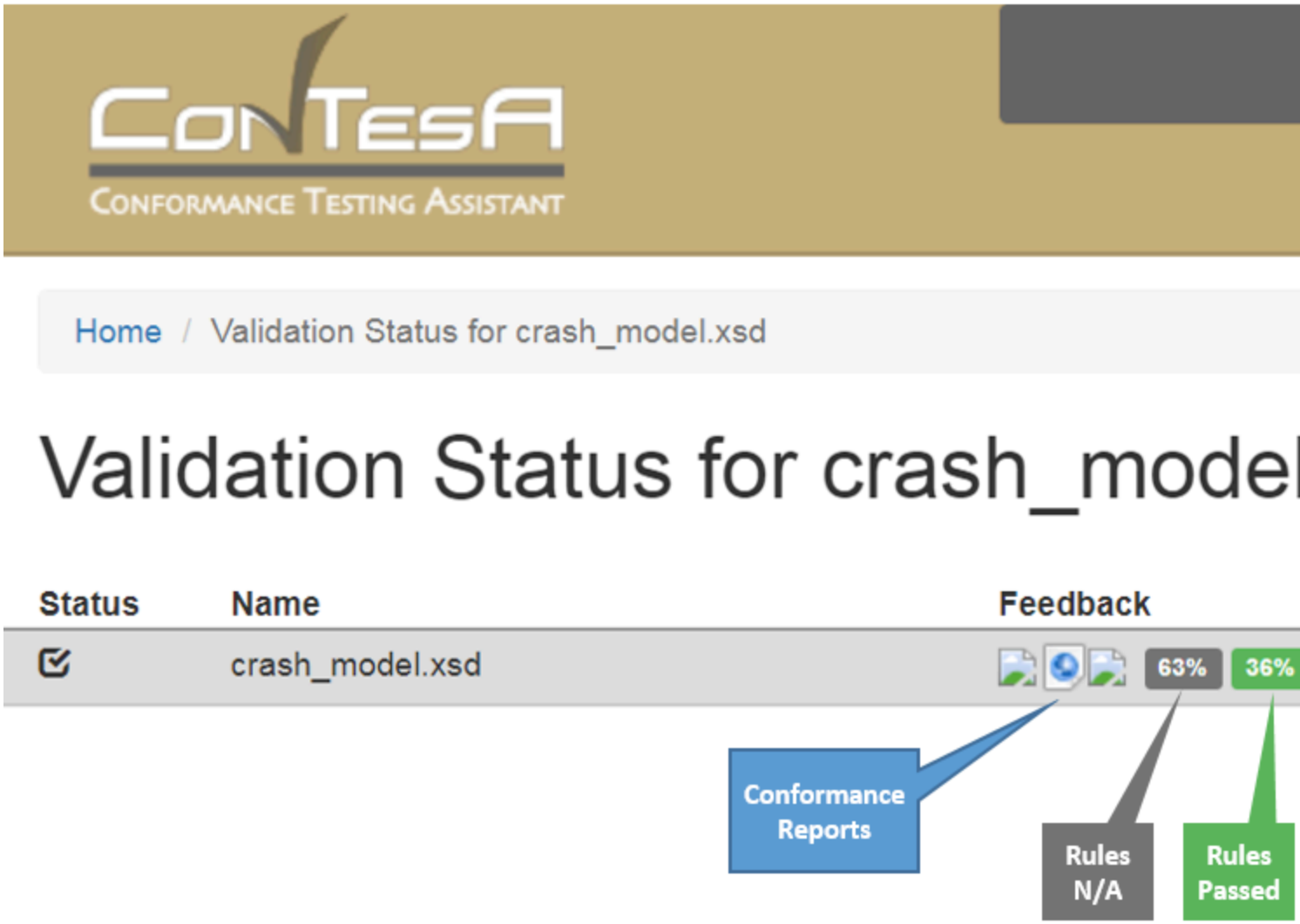
This can be either a single NIEM XML schema or a set of schemas packaged together in a zip file.

> Because ConTesA does not perform XML schema validation, it is not necessary to upload a NIEM release or NIEM subset along with your own files.

Review results

After ConTesA has finished validation, review the status panel for highlights and the conformance reports. You may need to refresh the page to get the links for the conformance reports.

> The image below shows the validation status panel for a single schema that was uploaded to ConTesA, `crash_model.xsd`. There, you can see an overview of the results and view or download a conformance report for more information.



> There may be some or many rules from the NDR that will not be applicable to your schema. For example, if your schema does not include any simple union types, then no NDR rules related to simple union types will be executed. This is the "Rules not executed" count that is represented in the gray box. This is a different count than the number of NDR rules that require manual review, which is not reported.

Confirm all applicable files were tested

The NDR requires a `ct:conformanceTargets` attribute on all NIEM XML schemas. The value of this attribute informs ConTesA of which rule set(s) to use for testing. Without this attribute, no conformance tests will be run.

> See the section about the Conformance Targets Attribute Specification for more information about this attribute and examples of how to define it.

Any file that does not include the `ct:conformanceTargets` attribute, or any file that was not able to be tested for other reasons (such as not being well-formed XML), will appear under the line for **"No Rules to Run"**. Be sure to check this list, fix all applicable schemas that should be tested, and re-run

them in ConTesA.

- The image below shows example schema `highway_model.xsd` that did not include a `ct:conformanceTargets` attribute and was therefore unable to be validated by ConTesA.
- The missing attribute should be added to the schema and then it should be retested.

Conformance Report

A conformance report will be available in three formats: HTML, spreadsheet, and XML.

Summary

The conformance report will include a summary section, listing the number of rules that passed and failed.

ConTesA Conformance Report

Generation Date: Thu Dec 06 13:44:19 EST 2018
Validation Object: crash_model.xsd

Skip to: Rule Contexts Summary | Rule Results | Auto Fail Rules | Manual Fail Rules | All Rules

Rule Summary

Statistical information about all passing/failing rules in the system.

Auto Pass - 52 (36%)
There were 52 rules that automatically passed validation. The "Auto Pass" category describes rules that are automatically executed by ConTesA, and result in a "Success" or "Information" status.

Auto Fail - 1 (0%)
There were 1 rules that failed validation. The "Auto Fail" category describes rules that are automatically executed by ConTesA, and result in a "Warning", "Failure", or "Error" status. Each rule may have failed one or more times. See the "Auto Fail Rules" section for specific locations of the failures in your schema file(s). This category may also contain "Partial" rules. Partial rules are rules that cannot be fully automated by ConTesA and require the user to verify their status. Partial rules are placed into the "Auto Fail" category if their automated checks result in a status of "Warning", "Failure", or "Error".

Manual Pass - 0 (0%)
There were 0 rules that were manually verified by the user. The "manual" category describes rules that must be checked by the user because they are either not yet implemented, cannot be automated, or are only partially implemented in ConTesA. "Partial" rules are in the manual category, because the user must verify the portions of the rule ConTesA cannot automate. A rule is considered to be in the "Manual Pass" category if a user has verified that the manual rule will result in a "Success" status.

Auto Fail Rules

The key section to review from the conformance report lists the rules that the schema did not pass. Use the line number from the report to help track down the error or warning, and look up the rule number in the NDR if you need more information about it.

Rule Results (8): crash_model_bad.xsd

There were 8 results from executing against validation object 'crash_model_bad.xsd'.

Status	Line	Rule	Message
Auto Pass	14	Schema is CTAS-conformant	Rule 4-3: The document MUST be a conformant document as defined by the NIEM Conforman Specification.
Auto Pass	14	Target namespace is absolute URI	Rule 9-83: The value of the attribute targetNamespace MUST match the production <absolute- RFC 3986.
Auto Pass	14	Document is a schema document	Rule 7-3: The document MUST be a schema document.
Auto Pass	14	Document is an XML document	Rule 7-1: The document MUST be an XML document.
Auto Pass	14	Document uses XML namespaces properly	Rule 7-2: The document MUST be namespace-well-formed and namespace-valid.
Auto Pass	30	Standard opening phrase for element	Rule 11-31: The data definition for an element declaration with a name SHOULD begin with the phrase "(a|an)".
Auto Pass	38	Standard opening phrase for complex type	Rule 11-32: The data definition for a type SHOULD begin with the standard opening phrase "a c
Auto Fail	44	No xs:all	Rule 9-60: The schema MUST NOT contain the element xs:all

FAQ

I can't logout of ConTesA?

Try deleting your browser's data cache and cookies.

Where are my files?

The files uploaded for conformance testing are not stored on the ConTesA servers. Data exists temporarily in memory on the servers while the files are being processed; once processing is complete the data is cleared from memory.

What is my username?

Your *username* is the email address you used for registration with the ConTesA site.

I don't remember my registration email, how do I login?

Unfortunately, we cannot retrieve your user information without record of the email address you used to register.

Where is my user data stored?

Your user data is securely stored on the ConTesA servers.

What user data do you have?

ConTesA only stores the user data entered for registration as well as a record of the file names and their associated conformance reports.

Movement

***Movement* - an easier, simpler way to explore and use NIEM.**
Visit Movement

Movement enables you to search and explore the content of the NIEM model. It provides a user friendly interface, smarter search results, and a streamlined way to build JavaScript Object Notation (JSON) Schema in support of exchanges. It is also open source: the community who inspired its creation can contribute to it.

- Search and explore the content of NIEM.
- Build simple JSON subsets of the current release.
- Open-source

- Discover the possibilities
- Search the Business Glossary
- Build your exchange
- Got feedback?
- How to contribute
    - Tool Developers
    - Tool Users
- Join the Movement
- Additional Resources

Discover the possibilities

Created with input from the NIEM community, Movement provides a user friendly interface, smarter search results, and a streamlined way to build JavaScript Object Notation (JSON) Schema in support of exchanges.

Simple and straightforward, Movement removes the added challenge of understanding NIEM's technical architecture when exploring its common vocabulary, allowing the Program to better support evolving community needs. Movement allows anyone to explore what NIEM has to offer, regardless of model expertise or technical understanding.

Movement is also open source so the community who inspired its creation can contribute to it.

embed/
hOAfTlpA5tA?rel=0

Search the Business Glossary

The Business Glossary—a lightweight, language-agnostic version of NIEM's 12,000+ terms, definitions, and relationships—powers Movement's search functionality. Movement also offers more personalized, laser-focused search results—including options to filter, search by keyword, or sort by relevance—so that users can find what they're looking for faster.

Movement supports the current NIEM release: NIEM 5.0.

Build your exchange

For those who want to take their exploration a step further, users can quickly choose what they need from the search results and add them to Movement's "My Subset Builder" with just the click of a button. You can store the selected terms locally in your browser and generate/download customized JSON Schema. The JSON Schema formatting and technical dependencies are built into the schema generation functionality, allowing you to leverage NIEM terms and definitions and incorporate them into your JSON exchange technical architecture.

While building your exchange, you'll notice that Movement only allows you to add certain terms to your subset. Based on user feedback, some business rules are built into the tool. Movement does not allow the selection of NIEM complex types—these cater to the structure and relationship of the model instead of the validation aspects needed from a schema. By allowing the user to only select NIEM simple types, the tool creates a more approachable environment for NIEM users and JSON users alike, while focusing on the intended JSON Schema solution.

The conventional JSON Schema output of the tool is as simple as we can make it and demonstrates the Program's progress towards supporting organizations that want to develop a NIEM-based exchange solely using JSON. Some may argue it's too simple, and for others it's exactly what they need. We've built it as much as we can without broad user feedback. We would like to start the conversation to find

out what users would like to add back—it's in Beta until we do.

To learn more about what NIEM is doing with JSON, the NIEM JSON section on this site is a great place to start.

Got feedback?

We want to hear from you!

In the spirit of open-source tooling, we have provided a Scrum board that allows users to keep track of Movement's issues and enhancements. Anyone can submit a new issue for the tool for something they would like to see added or a bug. Once reviewed by the program office, issues will be added to the Scrum board's backlog. Developers and tool contributors can then address issues from the backlog and track their status using the Scrum board—providing an Agile approach to development and complete transparency to users.

Don't have a GitHub account and want to provide input? Please use our contact us page on NIEM.gov.

How to contribute

Tool Developers

We have ideas for additional functionality we'd love to see get implemented in the tool. If you want to join the Movement, here are a few suggestions for contributions:

- Extension schema functionality for things that are not in NIEM
- Have the user interface display the Code list options
- Export formats other than JSON Schema (XSD, Excel Spreadsheet, Java Library, .NET Library, NIEM UML, etc.)
- Implement profiles and dashboards to build your exchanges with a group
- Organizational data dictionary import
- Import of existing NIEM exchanges

NIEM has created an automated build pipeline to allow developers an easy way to modify and improve the tool by building off the source code and submitting their pull requests.

Tool Users

We are very interested in hearing your vision for Movement as well as your general feedback on its capabilities. You can provide your feedback directly on GitHub or though NIEM.gov.

Join the Movement

Movement is an exciting glimpse into the future of NIEM, and we can't wait to see what the community accomplishes by using it. Join the movement today—visit beta.movement.niem.gov to access the tool.

Additional Resources

- Movement Quick User Guide

  User guide explaining how to use Movement to search the current NIEM release and build partial NIEM JSON subsets.

- Movement and JSON Schema

  The current status of Movement's support of NIEM JSON subsets.

Movement Quick User Guide

The purpose of Movement is to help users navigate the model and create their own subsets to define an organization's data exchanges.

- Landing Page
- Search Results Page
- Details Page
- Subset Builder Page
- Header & Footer
- 'Add to Subset' Button

Landing Page

When a user visits beta.movement.niem.gov, they are automatically directed to Movement's Landing page.

Here you can input any keyword into the search bar and start to discover NIEM terms and definitions. If desired, the dropdown next to the keyword search box may be used to filter the results to a specific domain.

Search Results Page

Once a keyword has been searched, the Search Results page displays all the relevant terms for that specific keyword.

On this page, you can narrow down the results by one or several domains using the 'Filter by' section. You can also use the 'Sort by' dropdown to change the order of the results.

In addition, you can add terms directly from this page to your Subset Builder by clicking the 'Add to Subset' button for the specific term. You can also click on the term's name to visit its Details page.

For more information on why some terms have an 'Add to Subset' button and other don't, please review the 'Add to Subset' Button section below.

Details Page

You can access the Details page of a term by clicking its name. Here you can learn more about the chosen term, its definition and type, as well as locate any children terms desired to be in your subset. If the chosen term is a parent, you can add any children terms directly from this page to your Subset Builder by clicking the 'Add to Subset' button.

For more information on why some terms have an 'Add to Subset' button and other don't, please review the 'Add to Subset' Button section below.

Subset Builder Page

You can visit the ‘My Subset Builder’ page at any point to review your current subset.

Here, a user can review the terms that have been added to their subset, remove any (or all) terms not needed, and click on the specific term’s name to visit their Details page. As a user, you can export the JSON Schema format associated with the selected terms by clicking the ‘Download JSON Schema’ button at any time.

Currently, the content in a user’s ‘My Subset Builder’ is storied locally in the browser. Remember, we’re in beta and looking for your feedback directly on GitHub or though NIEM.gov!

Header & Footer

Every page in Movement has a header and footer that remains consistent.

The header contains a search bar that allows users to quickly search another keyword without going back to the Landing page. It also contains the ‘My Subset Builder’ button where you can access the subset currently being developed.

There are several other useful links that you can access within the header and footer—you can visit the NIEM.gov website and contact us page using the ‘NIEM.gov’ and ‘Contact Us’ links respectively; access the open source GitHub repo using the ‘Join the Project’ button; go to the Movement Overview page by clicking the ‘Learn More about Movement’ link; and even download the latest version of the authoritative NIEM model by clicking the ‘Download Current Model’ link.

‘Add to Subset’ Button

You’ll notice that Movement only allows you to add certain terms to your subset. Based on user feedback, some business rules are built into the tool. The NIEM model has relationships that can be explained at a high level as parents (or complex types) and children (or simple types). Movement does not allow the selection of NIEM complex types—these cater to the structure and relationship of the model instead of the validation aspects needed from a schema. By allowing the user to only select NIEM simple types, the tool creates a more approachable environment for NIEM users and JSON users alike, while focusing on the intended JSON Schema solution.

We’ve built the tool as much as we can without broad user feedback. We would like to start the conversation to find out what you’d like to add back—it’s in Beta until we do. For example, if you think complex types are important for your schema output, we could use your help to identify the best way to include them, perhaps automatically in the schema output. You can provide your feedback directly on GitHub or though NIEM.gov. Ideally, we’ll have multiple people who share the same idea and help us craft the business rules to hand it off to the development team or hold a hackathon to get it built!

Movement and JSON Schema

- Target audience
- What Movement puts out today
- Sample Use Case Scenario
- Sample JSON Schema output
- Tool output summary
    - Enums
    - Simple types
    - JSON-LD features

Target audience

Movement was build on the assumption of two types of JSON users:

1. **Those who would like to leverage only the NIEM terms and definitions when building custom JSON objects for a lightweight exchange.**

   For this user type, we have the current JSON Schema output of the Beta Movement Tool. It's as simple as we can make it—through a lightweight version of the model content, and the ability to only select simple types from it, you can auto generate a conventional JSON Schema. Some may argue it's too simple, and for others it's exactly what they need. We realize that this is a different approach to what NIEM has traditionally offered, such as hierocracy and relationships of model content. We've built it as much as we can without broad user feedback. We would like to start the conversation to find out what you'd like to add back—it's in Beta until we do.

2. **Those who would like to leverage the content in NIEM as well as its relationships for more complex exchanges that require context.**

   This user type requires JSON Schema functionality that is more reflective of what you can do today with NIEM in XML–it takes into consideration things like reusability, hierarchy structure, and cardinality. This solution also establishes the foundation upon which JSON-Linked Data (JSON-LD) can occur. To learn more about this approach and access guidance for how to use it, visit our JSON landing page on GitHub.

   In addition, starting with the NIEM 4.0 release, NIEM provides a default JSON-LD context for NIEM namespaces. The JSON-LD context is a JSON file that identifies, for every namespace for NIEM content, the namespace prefix and its corresponding namespace URI.

Help us drive this effort forward by telling us what functionalities you will actually use!

We drafted up a sample use case scenario for you to visualize both approaches to help understand details for each.

What Movement puts out today

Movement provides standard JSON Schema best practices and types, is human and machine readable,

and validates your simple type JSON objects while leveraging NIEM terms and definitions.

Advantages:

- Smaller/more concise JSON Schema output
- Readable
- No custom placeholder elements
- Uses JSON specific types

Drawbacks:

- Lack of cardinality, enum descriptions, substitution groups, and parent-child structured relationships/context
- No JSON-LD adaptability

Sample Use Case Scenario

User picks 4 simple elements for the exchange:

- VehicleAxleQuantity
- IdentificationID
- Amount
- CurrencyCode

Sample JSON Schema output

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "additionalProperties": false,
  "properties": {
    "nc:VehicleAxleQuantity": {
      "description": "A count of common axles of rotation of one or more wheel
      "type": "number"
    },
    "nc:Amount": {
      "description": "An amount of money.",
      "type": "number"
    },
    "nc:CurrencyCode": {
      "description": "A unit of money or exchange.",
      "$ref": "#/definitions/iso_4217:CurrencyCodeType"
    },
    "nc:IdentificationID": {
      "description": "An identifier.",
      "type": "string"
    }
  },
  "definitions": {
```

```
    "iso_4217:CurrencyCodeType": {
      "description": "A data type for a currency that qualifies a monetary amo
      "allOf": [
        {
          "$ref": "#/definitions/iso_4217:CurrencyCodeSimpleType"
        }
      ]
    },
    "iso_4217:CurrencyCodeSimpleType": {
      "description": "A data type for a currency that qualifies a monetary amo
      "enum": ["EUR","GBP","USD"]
    }
  }
}
```

Tool output summary

Uses conventional JSON Schema notation to replicate NIEM simple types.

Enums

Enums are represented in the conventional JSON Schema way. This schema approach does not provide the description for each option of an enum type element within the JSON Schema.

For example:

```
"enum": ["1","2","3"]
```

Simple types

Simple types do not take cardinality into consideration. This schema approach allows developers to use single key to single value within their JSON objects.

For example:

```
"Amount": 12
```

JSON-LD features

Does not take into account eventual JSON-LD adaptability or context.

Need a more robust solution? Please see the NIEM JSON section for the guidance on how to build JSON Schema that supports more features of NIEM and JSON-LD.

Migration Tool

The **Migration Tool** enables you to upgrade an XML Schema release subset to a newer version. This supports the process to update an existing exchange to use a more recent NIEM release.

Migrations may only be made over consecutive releases. It will be necessary to run multiple migrations to convert to a non-consecutive release. For example, to migrate a 3.1 release subset to 4.0, two migrations must be made: 3.1 to 3.2, and then 3.2 to 4.0.

- Release subset migrated to a newer version
- Updated wantlist
- Migration report that contains status information and any issues encountered

The migration tool only migrates release subsets. Local NIEM schemas must be updated manually.

NIEM resources for the Oxygen XML Editor

The Oxygen XML Editor is a commercial multi-platform XML editor, XSLT/XQuery debugger and profiler with Unicode support. NIEM provides some additional information and resources for this tool that may assist with domain and IEPD schema development and conformance testing.

- Check NDR Conformance Using Schematron Validation

  Use Oxygen to test a XML schema for NDR conformance.

- NIEM Snippets

  NIEM XML follows a regular, well-defined syntax. Snippets can be used to quickly and consistently fill in standard syntax, leaving users to fill in the blank or partially-filled in values for fields like name and definition.

- Using XML Catalogs with Oxygen

  Use XML catalog files in Oxygen to specify or override schema file locations.

Check NDR Conformance Using Schematron Validation

- Setup
- Schematron Validation
- Review Conformance Errors
- Related Links

ConTesA is a free web tool provided by the NIEM program that also may be used to check NDR conformance.

Setup

Download the NDR schematron rules. These are available in the NDR zip file and on the NDR GitHub repo.

Schematron files have a `.sch` file extension.

The Schematron file(s) will be needed each time you test NDR conformance. The two key files for this task are:

- `ndr-rules-conformance-target-ext.sch` - Schematron rules for EXT schemas
- `ndr-rules-conformance-target-ref.sch` - Schematron rules for REF schemas

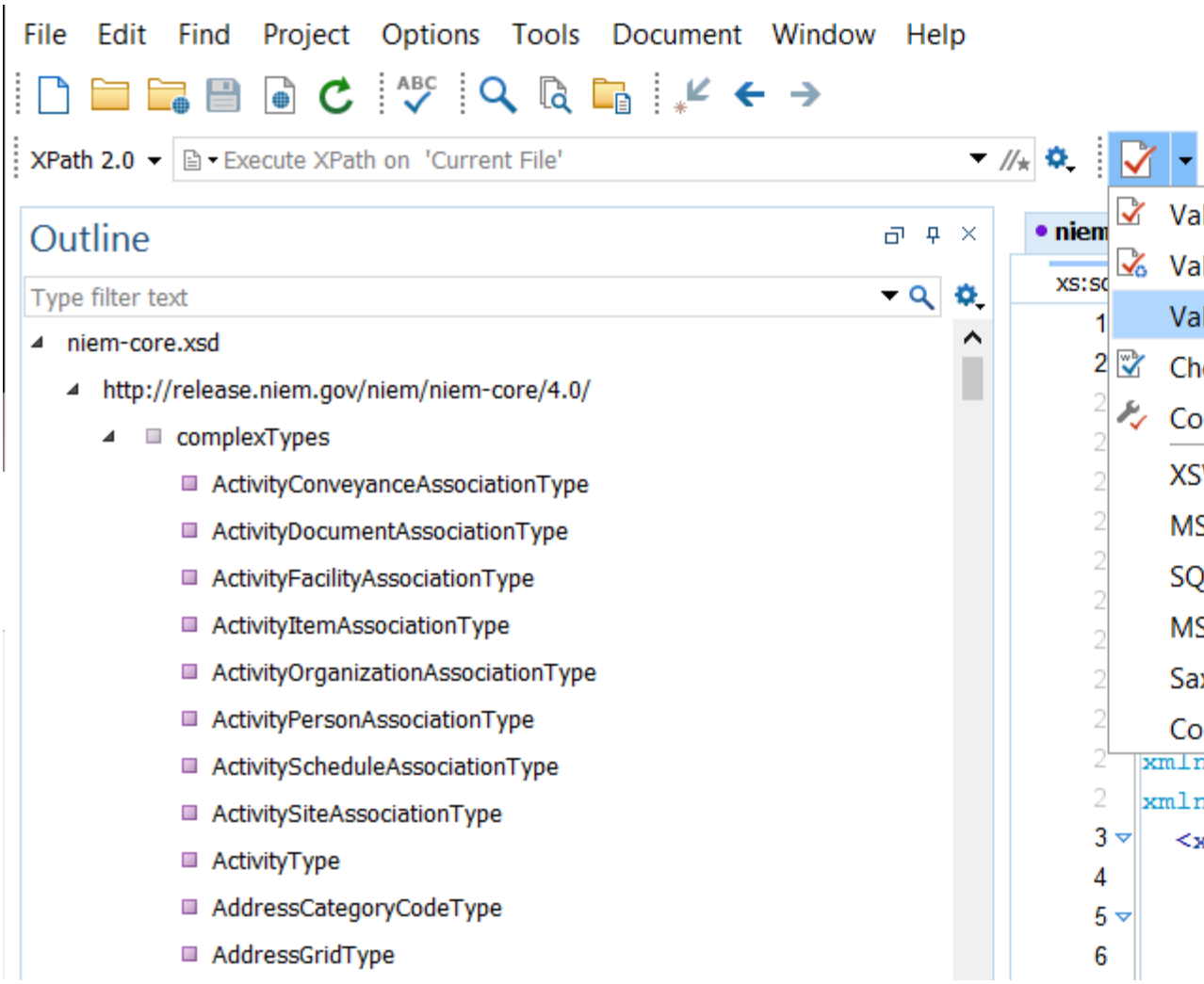
Schematron Validation

- In Oxygen, open the NIEM XML schema to test for NDR conformance. This might be an IEPD extension schema or a NIEM domain schema under development.

- Open the "Validate with…" dialog box. There are three different ways of doing this:

    - (Option 1) From the menu bar, click on Document > Validate > Validate with…

    - (Option 2) Click the drop-down on the `Validate` (check mark) button and select `Validate with...`, as shown below.



- (Option 3) From an open Oxygen project, right-click on one or more selected schema files or folders, as shown below.

File Edit Find Project Options Tools Document Window Help

XPath 2.0 | Execute XPath on 'Current File'

Project

niem-release-git.xpr

- niem-release-git.xpr
- **niem**
  - adapters
  - codes
  - domains
  - external
  - niem-core
    - 4.0
      - niem-core.xsd
  - proxy
  - utility
  - xml-catalog.xm

Outline

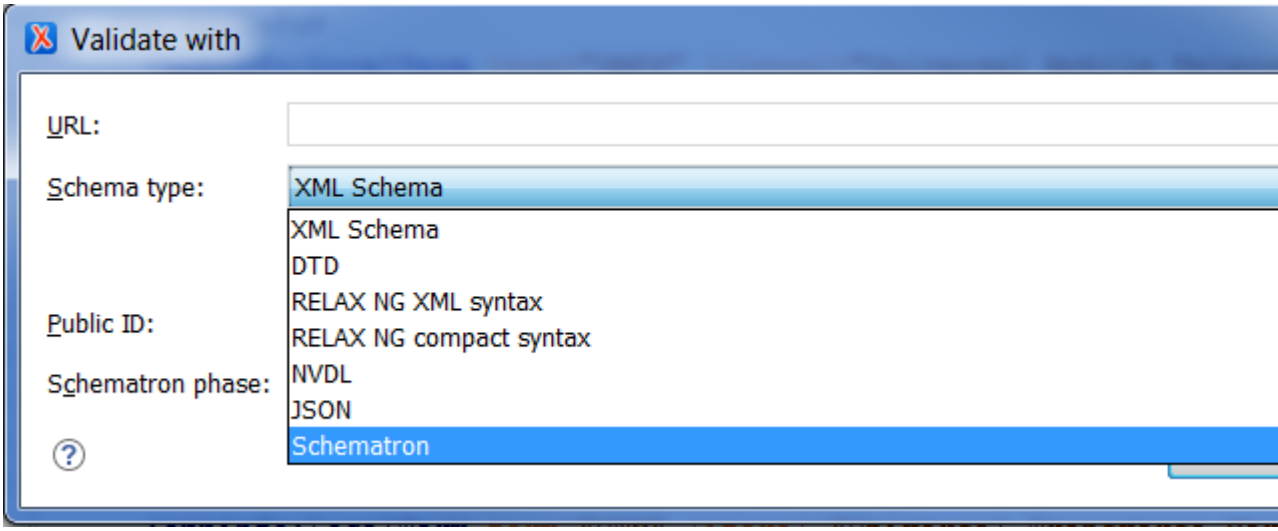
Type filter text

- niem-core.xsd
  - http://release.niem.gov/niem/niem-core/4.0/
    - complexTypes
      - ActivityConveyanceAssociationType
      - ActivityDocumentAssociationType
      - ActivityFacilityAssociationType
      - ActivityItemAssociationType
      - ActivityOrganizationAssociationType
      - ActivityPersonAssociationType
      - heduleAssociationType
      - eAssociationType
      - pe
      - tegoryCodeType
      - idType
      - ype
      - pe
      - e
      - pe
      - sureType
      - easureType
      - nuteType
      - teMeasureType
      - condType
      - ype
      - e
      - BearingTy
      - BinaryHasl
      - BinaryHashType

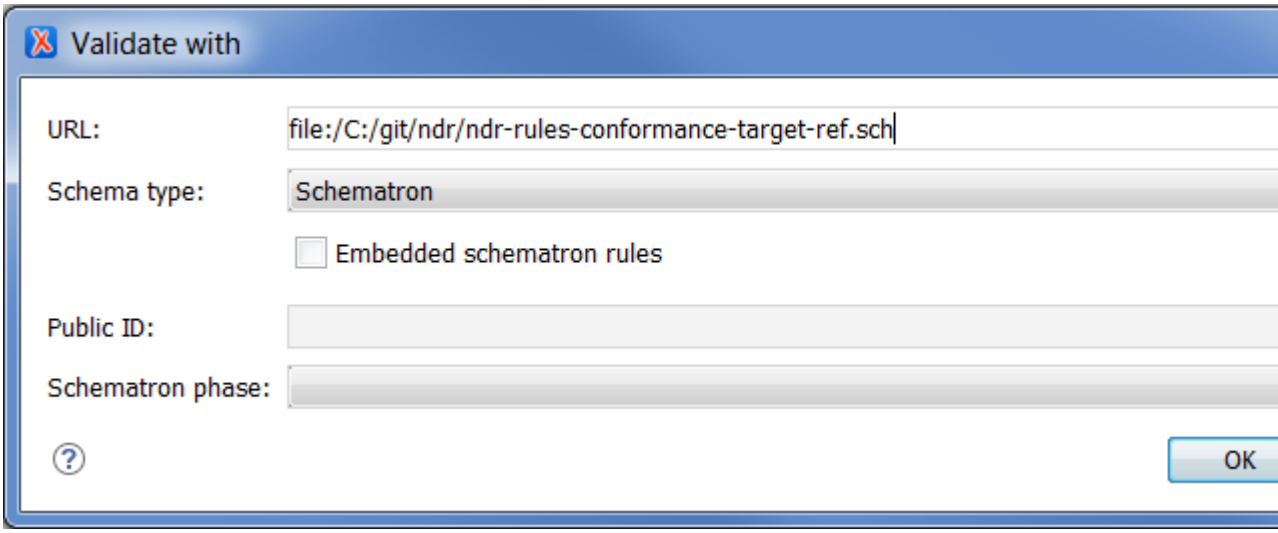
New	
Open	
Show in Explorer	
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Remove from Disk	Shift+Delete
Refactoring	
Refresh	F5
Find/Replace in Files...	Ctrl+Shift+H
XPath in Files...	
Open/Find Resource...	Ctrl+Shift+R
Check Spelling in Files...	
Format and Indent Files...	
Transform	
Validate	
Detect Master Files...	

- Check Well-Forme
- Validate
- Validate with Sche
- Configure Validatio

- In the `Validate with` dialog box…
  - Change the `Schema Type` field from `XML Schema` to `Schematron`.



  - In the `URL` field, browse to the downloaded Schematron file. Make sure to choose the appropriate file (REF or EXT) for your NIEM schema.
    - *IEPD extension schemas typically use EXT rules.*
    - *Domain schemas use REF rules.*



  - Click OK to run the conformance validation.

Review Conformance Errors

Conformance errors and warnings will appear in the `Results` panel.

- The NDR rule number appears, along with the rule title, so that the NDR may be easily referenced for more information about the rule.

- Double-click on an error or warning to jump to that line in the schema.

The image below shows several modifications to a local copy of Core in order to demonstrate errors:

niem-core.xsd [C:\git\niem-releases\niem\niem-core\4.0\niem-core.xsd] - <oXygen/> XML Editor

File Edit Find Project Options Tools Document Window Help

XPath 2.0 | Execute XPath on 'Current File'

Outline

Type filter text

- niem-core.xsd
  - http://release.niem.gov/niem/niem-core/4.0/
    - complexTypes
      - ActivityConveyanceAssociationType
      - ActivityDocumentAssociation
      - ActivityFacilityAssociationType
      - ActivityItemAssociationType
      - ActivityOrganizationAssociationType
      - ActivityPersonAssociationType
      - ActivityScheduleAssociationType
      - ActivitySiteAssociationType
      - ActivityType
      - AddressCategoryCodeType
      - AddressGridType
      - AddressType
      - AircraftType
      - AirportType
      - AmountType

niem-core.xsd*

xs:schema xs:complexType

```
56          <xs:element ref="nc:Acti
56 maxOccurs="unbounded"/>
57        </xs:sequence>
58      </xs:extension>
59    </xs:complexContent>
60  </xs:complexType>
61  <xs:complexType name="ActivityDo
62    <xs:annotation>
63      <xs:documentation>A data typ
64    </xs:annotation>
65    <xs:complexContent>
66      <xs:extension base="nc:Assoc
67        <xs:sequence>
68          <xs:element ref="nc:Acti
69          <xs:element ref="nc:Docu
70          <xs:element ref="nc:Acti
71        </xs:sequence>
72      </xs:extension>
73    </xs:complexContent>
74  </xs:complexType>
```

Rule 9-25: A complex type MUST have a data definition.

Text Grid Author Design

Results

	Info	Description - 8 items
	–	E [ISO Schematron] Rule 9-25: A complex type MUST have a data definition.
	–	E [ISO Schematron] Rule 10-2: An object type with complex content MUST be derived from structures:ObjectTy
	–	E [ISO Schematron] Rule 10-21: A type MUST have an association type name if and only if it is derived from an
	–	E [ISO Schematron] Rule 10-23: An augmentable type MUST contain an element use of its augmentation point e
	–	E [ISO Schematron] Rule 10-25: A schema document containing an augmentation point element declaration MU
	–	E [ISO Schematron] Rule 10-28: An augmentation point element MUST only be referenced by its corresponding
	–	E [ISO Schematron] Rule 11-1: A type definition schema component that does not define a proxy type MUST hav
	–	E [ISO Schematron] file:/C:/git/niem-releases/niem/niem-core/4.0/niem-core.xsd:5141: type not found: {http:/

Problems

After making corrections, rerun the steps above to test conformance on the updated schema.

Related Links

Please see Oxygen's video for more information about Schematron validation in Oxygen.

NIEM Snippets

NIEM XML follows a regular, well-defined syntax. Snippets can be used to quickly and consistently fill in standard syntax, leaving users to fill in the blank or partially-filled in values for fields like name and definition. Snippets are currently provided for NIEM XML constructs.

- Overview
- Demo
- Available NIEM snippets
  - Property snippets
  - Type snippets
  - Combo snippets
  - Schema snippets
  - Other snippets
- Conventions
  - Blanks
  - Dots (…)
  - ALLCAPS
  - Ordering
- Using NIEM snippets with Oxygen
  - Import snippets
  - Use a snippet
  - Modify a snippet
  - Reload snippets

Overview

The following is an example of a snippet or a template to create a basic element in NIEM. It includes the standard schema syntax required by the NDR, but leaves the fields for the element's name, type, and definition blank for the user to fill in:

```
<xs:element name="" type="" nillable="true">
  <xs:annotation>
    <xs:documentation></xs:documentation>
  </xs:annotation>
</xs:element>
```

This is a very simple example, but it's the kind of thing schema developers have to do over and over again. There is often built-in support in XML editors, including Oxygen, for schema syntax, but those provide for a wider range of options than NIEM allows. These NIEM-specific snippets provide an easy way for developers to quickly stub out the exact syntax the NDR requires and focus on filling in the semantics. It's easier, for example, to create a new augmentation when all you need to do is fill in the blanks rather than figure out how to get the syntax right as well.

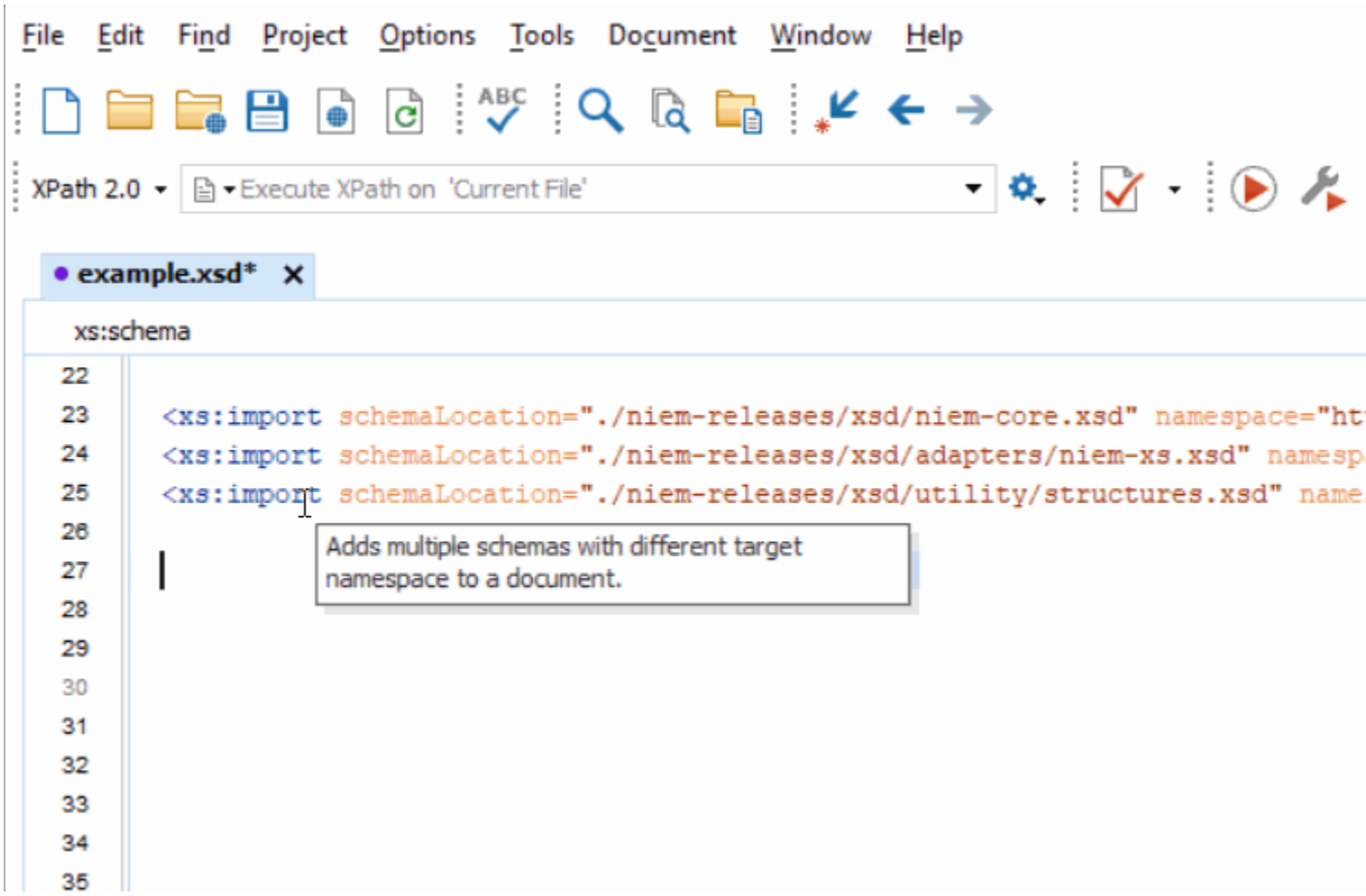
There are over two dozen snippets provided, ranging from simple snippets for an individual concept, snippets that stub out an entire namespace, and snippets that provide templates for related concepts at the same time.

Snippets are also available on this site for users who do not have the Oxygen XML Editor. Go to the Reference / Concepts section, navigate to the appropriate topic, and copy/paste the relevant snippets from the XML or JSON sections to your editor of choice. For example, a snippet to create a new element is available here.

Demo

Load NIEM snippets into Oxygen in order to easily insert common templates into NIEM domain or IEPD extension schemas:

- Load NIEM snippets (see instructions below)
- Type `niem` into an XML schema file
- Hit Ctrl + Space (Windows) or Cmd + Space (Mac) to see a list of applicable snippets with names that begin with "niem"
- Find and select the appropriate snippet
- Fill in the empty or partially-filled fields; update default values as necessary



Available NIEM snippets

The snippets include some commonly-used default values. If the given defaults are wrong for

your typical scenario, you can customize them in Oxygen. Some are tailored more for IEPD developers than domain modelers, especially those relating to default cardinalities. See the instructions on how to modify a snippet.

Property snippets

The following snippets provide template for various kinds of NIEM properties.

Label	Description	Example pattern	Info
niem-elt	NIEM element	ag:AgricultureProductionPlan	Info
niem-elt-abstract	NIEM abstract element	st:RoadwayGradeAbstract	Info
niem-elt-augPoint	NIEM augmentation point element (abstract)	ip:SectorAugmentationPoint	Info
niem-elt-substitution	NIEM substitutable element	st:RoadwayGradeCode	Info
niem-attribute	NIEM attribute	mo:codespaceID	Info

Type snippets

The following snippets provide templates for various kinds of NIEM types.

Label	Description	Example pattern	Info
niem-type-object	NIEM object type (has sub-properties)	ip:SectorType	Info
niem-type-csc	NIEM complex type with simple content (has a value and attributes)	st:MedianCategoryCodeType	Info
niem-type-simple	NIEM simple type (has a value)	mo:MGRSCoordinateStringSimpleType	Info
niem-type-enum	NIEM simple type with enumerations (has a value from a given set of codes)	st:MedianCategoryCodeSimpleType	Info
niem-type-list	NIEM simple list type (has a space-separated list of values)	nc:IntegerListSimpleType	Info
niem-type-union	NIEM simple union type (combines codes from multiple types)	biom:FrictionRidgePositionCodeSimpleType	Info
niem-type-adapter	NIEM adapter type (wraps an element from an external standard)	mo:WGS84EllipseType	Info

Combo snippets

The following snippets provide templates for two or more related NIEM components that are often built at the same time.

Label	Description	Example pattern	Info
niem-assn-2	NIEM association type NIEM association element	mo:AircraftTargetAssociationType mo:AircraftAssignedTargetAssociation	Info
niem-assn-3	NIEM association type NIEM association element NIEM association augmentation point	mo:AircraftTargetAssociationType mo:AircraftAssignedTargetAssociation mo:AircraftTargetAssociationAugmentationPoint	Info
niem-aug-2	NIEM augmentation type NIEM augmentation element	it:PersonAugmentationType it:PersonAugmentation	Info
niem-code-2	NIEM code simple type NIEM code CSC type	st:MedianCategoryCodeSimpleType st:MedianCategoryCodeType	Info
niem-code-3	NIEM code simple type NIEM code CSC type NIEM code element	st:MedianCategoryCodeSimpleType st:MedianCategoryCodeType st:MedianCategoryCode	Info
niem-code-4	NIEM code simple type NIEM code CSC type NIEM abstract element NIEM substitutable code element	mo:SignalDirectionCodeSimpleType mo:SignalDirectionCodeType mo:SignalDirectionAbstract mo:SignalDirectionCode	Info
niem-metadata-2	NIEM metadata type NIEM metadata element	cui:DocumentMarkingMetadataType cui:DocumentMarkingMetadata	Info
niem-type-object-2	NIEM augmentation point element NIEM object type	ip:SectorAugmentationPoint ip:SectorType	Info

Schema snippets

The following snippets can be used to initialize a NIEM XML schema and includes a general template for the schema header, definition, local terminology section, and import statements.

Label	Description	Example pattern	Info
niem-schema-4-ref	NIEM 4.0-style reference schema template (for a release schema)	Schema header for the NIEM 4.0 Intelligence domain schema	Info
niem-schema-4-ext	NIEM 4.0-style extension schema template (for an IEPD schema)		Info
niem-schema-5-ref	NIEM 5.0-style reference schema template (for a release schema)	Schema header for the NIEM 5.0 Intelligence domain schema	Info
niem-schema-5-ext	NIEM 5.0-style extension schema template (for an IEPD schema)		Info

Other snippets

The following snippets provide additional NIEM templates.

Label	Description	Example pattern	Info
niem-enum	NIEM enumeration	Code "APPROVED"	Info

Label	Description	Example pattern	Info
niem-import	NIEM import statement for a conformant namespace	Import NIEM Core	Info
niem-import-external	NIEM import statement for an external standard	Import GML (not a NIEM-conformant namespace)	Info
niem-term-literal	NIEM local terminology described by a literal string	Local term "UTM" with literal	Info
niem-term-def	NIEM local terminology described by a definition	Local term "Waypoint" with definition	Info

Conventions

Several conventions were used with the snippets.

Note that these too can be adjusted by modifying the snippets in Oxygen in case you would like the defaults set differently.

Blanks

Blanks are used for values where no helpful defaults can be provided.

Dots (…)

A set of '…' characters were used in places where a field has been partially filled in but requires additional text. Otherwise, it might not be immediately obvious that something is missing.

For example, it may not be obvious that the following element name and type reference are incomplete:

```
<xs:element name="Code" type="CodeType" nillable="true">
  <xs:annotation>
    <xs:documentation></xs:documentation>
  </xs:annotation>
</xs:element>
```

Adding dots to indicate missing text helps highlight where updates need to be made. The dots also have the benefit of being invalid characters in a property or type name. Although still valid within definitions, Oxygen can at least easily show these as errors when they are left remaining in component names and references:

```
<xs:element name="...Code" type="...CodeType" nillable="true">
  <xs:annotation>
    <xs:documentation></xs:documentation>
  </xs:annotation>
</xs:element>
```

> Exception: Oxygen snippets allows you to designate where the cursor should appear. In cases where an incomplete value appears at the same place the cursor will be set, no dots have been

added. The cursor itself is the cue to fill in the rest of the information. Dots are used to indicate incomplete information in the fields that follow.

ALLCAPS

Labels written as ALLCAPS were used in places where Oxygen can be used to autofill the value if it has already been defined.

For example, the snippets for the `niem-code-2` pattern provide templates for a code simple type and a code complex type with simple content (CSC). If you fill in the information for the simple type first as it appears, Oxygen will let you autofill in the name of the simple type you just created as you fill in the CSC's base type. Autofill is quicker than editing an alternate default value set as `...CodeSimpleType`.

In order to use autofill, you first need to select the current value in the field. Once fully highlighted, hit Ctrl/Cmd + Space for a list of available components and begin typing to get to the one you need.

> A value without delimiters like 'ALLCAPS' can be selected in Oxygen from the keyboard in one Ctrl/Cmd-Shift + arrow key jump. Selecting `...CodeSimpleType` would take Oxygen six jumps to fully highlight, one arrow key stroke for each dot and upper case letter. This is one of the reasons this ALLCAPS formatting convention has been selected.

Ordering

Component snippets are ordered by dependencies. The order for components in the `niem-code-4` snippet is

1. Code simple type
2. Code CSC type (base type is defined in # 1)
3. Abstract element
4. Code element (type is defined in # 2; substitution group is defined in # 3)

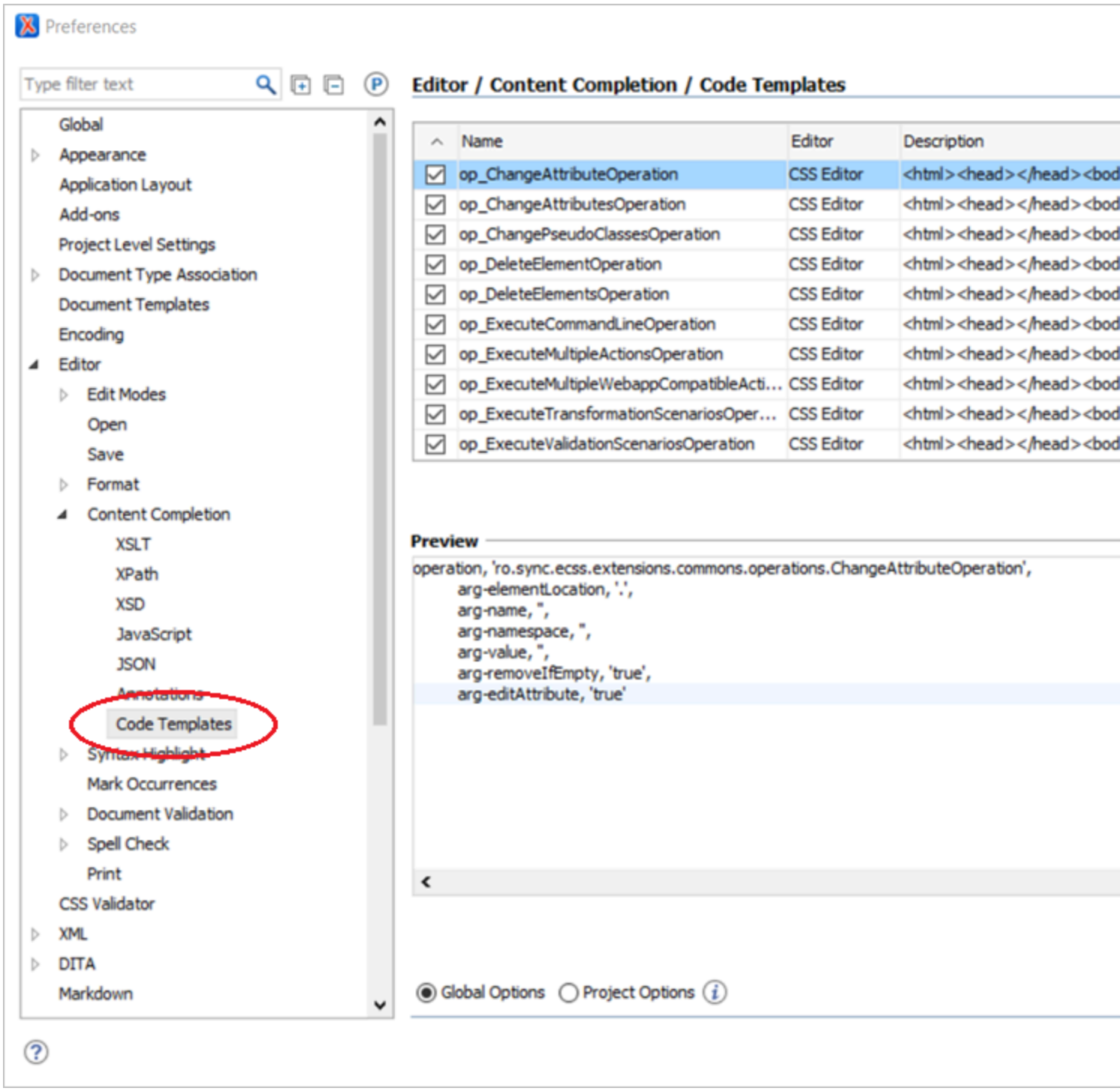
This ordering prevents encountering required fields that have not yet been defined.

Using NIEM snippets with Oxygen

For more information on snippets in Oxygen, please see their page about Code Templates.

Import snippets

- Download the **NIEM Oxygen snippets file**
- In Oxygen, click on menu item **Options** -> **Preferences…**
- In the left panel sidebar, click on **Editor** -> **Content Completion** -> **Code Templates**
- Click the Import button near the bottom right of the panel and select the downloaded snippets file

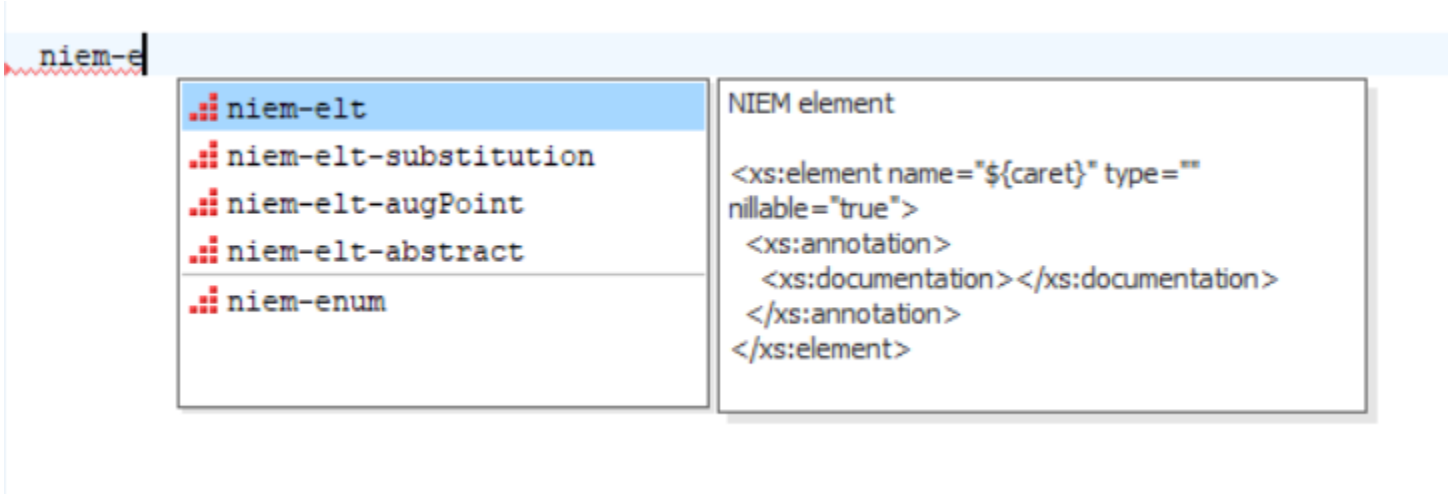


Use a snippet

All NIEM snippets begin with `niem-`

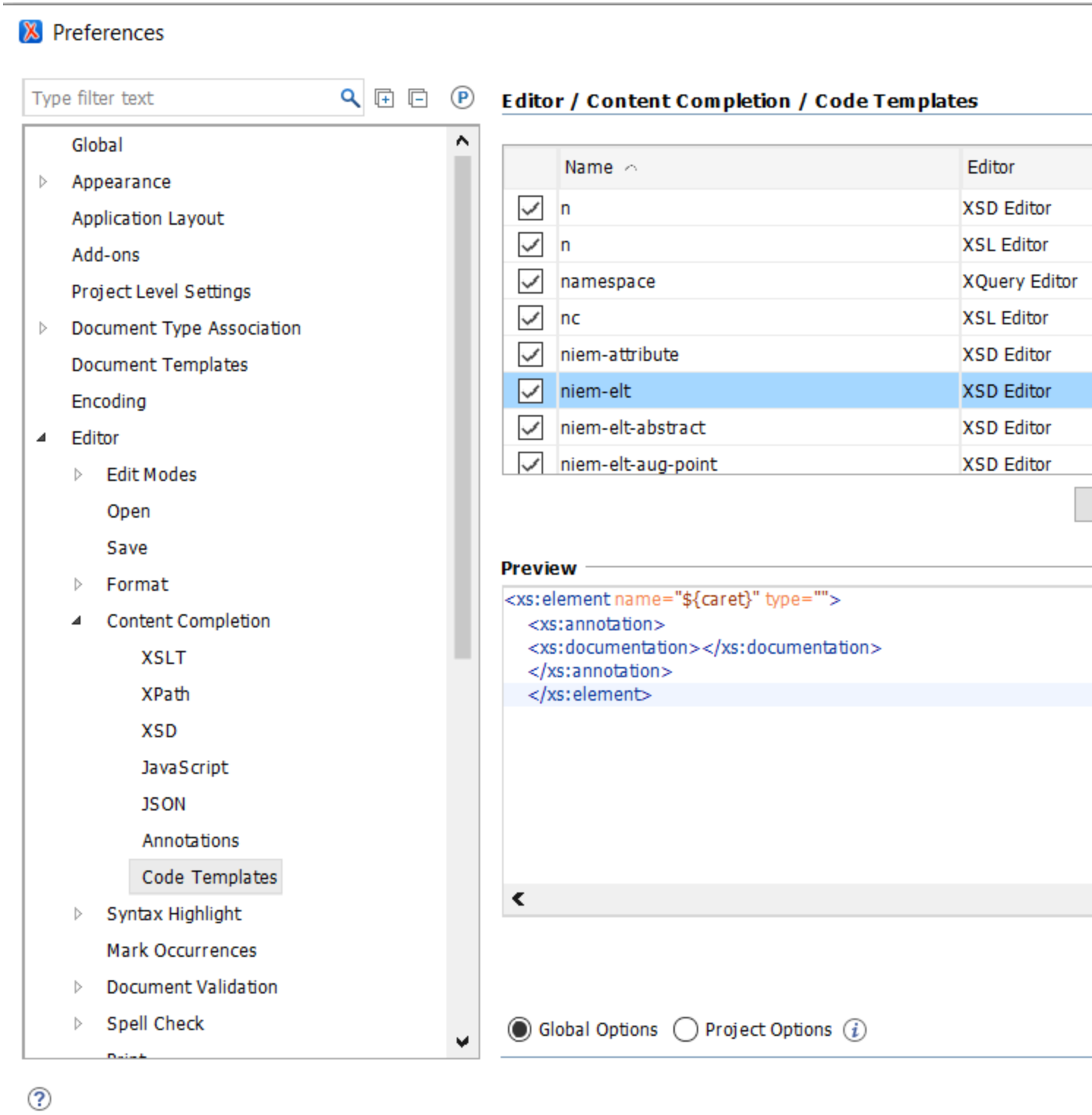
- Type “niem” in an XML schema file
- Hit Ctrl/Cmd + Space
- Scroll through the drop-down list to see all snippets that begin with “niem”
- Hit “Tab” or “Enter” to load the snippet

- Fill in fields like names and definitions; adjust default values as needed

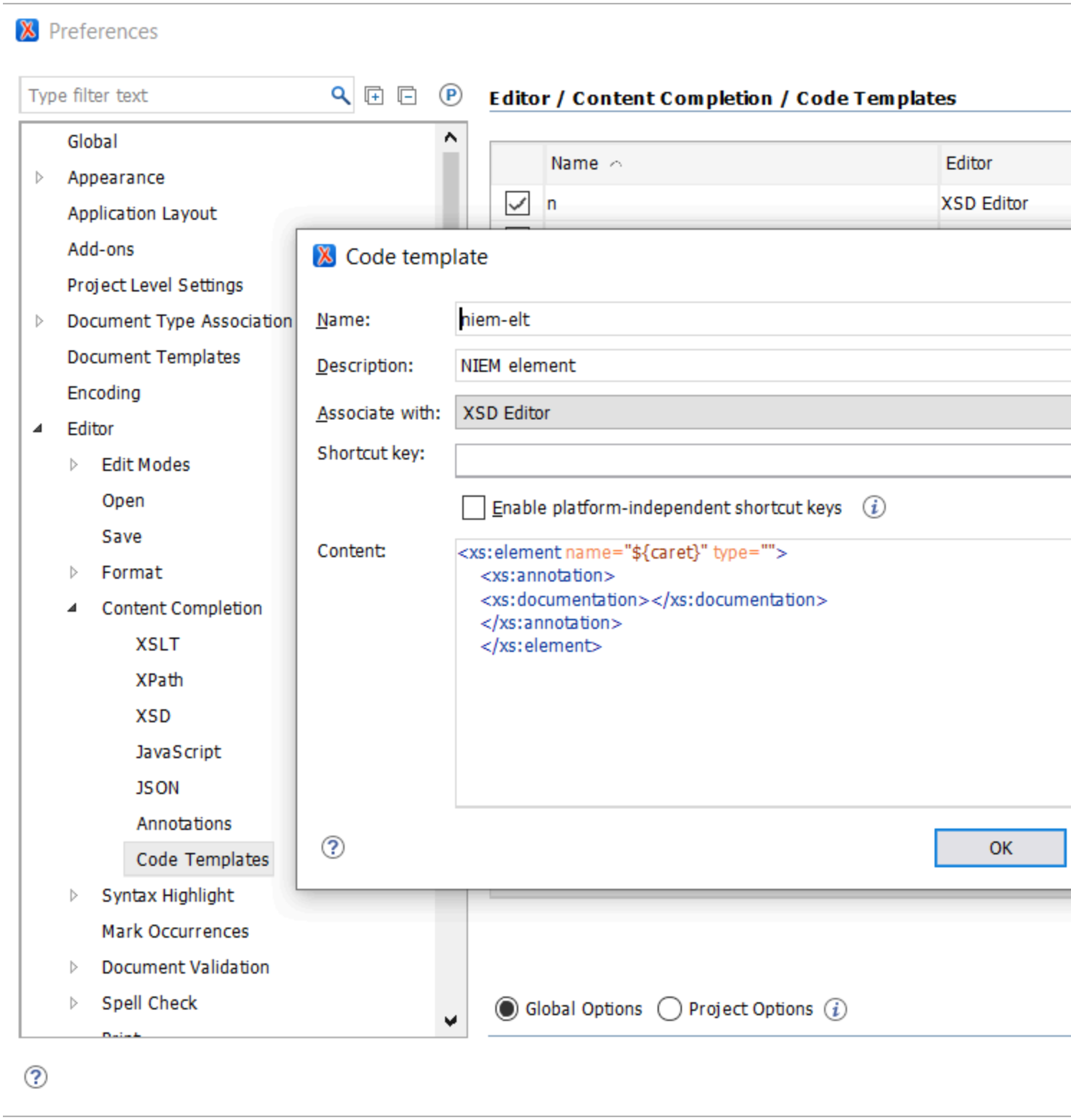


Modify a snippet

- In Oxygen, click on **Options** -> **Preferences**
- In the left panel sidebar, click **Editor** -> **Content Completion** -> **Code Templates**
- Scroll down and select the snippet to modify
    - The snippet will begin with “niem-“
    - NIEM snippets will appear under “XSD Editor” and/or “JSON Editor” group
- Click the “Edit” button



- Modify the snippet name, description, code, or add a keyboard shortcut.
- Notes:
  - The `${caret}` variable marks the position where the cursor will appear once the snippet is loaded.



Reload snippets

- To load an updated snippets file, simply go through the original import steps again.
- Snippets with the same name and editor group will be overwritten.
- Individual snippets can be deleted or snippets can be reset by clicking the **Restore Defaults**

button.

Using XML Catalogs with Oxygen

Use XML catalog files in Oxygen to specify or override schema file locations.

- XML Catalog Preferences
- Document Type Association
- More

Oxygen has several different ways to add a catalog file.

XML Catalog Preferences

To add a global user-defined catalog in Oxygen via the XML Catalog preferences page:

- Go to Options > Preferences > XML > XML Catalog
- Click the “Add” button to add the new catalog
- Restart Oxygen XML Editor

Preferences

Type filter text

- Global
- Appearance
- Application Layout
- Add-ons
- Document Type Association
  - Locations
- Encoding
- Editor
- CSS Validator
- XML
  - **XML Catalog [P]**
  - XML Parser
  - Sample XML Files Generator
  - XProc
  - XSLT-FO-XQuery
  - Ant
  - Import
  - XML Signing Certificates
  - XML Refactoring
- DITA
- Data Sources
- SVN
- Diff
- Archive
- Plugins
- External Tools
- Menu Shortcut Keys
- File Types
- Open/Find Resource
- Custom Editor Variables
- Network Connection Settings
- XML Structure Outline
- Views

XML / XML Catalog

Prefer: ○ system ◉ public

Verbosity: ◉ None ○ Unresolved entities ○ All messages

☑ Resolve schema locations also through system mappings

☐ Process "schemaLocation" namespaces through URI mappings fo

☑ Use default catalog

C:\Program Files\Oxygen XML Editor 20\frameworks\catalog.xml

Catalogs
file:/C:/git/Developer-Network-IEPD-Series--SuperHero/base_iepd/x

Add

Additional catalogs are added from each document type.

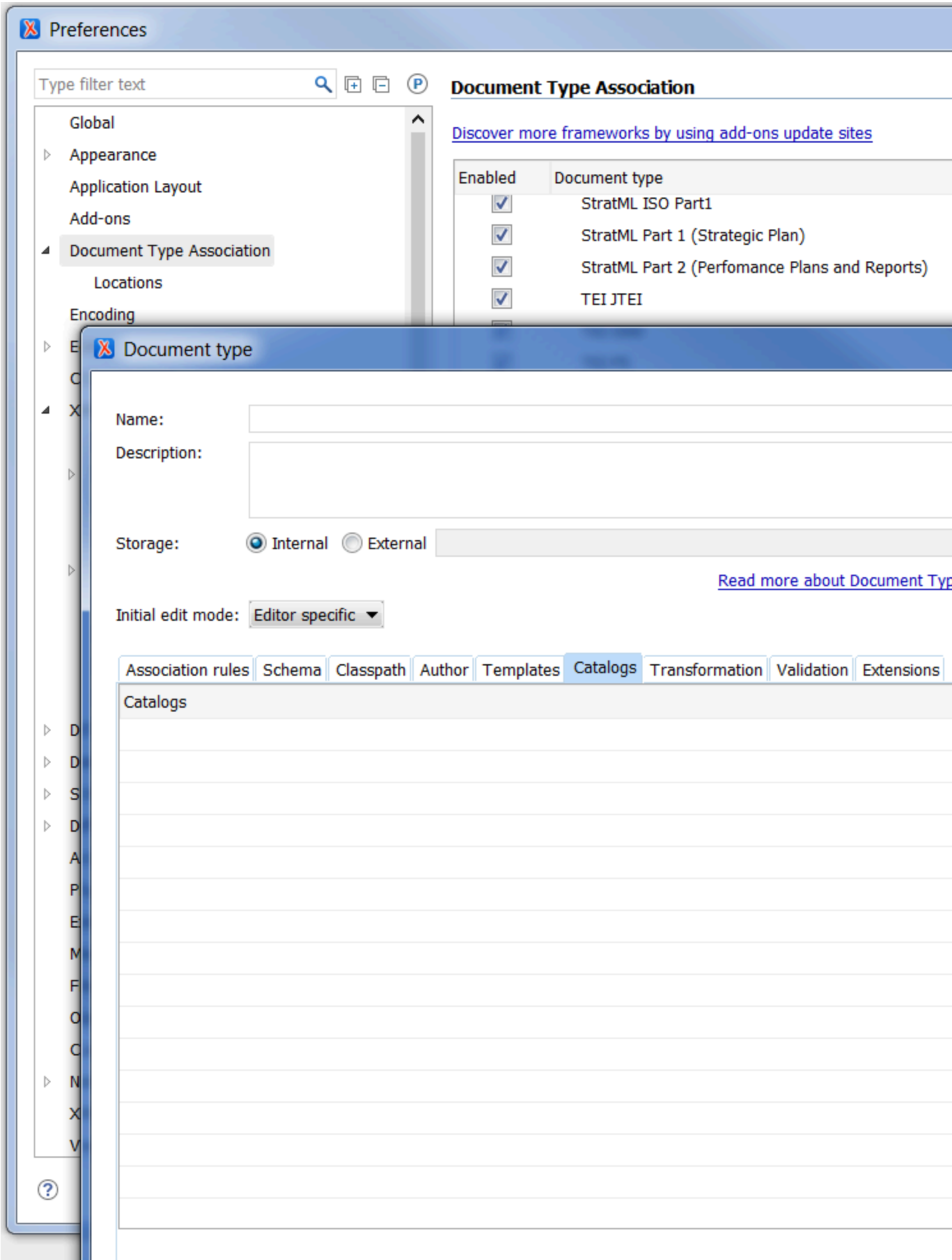
Changes in the catalogs list or the files referred may require

○ Global Options ◉ Project Options

Document Type Association

To add a user-defined catalog in Oxygen via a document type association:

- Go to Options > Preferences > Document Type Association
- Add a new Document Type for the XML file(s)
- Add the catalog in the Catalogs tab
- Add an Association Rule


Preferences
Type filter text
Global
Appearance
Application Layout
Add-ons
Document Type Association
Locations
Encoding
Document Type Association
Discover more frameworks by using add-ons update sites
Enabled
Document type
StratML ISO Part1
StratML Part 1 (Strategic Plan)
StratML Part 2 (Perfomance Plans and Reports)
TEI JTEI
Document type
Name:
Description:
Storage:
Internal
External
Read more about Document Typ
Initial edit mode:
Editor specific
Association rules
Schema
Classpath
Author
Templates
Catalogs
Transformation
Validation
Extensions
Catalogs

More

For more information, please see Oxygen's catalog help files:

- Working with XML Catalogs
- XML Catalog Preferences
- Document Type Configuration Dialog Box
- Configuring XML Catalogs via a Document Type Association

NIEM resources for XMLSpy

XMLSpy is a proprietary XML editor and integrated development environment developed by Altova. NIEM provides some additional information and resources for this tool that may assist with domain and IEPD schema development and conformance testing.

- Using XML Catalogs with XMLSPy

  Use XML catalog files in XMLSpy to specify or override schema file locations.

Using XML Catalogs with XMLSPy

Use XML catalog files in XMLSpy to specify or override schema file locations.

Add custom file associations to the Altova `CustomCatalog.xml` file. Add one or more `nextCatalog` elements in order for XMLSpy to use user-created catalog files.

Steps:

- Go to File > Open.
- Browse to the Altova/XMLSpy#### data directory
- Open `CustomCatalog.xml` (see below)
- Add a `nextCatalog` element that specifies the location of your catalog file (example below)
- Restart XMLSpy

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog
  xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:oasis:names:tc:entity:xmlns:xml:catalog Catalog.xsd'

  <nextCatalog catalog="https://example.com/xml-catalog.xml"</u>/>

</catalog>
```

For more information, please see Altova's Catalogs in XMLSpy.

NIEM Domain Modeling Guide

Version: September 29, 2016

- Prologue
- Technical Basics
  - HLVA
  - Namespaces and versions
  - NIEM release cycle
  - Domain updates
  - Conformance and QA
- Model data components
- Search NIEM
  - Schema Subset Generation Tool (SSGT)
  - Movement
  - NIEM Model Spreadsheet
  - Other methods to search NIEM content
- Mapping to NIEM
  - How to map to NIEM
  - Map construction
  - Mapping strategy
  - Handling partial matches
- Evaluate component quality
- Appendix A - Governance
  - Regular meetings
  - Email lists
  - Questions and feedback
- Appendix B - Resources
- Appendix C - NIEM glossary

Prologue

Purpose of NIEM

The National Information Exchange Model (NIEM) is a joint partnership across all levels of government (federal, state, tribal, and local) as well as private industry. It is a uniform, harmonized, well-defined set of metadata structures normatively represented in the W3C XML Schema Definition (XSD) Language. We refer to the structures in this set as *NIEM data components*. They are openly available for use (and reuse) for designing, developing, and implementing standard XML-based information exchange specifications (IEPDs).

The primary purpose for the NIEM is to exchange information and data at the message level across different communities of interest. This does not limit its aptness for sharing information within a single community. However, it does drive its foundational design principles and rules. To achieve consensus for a large common vocabulary, with well-defined semantics, and consistent structure that enables numerous communities of interest to share information at the message level, requires a careful balance of rational constraints, flexibility, compromise, and collaboration. NIEM is designed to develop,

distribute, and support large-scale enterprise-wide information exchange standards and associated procedures. In turn, this enables a variety of jurisdictions to automate information sharing during both emergencies and daily routine operations. Meeting this goal requires that all participating communities of interest model and maintain their NIEM data components according to NIEM specifications, even when to doing so may compromise long standing traditions within a community (for example, simple data names, terse definitions, or complex data structures). This is necessary (1) to ensure different communities can share message level information and its associated semantics accurately, and (2) to facilitate model consensus and continued scalability.

NIEM Domains

A NIEM *domain* is a Community of Interest (CoI) aligned around a common interest. For NIEM the common interest is generally based on a Line of Business (LoB) (or related group of such) for which information must be shared or exchanged.

A NIEM *domain steward* is responsible to ensure a given domain model (namespace or namespaces) is governed and maintained under the NIEM specifications. To do this effectively, the domain steward should commission team of subject matter experts with technical skills that collectively represent various levels of government for that CoI or LoB. However, the reality is that a domain is sometimes sponsored and governed by one organization because it may be difficult to identify volunteers from various levels of government within the CoI or LoB who can or will participate. These are usually the smaller domains with less content to govern.

Domains are a key aspect of the NIEM architecture for several reasons. NIEM was designed with one set of common Core data components governed by the NIEM Business Architecture Committee (NBAC), and multiple sets of domain data components (derived from the Core) each semi-independently governed by a domain technical team. By design, NIEM partitioning corresponds directly to its governance, and in turn, this facilitates both its version architecture and its ability to scale up (by onboarding new domains). Furthermore, almost every NBAC member is also a member of at least one domain team which provides significant cross-pollination for collaborating on issue resolution, harmonization, refactorization, refinement of data components during NIEM release development cycles. `:w

Finally, the NIEM architecture is flexible enough to allow the establishment of domains at-large that intentionally do not participate in governance, and yet, can still use NIEM to share information with member domains or other domains at large.

What this guide covers

- Technical aspects of domain model management.
- How to start a new domain model.
- How to evaluate the quality of NIEM data components.
- Data component naming and definitions.
- Creating NIEM code lists quickly.
- Technical responsibilities of a domain:
    - Structure its domain content so other domains can reuse it.
    - Design its data components so it can be understood by other domains.
    - Participate in harmonization and issue resolution when its domain is involved.
    - Consider impacts that changes to its domain will have on other domains (or Core).

    - Maintain representation in NBAC and participate.
- What impacts my domain model? What does my domain model impact?
- Maintaining a domain model (what to model, when to change it, etc.)
- Harmonization
- Participating in a release cycle, domain update, and other NIEM technical processes.
- Technical assistance, tools, templates, best practices, references, etc.

This guide does not necessarily detail all topics it discusses. However, the guide often references other NIEM documents or Web pages for more information. It also includes an extensive appendix of references to other NIEM resources.

What you should be familiar with to use this guide

To use this guide, you should be somewhat familiar with the following topics and documents:

- Extensible Markup Language (XML) 1.0 (5th Edition)
- W3C XML Schema Definition (XSD) Language
    - Part 1: Structures
    - Part 2: Datatypes
- General NIEM concepts; for example, as presented in:
    - NIEM Introduction
    - NIEM Concept of Operations
    - NIEM Training
    - NIEM IEPD Starter Kit
- NIEM Conformance
- NIEM High Level Version Architecture (HLVA)
- NIEM Naming and Design Rules (NDR)

Audience

This document is intended for use by technical personnel who support NIEM domain modeling and harmonization during major and minor release cycles, including domain updates. In particular, we have tried to make it most useful to new or emerging domains that are building their initial content. Guidance in this document is useful to NIEM working level governance (NBAC, NTAC, PMO staff, and lead developer) who often evaluate the need to harmonize, refine, or refactor NIEM domain models. It may also be useful to organizations or groups interested in using NIEM but that do not intend to join its governance.

Technical Basics

HLVA

The NIEM High-Level Version Architecture (HLVA) 3.0 identifies processes, artifacts, governance responsibilities and how they interact to produce new releases of and updates to the NIEM model. The HLVA discusses issues with inherently conflicting tradeoffs including (but not limited to):

- Timeliness vs. stability
- Domain autonomy vs. interoperability

- Ease for information exchange developers vs. ease for domain developers

Under PMO direction and with technical assistance from the lead developer, the NBAC, NTAC, and domains execute NIEM governance at the operational level. Practicing the HLVA has demonstrated the following results:

- There will be reliable and consistent schedules for activities that produce a NIEM major or minor release ensuring that activity participants know the work timelines and when their input is due, and the NIEM users can plan for release dates.
- Between regular major and minor release cycles, any domain may independently publish updates on its own timeline and make them available for immediate use (with the knowledge that their changes may impact other domains that extend or reuse their data components).
- Domain updates are incorporated into the next NIEM release. So, periodic releases remove negative impacts, inconsistencies, or incoherence resulting from independent domain updates.
- Information exchange developers are provided with a NIEM release, which is an updated schema document set that is coherent, for increased usability.
- A specific, concrete path exists for domains to feed input into NBAC's update, core synchronization, and harmonization processes, for inclusion in a future NIEM release. Domain, NBAC, and NTAC work on new content and changes is more evenly spread over time periods between release cycles instead of intensifying them.
- All changes are visible. Each namespace URI is used for exactly one version of a schema document, and any changes to a schema result in a new namespace URI.
- Change logs support descriptions of changes made to each namespace.

- Domain independence
- Release optimization
- Types of reference content (major, minor, micro, domain update, core supplement)
- Repositories (release, publication)
- Collaboration

Namespaces and versions

- Important to domains who submit content in XSD.
- Describe the correlation of target namespaces and XML schema documents.
- Each schema document is assigned a unique target namespace.
- Namespace Syntax.
- URIs (Uniform Resource Identifiers), syntax, purpose, etc.

NIEM release cycle

- Release types: major, minor, micro, core supplement
- Scheduling: standard timeframes, milestones, reviews, stages (pre-alpha, alpha, beta, release candidate)
- Harmonization (who, what, how, the process, concepts, etc.)
- Issue resolution (who, what, how, the process, voting, tracking, etc.)
- Input formats
    - XSD (NIEM conformant XML Schema documents)
    - XLS (Microsoft Excel spreadsheet) Change Request (Christina's latest modifications)

    - Simple text explanations in email for very minor changes.
- Technical assistance (who, what, how)

Domain updates

- Domain independence and self-serviced
- Types of domain updates: incremental vs. replacement
- Multi-domain coordinated domain update
- Change content preparation and model management
- Metadata, lineage, change log

Conformance and QA

- General and specific conformance
- Verification (who, what, how, when, etc.)

Model data components

How to name, define, and structure data components:

Identify data requirements

NIEM was designed to share information across domains (communities of interest or lines of business). You can certainly consider both internal and external data requirements, but you definitely should identify data requirements for sharing information with communities and organizations outside your own. Ask yourself who you share your information with outside your domain? Also, who from outside your domain shares information that you need?

It helps to identify or develop simple scenarios, and within those scenarios identify common use cases for sharing information. It also helps to examine existing database schemes, data dictionaries, XML schemas, flat files, paper/electronic forms, workflows, etc. for data requirements. Such data sources can provide insights into what data is currently shared and how.

There are likely many variances in data names and definitions that already exist in the sources. To create a good domain model it is necessary to harmonize, i.e., decide on a single name, definition, and structure (type) for each data element; eliminate duplication. Then map the data model elements (and types) back to their authoritative sources (data dictionaries, database schemes, forms, etc.) and record this mapping for reference. This mapping will likely become an critical resource to programmers who will implement information exchanges with the domain model and may have to trace back to the legacy data sources.

Do not *boil the ocean*

Model data components for real information requirements that are known to exist or that you know are necessary and are based on actual information sharing scenarios or use cases.

Do not create NIEM data components for every possible contingency or likelihood. Do not create components that might be nice-to-have or that are “likely” future requirements. That said, this does not mean you shouldn’t model new data requirements that are definitely in near term plans.

Note that it is important to consider real information exchange scenarios and associated use cases that will identify both the existing and new near term requirements. If possible, envision what the domain should look like in the (not too distant) future and build the to-be model from the as-is baseline.

Start small scale

Scale back the effort rather than create data components that may have to be deleted or changed in later release cycles, and will subsequently confuse your domain community. Try to ensure the key domain classes (object types) are present in the model, but it is not necessary to be absolutely complete. It is easy to add properties to a type later (in a release or domain update). Furthermore, NIEM types are easily extended with additional elements in IEPDs using augmentation points. IEPD extensions confirm the need for new requirements, and feed them back to the reference model for future additions to NIEM.

Grow your domain model over time as you build NIEM experience. NIEM allows you to publish a domain update anytime outside of the annual release cycle. So, there is always time to catch up and never a need to rush for NIEM's sake.

Do not overbuild data components. Keep them simple: A type represents a real world object or concept. Elements describe the characteristics or parts of that object or concept. A complex type has elements, elements are typed, and so on down to primitive simple elements (of type string, text, name, date, amount, or token, etc.). For example:

- element Kk, complexType Xx has
    - element Aa, complexType (with simple content) string
    - element Bb, complexType (with simply content) date
    - element Cc, complexType (with simply content) amount
    - element Dd, complexType Yy has
        - element Ee, complexType (with simply content) name
        - element Ff, complexType (with simply content) text
        - element Gg, complexType Zz has
            - element Hh, complexType (with simply content) string
            - element Ii, complexType (with simply content) percent
            - element Jj, complexType (with simply content) text

All NIEM elements are defined as complexTypes that are extensions or ancestors of a complex base type in `structures.xsd`. These base types contain a simple object attribute group that supports NIEM built-in capabilities such as metadata and referencing.

Code lists also contain the simple object attribute group; however, code lists get that attribute group somewhat differently from elements. Code lists require both a complex type and a simple type. Each code element is defined by an associated `CodeType` (for example, `EyeColorCode` is defined by `EyeColorCodeType`). This `CodeType` was derived from a `CodeSimpleType` (following the example, `EyeColorCodeSimpleType`). The `CodeSimpleType` contains the code values as XML enumerations, while the `CodeType` extends the `CodeSimpleType` by adding the simple object attribute group, and the `CodeType` becomes an XML complexType with simple content.

Map data requirements to NIEM

- Reuse existing data components where possible.

- Identify missing requirements (gap analysis).
- Create new NIEM data components ONLY when needed to cover gaps.
- Consistent with the NIEM NDR and releases (reference schemas).
- Use elements for object properties; elements are far more flexible.
- Use attributes only if absolutely necessary; e.g., very tightly coupled metadata on an element, as attribute `xml:lang` is to an element of type `TextType`.

Clearly identify data components that will require future harmonization

with the Core (or with other domains if inputting a domain update).

Use reference materials to model various NIEM techniques

- Simple examples exist in the Reference Concepts section that illustrate NIEM techniques such as: augmentation, association, role, references, metadata, abstract elements and substitution groups, type extension, type adapter, etc.
- Review existing release reference schemas.
- The change request contains examples.
- Use the reference tools, e.g.,Schema Subset Generator, ConTesA, Code List Schema Generator, Migration Assistant, etc.

During release cycle, provide initial input in NIEM-conformant

XML Schema document (XSD) or Change Request (XLS) format.

Domain data models vs. IEPDs

- Domains contain the reusable building blocks for creating IEPDs; IEPDs are definitions for information exchanges.
- An IEP (Information Exchange Package) is an instance XML document that conforms to an IEPD.
- Differences in purpose, scope, reusability, NDR rules.
- IEPD extensions are candidates for domain content, but each should be evaluated and potentially remodeled.
- Use IEPD repositories to share IEPDs.
- Example IEPD: Cursor-on-Target MilOps Future Namespace.
- IEPD versioning
- When building a domain model, concentrate on designing XML data components that can be reused within any IEPD that any domain might build to share data with you. Do NOT build an IEPD for a domain model. To do this, try to design small parts that can be reused independently and are potentially standalone. Do not build a couple of very large monolithic structures. Think in terms of the kinds of small objects you need to represent, so they can be reused in multiple IEPDs.

The NIEM Lead Developer (GTRI) can and will help you

- Assists with conformance, quality, and modeling.
- Integrates inputs into the release or DU for your review.
- Can save you extra work and time if you make systemic errors.

- Get preliminary inputs in early for review, feedback, and assistance.

NIEM Data Component Names and Definitions

A scalable vocabulary that will be used by many different communities to exchange information must be understandable to all parties involved. To facilitate consistency and understanding NIEM has established rules for naming and defining its data components. These rules apply to all types, elements, and attributes. They were derived from ISO/IEC Standard 11179, Information Technology – Metadata Registries (MDR). This standard has been around since the 1990's and continues to be updated. For this reason, please do not expect that NIEM rules for names and definitions are exactly synchronous with Standard 11179. Yet, for the most part, the NIEM Naming and Design Rules (NDR) still generally follow 11179 rules and guidance for designing metadata definitions and names.

How to Draft Data Component Definitions

Each NIEM element, attribute, and type must be clearly defined before it will be accepted for a NIEM domain update or release. ISO/IEC Standard 11179 Part 4 is the guidance upon which NIEM definitions are formulated. The salient points of that guidance and the NIEM NDR rules are repeated here:

- Each data component definition must be unique from all others and distinguishable in meaning. No two definitions can be identical in wording or so close in meaning that they could refer to the same data component.

- Try to keep definitions simple and straight forward. This is not always possible, at least make them understandable to others who are not a part of your community of interest (i.e., domain).

- Element definitions almost always begin with an indefinite article (i.e., "a" or "an"), never a definite article (i.e., "the").

- Since it is often the case that a type and an element of that type can be defined with identical or similar words (for example, Person and PersonType), it is a NIEM best practice to begin a type definition with the phrase "A data type for …" This ensures that the definition for the element and its associated type are easily distinguishable.

- Best practices for the opening phrase of the definitions of other kinds of NIEM data components are:
    - abstract elements: "A data concept for …"
    - association elements: "An association for/between/among …"
    - augmentation elements: "An augmentation point for …"
    - role elements: "A role of a(n) …"
    - types: "A data type for …"

- If you have trouble designing a good definition for a data component, refer to the current NIEM release for examples.

- Avoid using the terms in the name of a data component to define it. That said, a good data component name may be self-defining. If there are no good synonyms to employee in the definition, and you must use one or more terms from the name, it is not an error.

- Aside from the typing implications of opening phrases mentioned above, do not put data typing information in a data component definition. An example of a bad definition is: `SocialSecurityID` - "A 9-digit number with hyphens that identifies a person in the U.S." The fact this element is a "9-digit number with hyphens" should not be included in its definition. In most cases separators are meaningless and unnecessary visual aids for human readability. The correct way to define this identifier is to use the `xs:pattern` attribute to restrict it to nine digits and without hyphens. Note that non-alphanumeric characters may be used within identifiers if they are an integral part of the identifier itself (for example, passwords), and are NOT merely being used as visual separators.

- Based on the foregoing, it is good practice to avoid use of the word "type" within definitions, because in most cases,"type" refers to data typing. Instead, in the appropriate cases, a definition should use terms such as kinds, class, category, nature, genre, or form to refer to classifications (another relatively common word-sense of "type").

- Code list type definitions — A code list must have a definition for both its associated `CodeType` and `CodeSimpleType`. Both of these data types can have the same definition (one of the few exceptions to the unique definition rule) since they are semantically the same type. The difference is that the complex type extends the simple type to add several common properties that are part of the NIEM infrastructure. The definition should NOT refer to the code values or the code literals. For example, the definition for `DayOfWeekCodeSimpleType` could be "A code for a day in a week". It should NOT include Su=Sunday, Mo=Monday, etc. These values and associated literals will be recorded in the `xs:enumeration` elements within the XSD for the `CodeSimpleType`.

- Type definitions should describe what a type is, not list and define its contents. Describe it as an object not a container of attributes. For example, `VehicleType`:

    - `VehicleType` — (bad definition) A data type that contains the following (properties or characteristics) VehicleColorInteriorText, VehicleDoorQuantity, VehicleIdentification, VehicleMake, VehicleModel, …
    - `VehicleType` — (good definition) A data type for a means of ground transportation designed to carry an operator, passengers, and/or cargo.
- An element definition should describe what the element is, not reference its data type or generic necessary data elements or related information. The following are all examples of bad element definitions for the element named `Neoskizzle`:
    - A `Neoskizzle` container containing many references to elements as defined in `NeoskizzleType`.
    - A `Neoskizzle` contains identification information as well as many other characteristics. See type definition.
    - A `Neoskizzle` containing necessary data elements.
    - `Neoskizzle` related information.
    - A person who neoskizzles. — This definition still uses the element name (or a derivative) to define itself. This may be fine if the word is very common. In this case, that does not work.

At this point, do you have any idea what "neoskizzle" means or is by reading any one of the definitions above? Of course not. So, here is an example of a good definition: A person who takes part

in an event, activity, meeting, or other social function. Apparently, a synonym for “neoskizzle” is “participant”. So, why not use the term “Participant” for the element name? (By the way, in case you hadn’t guessed, there is no such word as “neoskizzle”. It’s made up.)

How to Design Data Component Names

The NIEM NDR provides fairly clear rules and guidance regarding the naming of data components. The most important NDR sections for understanding NIEM data naming are:

- Consistent Naming. This helps to establish clear meaning. NIEM also follows ISO/IEC Standard 11179 syntax for data names.
- ISO 11179 Part 5
- Naming Rules

Summary of naming syntax

The syntax of NIEM data names comes from IEC/ISO 11179 Part 5.

A single NIEM data name may consist of a number of terms. A term is a meaningful word, an abbreviation for a word, or an acronym. Word terms are one of:

1. The preferred U.S. spelling of words defined in the Oxford English Dictionary (The OED is a subscription based dictionary. A free version is available at Oxford Dictionaries).
2. `ID`, the authorized abbreviation for `Identifier`.
3. `URI`, the authorized abbreviation for `Uniform Resource Identifier`.
4. Local terminology, acronyms, or abbreviations that must be defined within schema documents per the NDR Section Use of Acronyms, Initialisms, Abbreviations, and Jargon.

In accordance with IEC/ISO 11179 Part 5, terms that make up a NIEM data name are classified into four basic parts according to their placement and function. A term in a data name can be one of these:

- Object term - (may not be present if the property is generic enough) describes the real world object or concept to which this data name applies.
- Property term - describes the characteristic or portion of the object to which this data name applies.
- Representation term - describes the NIEM data type of this data name.
- Qualifier term - (optional) describes more specific information applicable to the object, property, or representation term; multiple qualifier terms may be used.

Example: `VehicleTrafficControlDeviceCategoryCode` — A data type for a kind of traffic control device (TCD) applicable to this motor vehicle at the crash location.

- `Vehicle` = object term
- `Traffic`, `Control`, `Device` = all qualifier terms modifying `Category`
- `Category` = property term
- `Code` = representation term

Other naming guidance

- **Avoid use of the term `Type`** (except as a representation term to identify the name of a NIEM type). This is reserved for the representation term indicating data typing; instead use `Category`. Because it is an extremely common concept in all domains, the only current exception is `BloodType`.

- **Avoid use of the term `Number` in data names**. Its use is usually too generic to be meaningful or helpful, unless a name is extremely common across all domains (not just a few). For example, the data name `TelephoneNumber` is common across all domains; in this case, it is most useful for clear meaning. Otherwise, a number should usually be an ID (Identifier), Quantity, Numeric (or Value), Amount, Measure, or Duration (of time) (See: https://reference.niem.gov/niem/specification/naming-and-design-rules/3.0/niem-ndr-3.0.html#section_10.8.7).

- **Do NOT use double terms** (i.e., consecutive identical terms such TypeType or NameName) unless such a term has very specific meaning. Double terms should be replaced with a single instance of the term (as long as it does not detract from the real meaning).

- **ID (Identifier) vs. Identification** — `ID` and `Identification` elements are easy to confuse. An `ID` is a string element that uniquely identifies an entity; so, an `ID` has simple content. An `Identification` element is a set of sub-elements. For example, an `Identification` element for a person usually has sub-elements such as `PersonName`, `PersonHeight`, `PersonWeight`, `PersonEyeColor`, `PersonHairColor` `IssueDate`, `ExpirationDate`, etc.; so, an `Identification` element has complex content (i.e., sub-elements). Usually one or more of its sub-elements will be `ID` elements.

- **`Text(Type)` vs. `Name(Type)`** — In the construction of NIEM element names, `Name` and `Text` are authorized representation terms (of type `NameType` and `TextType` respectively). The term `Name` is a word or phrase that constitutes the distinctive designation of and applies to a specific person, place, thing or concept. This is not necessarily an identifier, for example, there are multiple persons with the name "Bob". `Text` is a word or phrase in some language (usually English). A is a special subset of .

- **`Date(Type)`** — The only date and time format supported by W3C XML Schema is a subset of ISO 8601. NIEM uses this for date and time.

- **`Indicator(Type)`** — This term designates NIEM Booleans whose valid values are TRUE or FALSE. Do NOT change these values to YES/NO or 1/0. If required, translation to and from TRUE or FALSE is not difficult.

- **Best examples of data names** are in the current NIEM releases, in particular NIEM Core `niem-core.xsd v3.0`, or the Core from the most current major release.

Search NIEM

How to search for NIEM data components:

Although this section does not address NIEM modeling specifically, given the large number of NIEM data components, most modeling efforts will require searching the NIEM model for content to reuse,

duplication, examples of content models, points of extension, etc. This section introduces some basic methods of searching for NIEM content.

Schema Subset Generation Tool (SSGT)

The Schema Subset Generation Tool (SSGT) is one way to search for NIEM data components. This tool was designed to build and generate NIEM schema subsets. However, SSGT also contains a convenient search engine that can be used alone.

Basic Search

SSGT provides a full text search capability for a NIEM release. You can enter a search string into the criteria box and view a list of all NIEM data components that contain a lexical match to that string. The default is to search all data component text records in a single release, including names, definitions, and other metadata text fields. Namespaces are not part of the standard search. This is because namespace prefixes are extremely prevalent throughout the model and would tend to clutter most search results. As will be described later, the user can configure SSGT to search in particular ways.

Search is case independent. A search for `sonty` will return the data component `PersonType`.

SSGT can only search a single release (Later, we discuss a technique for searching multiple releases at once). The default is to search the most recent (current) operational release. If you need to search a previous NIEM release you must select it from the "options" menu.

SSGT always requires that you declare the class of the data component you are looking for, and only that class will be in the scope of the search. This means you must declare one of the following in the dropdown box:

- Property (the default class) - XSD data elements and attributes in a NIEM release.
- Type - XSD data types in a NIEM release; may be complex or simple.
- Namespace - namespaces used in a NIEM release
- Facet - code values in a NIEM release.
- External - profiles of standards used in a NIEM release that are not NIEM conformant (e.g., GML).
- Association - relationships among two or more objects.

Search results are displayed in a list of NIEM data component qualified names, i.e., QName syntax: namespace prefix, followed by colon character, followed by component name; for example, `nc:PersonFullName`.

Depending on the outcome of a simple (default) search, a resulting list will have as many as three parts in this order:

1. Exact match on name: All data names for which the search criteria exactly matched that entire naem (excluding its namespace prefix).
2. Partial match on name: All data names for which the search criteria matched a substring within that name (excluding its namespace prefix).
3. Match on other metadata: All data names for which the search criteria matched a substring within its associated data definition or other metadata (including namespace prefixes that may

occur within the metadata).

Each part of the search result (if it exists) is sorted alphabetically by Qname, which includes the namespace prefix so that component names are conveniently grouped by namespace.

SSGT does not indicate the two points (if either exists) at which these parts are joined. The only way to identify these break points is by visually inspecting the result list to find the locations where alphabetic ordering of names restarts.

Each data name is hyperlinked to its associated metadata and relationships. For an element this includes its definition, keywords (synonyms, if any), usage information (if any), usage examples (if any), its own type, and the types that use this element. For a type this includes its definition, content style, elements it contains, elements that are of this type, and base types from which it is derived. For convenience there are many features that help with identification of data components in the display. Property names (XML element and attribute names) are displayed in blue, and XML type names are displayed in red. XML element and attribute names are usually displayed with their associated XML type names. Hovering over a data component name will reveal its definition. A "details" link to the right of a property name and associated type name will reveal metadata about that property.

Note for MS Internet Explorer 11 users: This browser must be reconfigured to "compatibility mode" (not its default) for SSGT to work correctly:

1. In MS Internet Explorer 11, open Compatibility View (Tools / Compatibility View Settings).
2. Add tools.niem.gov and/or niem.gov to Compatibility View.
3. Go to https://tools.niem.gov/ and SSGT Search will work normally.

Advanced Search

In general, the SSGT advanced search capability provides additional constraints that limit the scope of a search.

Click `Show Advanced` to open the SSGT advanced search capabilities. Here you can constrain the search criteria:

1. with the exact phrase
2. with at least one of the words
3. without the words

or any combination of these constraints.

You can also constrain search to particular metadata fields that you select. These are Names, Definitions, Keywords (synonyms), Usage (descriptions of how a data component is commonly used), and Example Content (samples of valid values for the element).

You can limit a search to one or more domains by checking/unchecking the appropriate boxes. If you want to search content outside of all domains (for example, NIEM Core and all code lists), then just check "other". If you uncheck all boxes the tool will go back to searching the entire model (same as if all boxes are checked).

Multiple search criteria

You can enter multiple search terms separated by spaces in the search criteria box. In this case, SSGT search will return data component records that contain ALL those terms in any order.

Wildcard search

SSGT also has one simple wildcard character, an asterisk. It can be used as many times as desired within the search criteria box. For example, the following search criteria:

```
son\*nor\*sim\*yp
```

will return:

```
aamva\_d20:PersonOrganDonorCodeSimpleType
```

because this criteria matches this element name. A single asterisk in the search criteria box will return all data components in the release. (NOTE: After a wildcard search SSGT replaces each asterisk in the criteria with a percent character.)

Movement

Movement enables you to search and explore the content of the NIEM model. It provides a user friendly interface, and smart search results. It is also open source: the community who inspired its creation can contribute to it.

See more information about Movement and visit the tool itself.

NIEM Model Spreadsheet

For anyone relatively new to NIEM, the Model Spreadsheet is an easier way to search, navigate, and explore the model, and familiarize with NIEM content and structure relatively quickly.

Each NIEM release package contains a model spreadsheet that opens with Microsoft Excel. If you download and unzip a NIEM release package at https://release.niem.gov/ the spreadsheet will be `niem-m.n.xlsx` where `m` is the major (series) version integer, and `n` is the minor version integer.

The model spreadsheet is organized into the following tabs:

- NIEM Core data components, followed by …
- Each domain data component set in alphabetical order, followed by …
- Type Description - a list of all XML types and their definitions
- Elts sorted by type - a list of all XML elements
- Elts sorted by name - a list of all XML elements
- Attributes - a list of all XML attributes
- Namespaces - a list of all namespaces used in the release
- Class Tree - a list of all XML types arranged in a hierarchical derivation tree

Each tab contains a list that also includes various metadata about the data components listed. Each type

is hyperlinked to the location of its definition. The model spreadsheet is designed to explore types and their associated properties (sub-elements).

Excel menu `Find` and `Search` functions also take wildcards. Asterisk and question mark characters are used for multi- and single-character wildcards respectively. To find asterisks and question marks in the text, you must escape these characters with a tilde (~). So, `~*` and `~?` will find asterisks and question marks. To escape the tilde character itself, enter `~~`.

Excel offers other kinds of wildcards and features that provide search capability similar to regular expressions. PC users might look at https://support.office.com/en-us/article/Find-and-replace-text-by-using-regular-expressions-Advanced-eeaa03b0-e9f3-4921-b1e8-85b0ad1c427f#__toc286834987. It would be nice in both PC and Mac Excel had identical user interfaces, but that is not the case. So, for Mac users should check out http://searchengineland.com/advanced-filters-excels-amazing alternative-to-regex-143680 (and beware, shortcuts for Mac Excel 2011 don't always seem to work as expected).

Other methods to search NIEM content

Offline

Because NIEM is XML-based, most methods of searching NIEM are text oriented. Almost any good XML tool can be useful. Common XML editors that can do this are Altova XMLSpy and oXygen.

Other methods of searching NIEM use full text search capabilities that can quickly scan through large file sets. NIEM schemas contain an entire textual data dictionary. Multiple NIEM releases can be searched simultaneously using some version of grep (originally a Unix utility and available natively on an Apple MacIntosh; there are also version that run under Microsoft Windows). Grep is simple, fast, and facilitates user customization. By downloading all the NIEM release packages and placing their XML schemas into a single directory, grep can be customized to find and extract any text that can be described by a grep expression.

Online

Another method of searching NIEM schemas is through a Google site search. All NIEM component names and definitions are online in open text, both in their XSD form as well as XHTML. For example, https://release.niem.gov/niem/3.2/ is a Web page that indexes all the NIEM 3.2 schema documents. Furthermore, at https://release.niem.gov/ are all the NIEM release products since 2006. Each release is documented in HTML pages rooted at these links:

- NIEM 3.2 Namespaces Documentation
- NIEM 3.1 Namespaces Documentation
- NIEM 3.0 Namespaces Documentation
- NIEM 2.1 Namespaces Documentation
- NIEM 2.0 Namespaces Documentation
- NIEM 1.0 Namespaces Documentation

You can browse these pages that contain the contents of each NIEM release organized by namespace. The data components in each namespace are listed alphabetically by name, and each name is linked to a page that displays its definition and other metadata. Each namespace page also contains a direct link to its associated schema document.

You can also search both the release schemas and HTML documentation pages with Google. To do this, run a Google *site search* on data component names and definitions in the release schemas (XSD) or the namespace documentation pages (XHTML).

For example, in your browser address field, enter:

```
site:release.niem.gov/niem/ PersonType
```

Google search results will contain links to any Web page within the release.niem.gov/niem/ directory that contains a match to `PersonType`.

You may enter multiple search terms. For example, enter:

```
site:release.niem.gov/niem/ PersonType location
```

Search results will return links to pages that contain both `PersonType` and `location` in any order. Character case is ignored.

Note that as an Internet search engine, Google generally only indexes full words (tokens surrounded by space or delimiters). Google also uses "stemming" (recommend you Google that) and other heuristics to find pages with words related to search terms. So, while SSGT can NOT find substrings within words (i.e., at the lexical level) that match search terms (such as `sonty` in the earlier example), Google can find partial matches to sets of words in textual paragraphs, sentences, and phrases. For example, if you enter:

```
site:release.niem.gov/niem/ Pers
```

Google search returns pages that contain the string `pers` bound by spaces or other delimiters/separators (i.e., `pers` as a word), but it does not return pages that contain `Person` or `PersonType` because it only indexes words (which are bound by delimiter or separator characters; such as spaces, quotes, and other special characters).

Despite this limitation, Google has a number of powerful operators that can be applied to perform advanced site searches on NIEM schemas and documentation pages. See Google Advanced Search Operators.

Mapping to NIEM

This section describes how to map a database, model, dictionary, etc. to NIEM.

Note that this section is NOT meant to be an in-depth treatise on mapping. It is to provide an overview, some guidance, and suggestions for approaching this task. Mapping is usually hard and can be tedious, but it's also necessary for making a legacy system work with NIEM.

Purpose of mapping

niem adopters generally do not start without pre-existing data requirements. they often begin with existing legacy metadata associated with a database or data model. this metadata usually contains most or all of their data requirements. niem is designed for sharing information with organizations and

communities outside their sphere of influence. so, the primary targets for mapping to niem are those data requirements that will be shared externally. That said, a NIEM adopter may choose to map all his/her metadata to NIEM; however, this is not necessary.

Mapping is an important step to NIEM adoption because it:

1. Provides familiarity with the contents of NIEM.
2. Helps a NIEM adopter to consider and evaluate the scope of effort. For example, it may only be necessary to identify data requirements for sharing information with external entities (and map associated metadata to NIEM).
3. Establishes a detailed record of the semantic and structural associations between existing legacy metadata and NIEM representations.
4. Provides programmers (who will code or modify code for implementing NIEM information exchanges) the guidance for transforming source components to NIEM components and the reverse.

In computing and data management, data mapping is the process of identifying/recording semantic similarity between data elements of different data models or schemas. Data mapping is used as a first step for a wide variety of data integration tasks that often include data transformation or data mediation between a data source and a destination.

A mapping from a source (model, schema, database, etc.) to NIEM will have three functionally useful parts:

- A record of semantically equivalent data component matches from the source to NIEM.
- A gap analysis that identifies how well NIEM covers the source model or schema. This provides guidance for creating new NIEM components to meet missing data requirements.
- A description (or guidance) that indicates how the source and NIEM can be translated or mediated from one to the other.

How to map to NIEM

Although heuristics can be applied, few, if any, tools exist that automatically, accurately, and completely map between data models. There are far too many ways to represent and define data, and everyone has a different perspectives. A few tools exist that provide some degree of intelligent (often heuristic-based) help by "suggesting" possible matches. However, mapping between different data models, databases, or data schemas will still require tedious human review. Most tools that support mapping, help with the organizational and recording aspects. A spreadsheet is one of the simplest tools for documenting a map between a source (database, model, etc.) and NIEM. The basic concept of mapping is to match (and align) appropriate metadata components from the source to semantically equivalent or nearly equivalent components in NIEM.

Referring to https://en.wikipedia.org/wiki/Schema_matching, two terms are used to describe data mapping and are often used interchangeably: *schema matching* and *mapping*.

- *Schema matching* is the process of identifying that two data objects are semantically related (or equivalent).
- *Mapping* refers to transformations between the data objects or components mapped that translate meaning and preserve meaning from one model to the other.

This NIEM guidance will use the term *mapping* to refer to both, because both schema matching and mapping can be combined into a single document.

Map construction

Given a spreadsheet paradigm, a row identifies a semantic association (i.e., a match) between a source component and a NIEM component, and describes the nature of that association.

There are generally three possible outcomes for an association: exact, partial, or none. An *exact match* means that the components are semantically equivalent (or close enough). At the other extreme, *no match* (none) means that there are no NIEM components that can match or even partially match the source component. In this case, there will be no NIEM component identified with this record (until one is designed). This indicates a gap in the mapping for which a new NIEM component is required; that component will be designed (and proposed) later as part of the NIEM adoption and on-boarding process for new domains. And of course, somewhere between an *exact match* and *no match* lies a *partial match*.

Various types of information may be used to evaluate a match, including (but not limited to):

- Model or schema metadata such as element names, data types, structures, constraints (e.g., cardinality), etc.
- Instance data and usage information.
- Information from dictionaries, thesauri, ontologies, previous matching decisions, and feedback from others.

At present, a small number of NIEM components also include synonym, usage, and sample value information that may be helpful. This information can be found within SSGT (when it exists). The NBAC and Domains are always welcomed and encouraged to suggest additions to this kind of information.

Thus, for a given component of the source model, a matching generally requires three basic parts:

Source model component	**NIEM component**	**Description of mapping**
Name	Name	Nature of semantic match (one of):
Definition	Definition	- exact match
Datatype	Datatype	- partial match
etc.	etc.	- none (missing in NIEM)

For convenience and completeness mappings usually incorporate any additional metadata associated with each source and NIEM component (such as definition, datatype, cardinality, etc.). Also, a column for general comments/notes (not shown above) is incorporated into most mapping documents to ensure clarity and to record rationales for difficult associations.

Mapping strategy

While many strategies may exist for performing mapping between models and/or schemas, most of these can be classified into one of two general approaches: top down and bottom up.

Our own experience with mapping to NIEM indicates that a top down approach works best. This is probably because NIEM tends to employ a top down structure for data representations. NIEM is object-oriented, and therefore, models real world entities and concepts as collections of properties. The objects/concepts are usually complex and often the properties for such are also complex. So, for example a Person is a complexType, PersonType. PersonType has properties such as PersonBirthDate, PersonHairColorCode, PersonEyeColorCode, and PersonName. While the first three properties mentioned are simpleTypes, PersonName is type PersonNameType which is a complexType. PersonNameType breaks down into several simpleType elements representing the various parts of a person's name (such as first, middle, last, etc.).

Using a top down approach, identify the entities represented in your own model (or schemas) and match them to the same objects/concepts in NIEM (if they exist). There will likely be differences in the properties contained in each pairing. However, by starting with the top level entities you will likely narrow your search for matching properties at the atomic level. Map objects and their properties based on semantics not datatype.

Even if semantics of two properties are identical, datatypes will not always be the same. For example, you may determine that the source model represents a person's name as a simple string, `Name` that concatenates first, middle, and last names. NIEM represents a person's name as a complex element, `PersonName`, consisting of three separate string elements `PersonGivenName` (first), `PersonMiddleName`, `PersonSurName` (last). Each of these is only a partial match to the source element for a person's name. However, in this example, `Name` is an exact match to `PersonName`, and the transformation from one to the other (or reverse) is a simple set of operations that involve either concatenation or parsing depending on the direction of the transform.

If an object or concept in your model does not have an equivalent or similar entity in NIEM, flag it as a potential shortfall. Later you will build this object into NIEM as part of your new domain model (to be discuss later). Likewise, when comparing atomic level object/concept properties you should also flag properties in your model that are not represented in NIEM.

Handling partial matches

Exact matches and no matches are fairly easy to handle. An "exact match" reuses a NIEM component as-is, and a "no match" generates a new NIEM component.

However, partial semantic matches tend to be the norm. Sometimes they can be resolved by breaking down complex types into constituent components and subsequently mapping to them. However, most schemes and models for data representation have many dimensions that can complicate what would otherwise appear to be an exact or near exact match, including names, semantics, stucture, constraints, datatype, usage, and others. For this reason, sometimes treating a partial match as "no match" (and subsequently designing new NIEM components) may be more practical and productive.

[More to follow, including examples, Change Request, Component Mapping Template]

Evaluate component quality

Considerations in identifying, designing, defining, and structuring data components

Questions to ask or consider about metadata, semantics, and structure while modeling NIEM content. These can help you decide what to model, how, and what NIEM structures to employ or consider.

For each potential NIEM Core data component (type/property):

1. Is it a NIEM property or type?
    - Is it common enough to be part of NIEM Core? (depth of NIEM Core)
    - Or is it better used as a Domain extension to NIEM Core?
    - Is it required in NIEM information exchanges (IEPD) or messages? (scope)
    - What kinds of data exchanges use it or would use it?
    - Is it reusable in other messages?
    - Can it be used in composing IEPDs?
    - Does it overlap multiple requirement sources (models)?
    - What data models contain this component or one similar to it? (source/requirement)
2. Is it a characteristic property or subpart of the appropriate NIEM object (type)?
3. Is it too specific (not reusable)?
    - Should it be generalized to include in NIEM Core?
    - Better used as a Domain extension to NIEM Core?
4. Does this property or type overlap with another? Not distinct?
    - Should it be joined with another component?
    - Should it be split into multiple components? (too broad)
    - Select one; remove the other(s)?
5. Is its name meaningful to NIEM?
    - Is it properly named?
    - Are the terms of its name meaningful?
    - Are there meaningful definitions for each term?
    - Are the proper word senses clear or identified within the definition?
    - If a name term is local language (i.e., not in OED, or in OED but not using an OED word sense), jargon, acronym, or abbreviation are these terms defined with appInfo in the schema documents in which they are used?
6. Is its definition meaningful in NIEM?
    - Is it defined distinctly from other NIEM Core data components?
    - Is it distinct from other Domain components?
7. Is it modeled correctly (structurally correct; correspond to real-world)?
    - Does it have the right base-types and properties?
    - Is it missing common characteristic or subpart properties (simple/complex)?
    - Does it contain properties that should be deleted, replaced, or changed?
8. How is it related to other objects (types)?
    - Use NIEM associations?
    - Use NIEM roles?
9. How is it used? (primary context? multiple contexts?)

Appendix A - Governance

Regular meetings

- **NIEM Business Architecture Committee (NBAC)** — NBAC meets (by teleconference) on the last Thursday of every month from 2:00-3:00pm. Its primary missions are to identify the data

requirements for NIEM, govern NIEM [Core], and assist the domains with the management of their models. NBAC coordinates with NTAC, the PMO, and the domains.

- **NBAC Content Subcommittee** — This group is a standing subcommittee under [NBAC][NBAC] that meets (by teleconference) as often as necessary to resolve issues, refactor (remodel) NIEM content, and harmonize. During a NIEM release cycle this subcommittee meets fairly regularly to prepare for the initial alpha release. Meeting times and dates are subject to change and are established by the lead developer in coordination with subcommittee members.
- **NIEM Technical Architecture Committee (NTAC)** — NTAC meets (by teleconference) every other week on Wednesday from 1:30-3:00pm. Its primary missions are to identify new NIEM technical requirements, govern the NIEM architecture (the structure of the model), oversee the NIEM technical specifications and the tool strategy. NTAC coordinates with NBAC, the PMO, and the domains.
- **Program Management Office (PMO)** — PMO holds a regular coordination meeting (via teleconference) every other Monday from 1:00-2:00pm. This meeting is generally for the PMO staff, goverance committee co-chairs, lead developer, and key representatives of the largest domains.
- A **Joint NTAC/NBAC Face-to-Face meeting** is generally hosted twice a year by the PMO in the Washington DC area. Exact dates vary, but meetings are generally 2-3 days in Apr/May and Oct/Nov timeframes. Dates are usually timed to coincide with key milestones in a release cycle. These meetings are generally for all NBAC and NTAC members, all domain stewards or representatives, the lead developer, and the PMO staff. These meetings have multiple purposes, including reviewing material for a developing release, technical updates, reviewing the previous year, planning the next year, coordination between NBAC and NTAC, etc.

Email lists

If you are a domain representative, an NBAC member, or an NTAC member you may want to register for one or more of the following resources in order to participate more fully in NIEM governance and its associated processes.

- [NIEM Technical Architecture Committee](mailto:ntac@lists.gatech.edu (NTAC); request to be added (requires PMO approval).
- NIEM Business Architecture Committee (NBAC); request to be added (requires PMO approval).
- NIEM Configuration Control Tool (NCCT); for PMO, governance, and domain staffs; request account
- Collaboration zones for PMO, governance, and domain staffs; request to be added
- Conformance Testing Assistant (ConTesA); public; request account
- NIEM GitHub; public; sign-up for GitHub account

Questions and feedback

If you are a domain representative developing NIEM content or maintaining your domain namespace, or if you are an NBAC or NTAC member, then please talk directly to the lead developer for technical assistance. This is your privilege as a member of NIEM governance. The lead developer (GTRI) participates in all NIEM governance meetings, so please don't hesitate to ask the lead developer for technical help. They will be happy to schedule a telephone appointment to talk with you. Asking questions, making suggestions or recommendations through these resources are more for your

constituents. You can use these resources too, but for technical assistance it is usually faster to contact the lead developer directly.

- General technical comments and feedback: niem-comments@lists.gatech.edu
- NIEM Contact Center: information@niem.gov
- NIEM Contact Center: https://www.niem.gov/contact-us

Appendix B - Resources

URLs to releases, technical references, tools, templates, and other aids

Releases	URL	Notes
All NIEM release packages	https://release.niem.gov/	
NIEM domain updates, core supplements	https://publication.niem.gov/	
NIEM 5.0 release	https://release.niem.gov/niem/5.0/	

References	URL	Notes
NIEM technical references	https://niem.github.io/reference/	
NDR Schematron rules	https://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/niem-ndr-5.0.zip	use with Oxygen
NIEM 2.1 UML profile	http://www.omg.org/spec/NIEM-UML/1.0/	for NIEM 2.1 only
NIEM 3.0 UML profile	http://www.omg.org/spec/NIEM-UML/3.0/Beta1/	NIEM 3.0, 3.1, 3.2
Conformance Specification v5.0	https://reference.niem.gov/niem/specification/conformance/5.0/	
Naming and Design Rules (NDR) v5.0	https://reference.niem.gov/niem/specification/naming-and-design-rules/5.0/	
Conformance Targets Attribute Specification v3.0	https://reference.niem.gov/niem/specification/conformance-targets-attribute/3.0/	
Code Lists Specification v4.0	https://reference.niem.gov/niem/specification/code-lists/4.0/	
Information Exchange Package Documentation (IEPD) Specification v5.0	https://reference.niem.gov/niem/specification/model-package-description/5.0/	
High Level Version Architecture v3.0	https://reference.niem.gov/niem/specification/high-level-version-architecture/3.0/	
Change request v1.0	https://reference.niem.gov/niem/resource/change-request/	
Wantlist schemas (all versions)	https://reference.niem.gov/niem/resource/wantlist/	

Reference Tools	URL/Email	Notes
NIEM online reference tools	https://tools.niem.gov/	
Schema Subset Generation Tool (SSGT)	https://tools.niem.gov/niemtools/ssgt/index.iepd	Search, XML subsets
Conformance Testing Assistant	https://tools.niem.gov/contesa/	

Reference Tools	URL/Email	Notes
(ConTesA)		
Code List Schema Generator	https://tools.niem.gov/niemtools/codelist/index.iepd	Legacy; for NIEM 2.0
NIEM GitHub	https://niem.github.io/	
NIEM Releases Issue Tracker	https://github.com/niem/niem-releases/issues	For content-related issues

Other Tools	URL/Email	Notes
Wayfarer (online; by Tom Carlson)	http://niemwayfarer.org/	search NIEM
Wayfarer (for download; by Tom Carlson)	http://niemwayfarer.org/tools/niemsaw/	search NIEM
Oracle Content Assembly Mechanism (CAM)	https://sourceforge.net/projects/camprocessor/	XML editor, validator, designer
MITRE Openii	http://openii.sourceforge.net/	information integration tool suite
Altova XMLSpy	http://www.altova.com/xmlspy.html	XML editor (not free)
Altova NIEM solutions	http://www.altova.com/solutions/niem.html	other Altova support for NIEM
Oxygen XML Editor	https://www.oxygenxml.com/	XML editor (not free)

Other Resources	URL	Notes
NIEM home	https://www.niem.gov/	
Change request	https://reference.niem.gov/niem/resource/change-request/	
IEPD Clearinghouse	https://www.it.ojp.gov/niss/	mostly older IEPDs
ISO/IEC Standard 11179 Information Technology – Metadata Registries (MDR)	http://metadata-standards.org/11179/	guidance for names and definitions

Appendix C - NIEM glossary

This appendix defines terminology associated with NIEM. The first section defines common terms that are not formally defined in NIEM specifications. Subsequent sections reference the definitions for terms used in the NIEM specifications. Each list is sorted alphabetically.

Common NIEM terms

- **coordinated domain update** — A NIEM [domain update] that is a collaborative effort among multiple (two or more) domains. Simultaneously, each domain releases a new version of (update to) its namespace(s). Together these updates will be [harmonized][harmonization] and [coherent][coherence].
- **data component** — An NIEM XML element, attribute, or type.
- **funneling** — Describes a process for subjective reviewing of an intermediate [NIEM release]. Once governance has reviewed, identified, and resolved a subjective modeling issue within an intermediate release, the issue is closed and will not be reopened in a subsequent review during that [release cycle]. Funneling begins at the start of a [beta] release. Note that funneling is only applicable to subjective issues; errors in the model will be corrected at anytime during the [release cycle].

- **intermediate release** — An [alpha], [beta], or [release candidate] release (i.e., not an operational release).
- **term** — A word, acronym, or abbreviation concatenated in upper camel case with other terms to form a NIEM data component name. For example, the NIEM data component name `nc:PersonSSNIdentification` consists of three terms: `Person`, `SSN` (an acronym for Social Security Number), and `Identification`.

NIEM and JSON

Until recently, XML has been the single format for NIEM data. NIEM is now in the process of adopting JSON (Javascript Object Notation) as a second standard format for representing data based on the NIEM data model.

These pages are designed to provide an easy introduction to the purpose of NIEM, the reasons to use NIEM with JSON data, and the developer knowledge needed to put NIEM JSON into practice. They are intended for:

- Developers who know JSON but know nothing of NIEM or XML
- Developers familiar with NIEM XML who need a JSON version of a data exchange
- Data scientists wanting to understand or provide JSON data sources
- Managers wondering if NIEM JSON is right for their project

This is the starting page for learning about NIEM and JSON. Readers will probably follow links to the content of interest and bypass the rest. Some starting choices are:

- A NIEM-JSON tutorial

  This tutorial walks the reader through a small example of implementing an information exchange using NIEM-JSON. It begins with a small set of data requirements, constructs a NIEM information exchange model, expresses data for that model as JSON, then constructs a JSON schema for that data.

- NIEM and JSON Frequently Asked Questions

  Explains what NIEM and JSON each are and do and why they are useful together.

- A NIEM-JSON Reference Guide

  Provides normative and non-normative guidance on how NIEM and JSON are used, including lots of examples.

NIEM JSON is a work in progress. These pages are intended to help developers, data scientists, and managers understand the different ways they might use NIEM and JSON together. The **NIEM JSON Specification** (or see the overview) defines NIEM JSON conformance targets to accompany the existing NIEM XML conformance targets. Open source tools in the planning stage will help developers switch between JSON and XML (and perhaps other serializations) by creating translators for runtime data and conversions for build-time artifacts.

The intent is to make the NIEM approach valuable to anyone wanting to create or apply community agreement on data semantics, using these to provide precise definitions of data available through any sort of API, or to document and harmonize published data resources for data analytics.

A NIEM-JSON tutorial

This page walks through a simple example of implementing an exchange using NIEM-JSON.

1. Define data requirements

First, we start with a set of very simple data requirements. In this case, we'll use a brief set of data requirements, organized hierarchically. Each sub-bullet is a property of whatever contains it. The top concept is a motor vehicle crash.

- motor vehicle crash [1]
  - involved vehicle: vehicle [1-n]
    - vehicle identifier number (VIN) : identifier [1]
    - driver : person [1]
      - name : string [1]

2. Define a NIEM exchange model

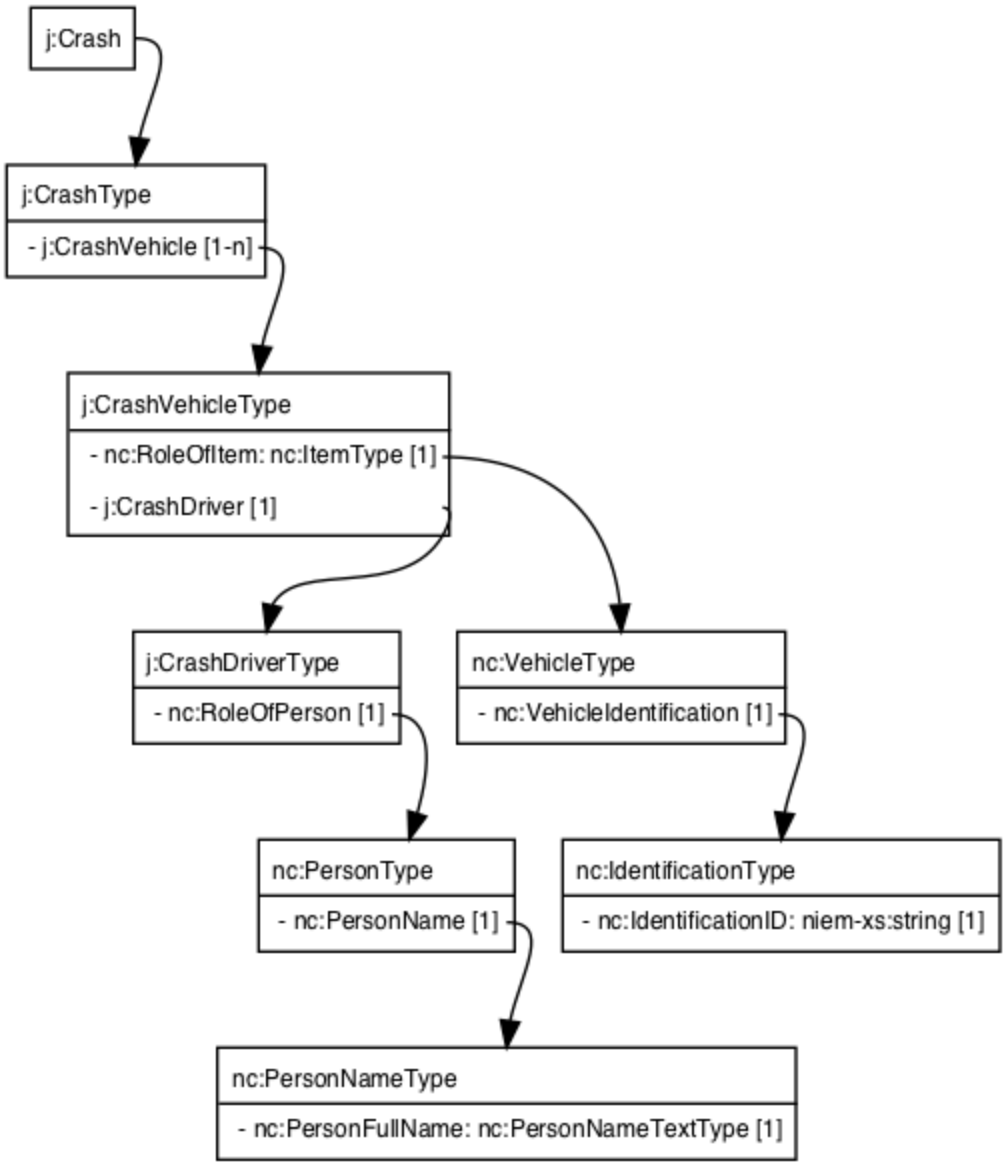
Next, we search for these data requirements in the NIEM Subset Schema Generation Tool (SSGT), yielding a NIEM object model.

> The SSGT is a tool that enables users to search through the NIEM data model and build a NIEM subset. There are other ways to search the model, including NIEM Movement. If you are new to NIEM, it is recommended that you take some time to understand how the data model works. See NIEM's model reference documentation to learn more. Additional training is forthcoming.

We have searched for these data requirements in the SSGT and found corresponding NIEM elements.

> You may download and save the resulting list of components as a NIEM wantlist, which you may load into the SSGT.

The NIEM representation of these data requirements contains more components than the list of source requirements above, due to the way NIEM structures its data: A crash vehicle is represented as a *role* of a vehicle, a crash driver is represented as a *role* of a person, and a person's name and a vehicle identifier are each represented by a structure that can carry more than just a simple value.



The above object model shows type information and cardinality, which won't appear in a resulting message.

3. Define a JSON-LD context

Our next step is to construct a JSON data example that uses the above object model. First, we construct a JSON-LD context object that identifies all of the namespace prefixes that are needed for the instance. Each NIEM namespace is converted into a URI according to the NIEM NDR, which requires that NIEM namespace URIs have the # character (i.e., number sign) appended, if one is not already present. Here is the resulting context:

```
{
  "j": "http://release.niem.gov/niem/domains/jxdm/6.0/#",
  "nc": "http://release.niem.gov/niem/niem-core/4.0/#"
}
```

4. Construct a sample JSON file

Next, we construct a corresponding JSON file. To do this, we walk through the above object model, adding the NIEM property names as JSON properties.

Every property that does not have a simple value (such as a string or a number) is represented as an object. For example, the `j:Crash` element is represented as an object:

```
"j:Crash" : {
  ... value for the crash property ...
}
```

If a property has *only* a single simple value, then it may be represented as a literal value, rather than an object. For example, the `nc:PersonFullName` element here needs only a simple value:

```
"nc:PersonFullName": "Gerry H. Everett"
```

If a property occurs more than once, then it must appear as a property with a value that is an array; each value of the array is a separate value of the property. For example, if an organization has 3 employees, it may appear as an "nc:Employee" property with a value of an array containing 3 objects, each representing a person (e.g., an object representing an instance of NIEM type `nc:PersonType`).

```
"nc:Employee": [
  { "nc:PersonFullName": "Gerry H. Everett" },
  { "nc:PersonFullName": "Floyd Q. Halverson" },
  { "nc:PersonFullName": "Malcolm P. Smith" }
]
```

4.1. Use JSON-LD features as needed

NIEM JSON leverages JSON-LD to provide features not present in plain JSON. For example:

- JSON-LD contexts are used to provide namespace features. Contexts have many features this walkthrough doesn't exercise. See the page on JSON-LD contexts* for more information.

- You may use the property `rdf:value` to represent an object that has a both attributes and a simple value.

- You may use the JSON-LD property `@id` to perform the function of NIEM XML's `structures:id`, `structures:ref`, and `structures:uri`.

- JSON-LD allows arrays to be used in instances very freely; it does not distinguish between an array with 1 entry and a single value. This is a property with a value that is a single string:

  ```
  "nc:PersonFullName": "Gerry H. Everett"
  ```

  This is a property whose value is *an array* containing one string:

```
"nc:PersonFullName": [ "Gerry H. Everett" ]
```

Under JSON-LD, these two examples have the same meaning. A developer needs to decide what syntax they want in instances, including how and when arrays will appear. These decisions may be enforced using a JSON schema.

- Every property in NIEM has a type. In some instances, the actual value of a property is not fully specified by the type defined for the property in the NIEM data model. In the example above, the property `nc:RoleOfItem` has type `nc:ItemType`, but the instance requires the value to have type `nc:VehicleType`, which is derived from `nc:ItemType`. In this case, and similar cases, you may use the JSON-LD property `@type` to identify the type of the property's value, as shown below.

4.2. A complete sample instance

As described above, we walk through the NIEM exchange model, adding each NIEM property as a JSON property. This results in the following JSON document:

```
{
  "@context" : {
    "j": "http://release.niem.gov/niem/domains/jxdm/6.0/",
    "nc": "http://release.niem.gov/niem/niem-core/4.0/"
  },
  "j:Crash": {
    "j:CrashVehicle": [
      {
        "nc:RoleOfItem": {
          "@type": "nc:VehicleType",
          "nc:VehicleIdentification": {
            "nc:IdentificationID": "5TFEM5F10EX067865"
          }
        },
        "j:CrashDriver": {
          "nc:RoleOfPerson": {
            "nc:PersonName": {
              "nc:PersonFullName": "Gerry H. Everett"
            }
          }
        }
      }, {
        "nc:RoleOfItem": {
          "@type": "nc:VehicleType",
          "nc:VehicleIdentification": {
            "nc:IdentificationID": "1FTSW2BR5AEA23373"
          }
        },
        "j:CrashDriver": {
          "nc:RoleOfPerson": {
            "nc:PersonName": {
              "nc:PersonFullName": "Chester D. Barbee"
```

```
                }
            }
        }
    }
  ]
  }
}
```

This example is available in the JSON-LD Playground.

5. If needed, construct a JSON schema for your instances

NIEM does not require the use of JSON schema for your JSON messages. NIEM merely requires that JSON documents have the correct format. However, you can create a JSON schema for your messages if you need to. Here, we'll step through the creation of a JSON schema for JSON data like the sample above. Note that there's nothing *normative* about this schema; this walkthrough is merely laying out one way to build a JSON schema from a NIEM exchange model. There are many schemas you could create for the same model.

5.1. Framework for an instance

First, we lay out the basic structure of the JSON schema. This does not have any content; there's nothing from the NIEM exchange model we defined above. It's a framework into which we'll put the definitions we need to express the NIEM exchange model.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
  },
  "additionalProperties": false,
  "required": [
  ]
  "definitions": {
  },
}
```

The JSON Schema properties we are using here are:

- `$schema`: Indicates the JSON Schema version being used by this schema.
- `type`: Directs that this piece of a schema will only match a JSON object.
- `properties`: Describes all the properties that are allowed by this piece of the schema
- `additionalProperties`: Indicates that this piece of the schema *only* allows properties that are listed under `properties`.
- `required`: Lists all properties that *must* occur in an object matching this piece of schema.
- `definitions`: A holding place for pieces of schemas which can be reused.

5.2. Top-level properties

We know that we require `@context` and `j:Crash` at the top, so we add them to `required`:

```
"required": [
  "@context",
  "j:Crash"
],
```

We also need these to be available as properties of the top level object, so we'll add them to `properties`. Rather than define their *structure* within `properties`, we'll point them to a definition within the `definitions` section of the schema. The properties `@context` and `j:Crash` are both simple; each references its definition in the `definitions` section of the schema.

```
"properties": {
  "@context": { "$ref": "#/definitions/@context" },
  "j:Crash": { "$ref": "#/definitions/j:CrashType" },
},
```

5.3. Definitions

Next, we write definitions for the content of the top-level properties. These definitions will be small schemas within a `definitions` section of the JSON schema document. First, we add the definition for `@context`, which will accept any object:

```
"@context": {
    "type": "object"
  },
```

The rest of the work will be adding NIEM types for `j:Crash` and recursively for all of its children; this will define the rest of the exchange model. Each definition (besides `@context`) is named after its NIEM type. We start with `j:CrashType`. This type adds a bit of complexity, since its property `j:CrashVehicle` can occur 1-n times (at least once, with no fixed maximum on the number of times it may occur). This means that `j:CrashVehicle` must be defined as an array:

```
"j:CrashType": {
  "type": "object",
  "properties": {
    "j:CrashVehicle": {
      "type": "array",
      "items": { "$ref": "#/definitions/j:CrashVehicleType" },
      "minItems": 1
    }
  },
  "additionalProperties": false,
  "required": [ "j:CrashVehicle" ]
},
```

`j:CrashVehicle` is an array with items matching the definition for type `j:CrashVehicleType`.

The properties defining `j:CrashVehicle` are:

- `type`: Defines this property as an array.
- `items`: Indicates that each item of the array must match the definition for `j:CrashVehicleType`.
- `minItems`: Directs that there must be at least one object within the array, matching the cardinality in the exchange model of 1-n.

This defines a piece of a schema that matches an object with a single property, `j:CrashVehicle`. As above, the definition used for the child element is named for the type of the element, in this case, `j:CrashVehicleType`:

```
"j:CrashVehicleType": {
  "type": "object",
  "properties": {
    "nc:RoleOfItem": { "$ref": "#/definitions/nc:VehicleType" },
    "j:CrashDriver": { "$ref": "#/definitions/j:CrashDriverType" }
  },
  "additionalProperties": false,
  "required": [ "nc:RoleOfItem", "j:CrashDriver" ]
},
```

We continue like this through the object model, adding only what we need to make the schema validate the sample instance. When we get to the definition for `nc:PersonFullName`, it has its definition from type `nc:PersonNameTextType`, which has a string value. This appears in the JSON schema as:

```
"nc:PersonNameTextType": {
  "type": "string"
},
```

5.4. Resulting schema

Repeating the above for the entire exchange model yields the following JSON schema. This validates the above sample instance.

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "@context": { "$ref": "#/definitions/@context" },
    "j:Crash": { "$ref": "#/definitions/j:CrashType" }
  },
  "additionalProperties": false,
  "required": [
    "@context",
    "j:Crash"
  ],
  "definitions": {
```

```
"@context": {
  "type": "object"
},
"j:CrashType": {
  "type": "object",
  "properties": {
    "j:CrashVehicle": {
      "type": "array",
      "items": { "$ref": "#/definitions/j:CrashVehicleType" },
      "minItems": 1
    }
  },
  "additionalProperties": false,
  "required": [ "j:CrashVehicle" ]
},
"j:CrashVehicleType": {
  "type": "object",
  "properties": {
    "nc:RoleOfItem": { "$ref": "#/definitions/nc:VehicleType" },
    "j:CrashDriver": { "$ref": "#/definitions/j:CrashDriverType" }
  },
  "additionalProperties": false,
  "required": [ "nc:RoleOfItem", "j:CrashDriver" ]
},
"j:CrashDriverType": {
  "type": "object",
  "properties": {
    "nc:RoleOfPerson": { "$ref": "#/definitions/nc:PersonType" }
  },
  "additionalProperties": false,
  "required": [ "nc:RoleOfPerson" ]
},
"nc:PersonType": {
  "type": "object",
  "properties": {
    "nc:PersonName": { "$ref": "#/definitions/nc:PersonNameType" }
  },
  "additionalProperties": false,
  "required": [ "nc:PersonName" ]
},
"nc:PersonNameType": {
  "type": "object",
  "properties": {
    "nc:PersonFullName": { "$ref": "#/definitions/nc:PersonNameTextType" }
  },
  "additionalProperties": false,
  "required": [ "nc:PersonFullName" ]
},
"nc:PersonNameTextType": {
  "type": "string"
```

```
    },
    "nc:VehicleType": {
      "type": "object",
      "properties": {
        "@type": { "type": "string" },
        "nc:VehicleIdentification": { "$ref": "#/definitions/nc:Identificatior
      },
      "additionalProperties": false,
      "required": [ "@type", "nc:VehicleIdentification" ]
    },
    "nc:IdentificationType": {
      "type": "object",
      "properties": {
        "nc:IdentificationID": { "$ref": "#/definitions/niem-xs:string" }
      },
      "additionalProperties": false,
      "required": [ "nc:IdentificationID" ]
    },
    "niem-xs:string": {
      "type": "string"
    }
  }
}
```

NIEM and JSON Frequently Asked Questions

- What is NIEM?: NIEM is best understood as a national program with a standards-based process for creating community-agreed reusable data definitions and for assembling those definitions into data resource specifications. This section describes;

    - Reusable definitions and NIEM data model
    - Data resource specifications
    - NIEM technical specifications
    - NIEM governance
    - Different ways to use and participate in NIEM

- Why use NIEM with JSON?: NIEM is primarily about data understanding and interoperability. These things become difficult when there are many data resources, constructed and used by many developers or data analysts. NIEM can help when those problems emerge, by providing:

    - Meaningful data names and machine-readable documentation
    - Global identifiers for data components
    - Self-describing data
    - Reuse of community-agreed data definitions

- What is NIEM JSON data?: JSON data has a hierarchical structure based on simple values, arrays, and objects. NIEM JSON uses JSON-LD to handle what is supported in XML via XML Namespaces and XML ID and IDREF attributes.

    - JSON-LD contexts allow short terms to designate URIs as property names.
    - JSON-LD's ID properties (e.g., `@id`) can identify links between data, as ID and IDREF do (e.g., NIEM `structures:id` and `structures:ref`)

What is NIEM?

NIEM is the acronym for the *National Information Exchange Model*. This implies that NIEM is a data model. While NIEM does in fact have a data model, there is much more to NIEM than that. This section will describe:

- NIEM-based reusable data components
- the NIEM data model
- the NIEM process
- NIEM technical specifications
- NIEM governance

It is best to think of NIEM as a community and standards-like organization similar to the World Wide Web Consortium (W3C) or Open Geospatial Consortium (OGC), whose members voluntarily cooperate to define and employ its technical specifications, data model, etc. While there are of course important differences, some illustrative parallels are:

Organization	NIEM	OGC
Technical specifications	NIEM Naming and Design Rules, etc.	Geography Markup Language (GML) Encoding Standard, etc.
Data model	XML schemas for NIEM model, etc.	XML schemas for GML, etc.

There are several ways to be a part of NIEM, ranging from merely using a NIEM-based data resource specification, to participating in the committees that create and control the NIEM products.

Reusable data components

A data element combines a name, a definition, and a value domain (or data type). For example:

- **Name**: PersonGivenName
- **Definition**: A first name of a person
- **Value domain**: *character string*

NIEM's reusable data components are machine-readable data element definitions in different languages; XML Schema, JSON Schema, and RDF. These components are like LEGO(tm) blocks for data; developers choose the blocks they want and assemble them into the data structure they are trying to create.

The NIEM data model

Following the LEGO analogy, the NIEM data model is a generic box of LEGO blocks that can be used to create many different LEGO structures. The NIEM model is the collection of data components created by the NIEM community, each created because it will be useful in many different data resources. Developers look through the box, choose those components that satisfy their business need, and leave the rest behind.

The NIEM data model is not supposed to be comprehensive. It is not the uber-model of everything for everybody. Developers always leave some part of the NIEM model unused, and developers almost always need to define data components of their own for concepts that are not in the NIEM model. (They are encouraged, but not required, to make those components available for others to reuse.)

The NIEM model is partitioned into several subject-area domain models, each managed by a separate domain community according to its needs. There is also the *NIEM core*, which contains data components thought useful and agreed by all the domain communities. The petal diagram in this section illustrates the NIEM core and current domains. (More on the model management process in NIEM governance.)

The NIEM model is published in two forms. The normative model is currently published as a collection of XML Schema documents. There is also a spreadsheet version. Plans to publish the model as JSON Schema and RDF are on the drawing board.

The NIEM process

Developers use the NIEM process to define the syntax and semantics of some data resource. That resource can be the data passed through an API, or messages sent from one system to another, a data document published on the web, etc. The process is:

1. Determine the information requirements. What are the facts to be conveyed in the data resource?

2. For each concept in the information requirements

   2.1 Search for and select a data component in the NIEM model to represent this concept. NIEM provides tools to assist with this search: the Subset Schema Generation Tool (SSGT) and Movement.

   2.2 Possibly adapt a data component from an external model, if there is no suitable component in the NIEM model. For example, the NIEM model provides several geospatial components adapted from Geographic Markup Language (GML).

   2.3 Define a new data component for the missing concept, if there is no component suitable for reuse. Developers may share their new component for reuse by others, but are not obligated to

do so.

3. Have new or modified information requirements been revealed? Change the requirements in step #1 accordingly and repeat.

4. When all of the information requirements are satisfied, assemble the selected data components into a schema that specifies the data resource. This schema represents the resource's content model, which documents the meaning of each data component, defines which components are mandatory and optional, and may be used to validate individual instances of the resource during development or operations.

The schema is the central component of a NIEM-based data resource specification, for historical reasons called the *Information Exchange Package Documentation (IEPD)*. NIEM supplies an IEPD framework for arranging documentation and gathering all of the artifacts needed by developers and analysts. Any number of consumers can use the specification to understand what to expect in the resource; any number of producers can use it to know what they are expected to provide.

NIEM Technical Specifications

LEGO blocks snap together only because of the precise size and location of each block's pegs and cylinders. If the NIEM model is the set of data blocks, then the NIEM technical specifications are the rules which ensure that separately-developed blocks can be composed into a single schema.

> XML is very flexible. Everyone can do their own thing, and, by golly, everyone is! – Steve McVey at XML '99

At present most of the NIEM technical specifications are written assuming XML as the data syntax and XML Schema as the primary specification language. There are many, many different ways to represent information with XML. A certain level of consistency is required in order to compose many separately-designed XML data components into a single resource specification that makes sense.

This consistency is accomplished by the NIEM Naming and Design Rules (NDR), which constrains the use of XML and XML Schema within NIEM. Most of the NDR rules are written in Schematron, and can be used to automatically assess conformance of the schema in a resource specification (an IEPD). The NDR also provides a meta-model for NIEM XML, based on the Resource Description Framework (RDF), specifying how XML elements and attributes equate to RDF resources, properties of those resources, and values of those properties.

The technical specifications for NIEM JSON are now on the drawing board. One thing already known: the NDR for NIEM JSON (if one is actually required) can be much simpler than the XML rules. This is because NIEM JSON is based on the JSON-LD specification, which already does for JSON much of what the existing NIEM NDR does for XML.

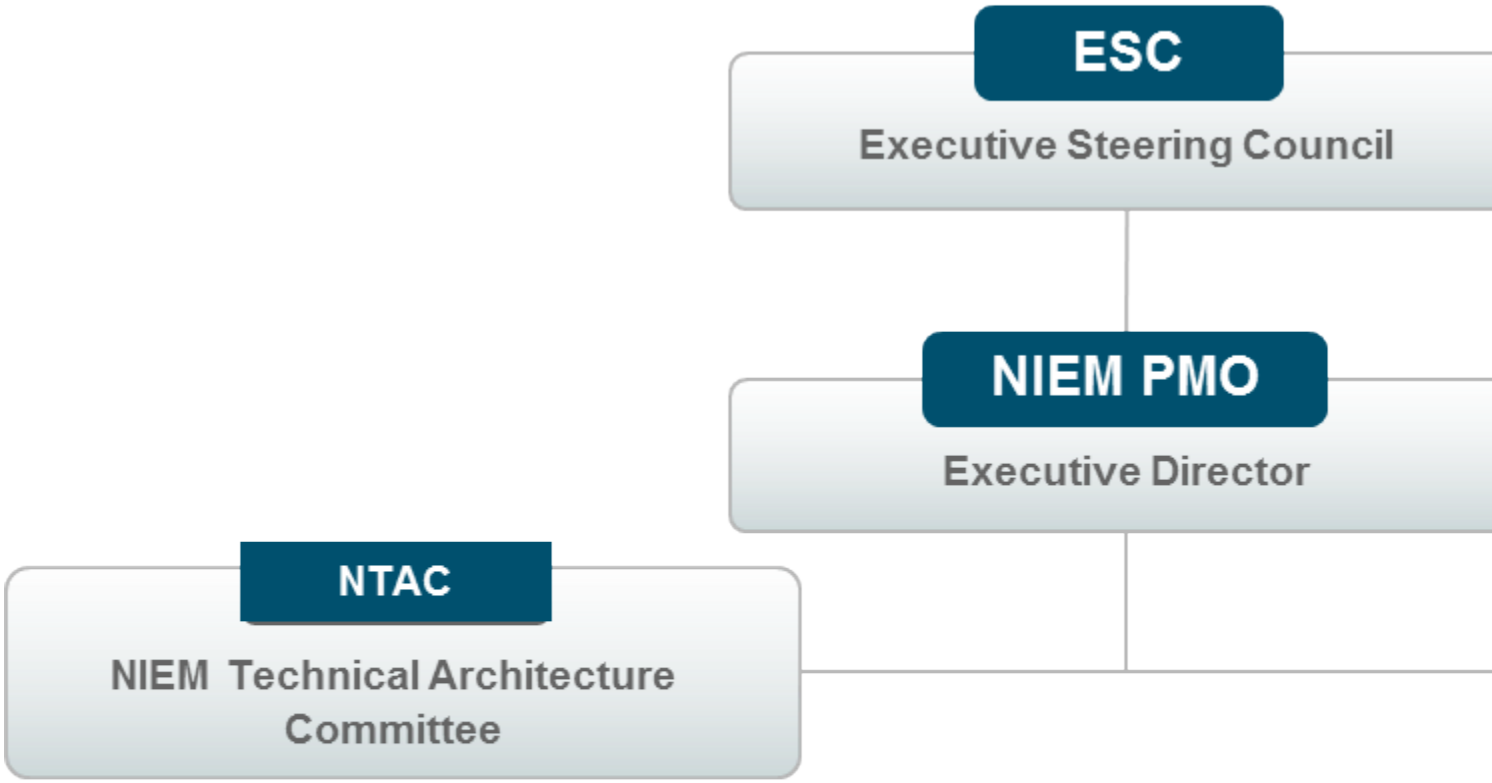
NIEM Governance

The NIEM data model is governed by the NIEM Domains and the NIEM Business Architecture Committee (NBAC). The purpose is to create a useful consensus model, with just the data elements that are likely to be used in many exchange specifications, and just as much harmonization as the community thinks worth doing. Domains are organized by subject area, and are open to all interested parties, with

participation from US federal, state, local, and tribal government, plus international and commercial members. The NBAC is comprised of domain representatives who govern the NIEM core model.

The NIEM Technical Architecture Committee (NTAC) defines and supports the technical architecture that governs NIEM; it documents, implements, and maintains NIEM's technical specifications.

Those bodies, the NTAC, the NBAC, and the Domains, comprise a voluntary consensus standards body for the NIEM standards. They operate according to the principles of openness, balance of interest, due process, an appeals process, and consensus, as defined in *ANSI Essential Requirements*.



There are also two administrative bodies above the level of standards development. The NIEM Executive Steering Council (ESC) is the decision-making body regarding membership, funding requirements, program or technical direction, committee, and Program Management Office (PMO) personnel appointments. The NIEM Program Management Office (PMO) executes the vision of NIEM established by the Executive Steering Council (ESC), while managing the day-to-day operations of NIEM. The office encourages the adoption and use of NIEM and oversees all working group and committee activities

Ways To Use And Participate In NIEM

At the simplest level, data scientists and software developers use a specification constructed through the NIEM process as documentation for a data resource they need to produce or consume. In this case the users need not be aware of NIEM at all. For example, the Automated Secure Alarm Protocol (ASAP) is an ANSI standard that is used for machine-to-machine communication data between alarm monitoring companies and 911 centers. Its data format is NIEM-conforming and was created through the NIEM process, but software developers building applications for the protocol do not have to know or care about that. They only have to produce or consume data that conforms to the specification.

The next level of NIEM use is to reuse NIEM components for one portion of a data resource. This treats the NIEM model as just another data vocabulary among others, such as Dublin Core and schema.org.

Scientists and developers may create their own NIEM-based specification to document a data resource they produce. In this case all of the data components will be NIEM-conforming, whether reused from the NIEM model, adapted from another standard, or created from scratch.

Several scientist/developer teams may form a community to create a data resource specification as the standard for data they all produce and consume. Or they may agree on a set of data components they will all use, while not attempting to standardize each complete resource. This is very similar to a NIEM domain's activity. The new community may seek recognition as a new domain, or affiliation with an existing domain, but is not required to do so.

Scientists and developers can participate in a NIEM domain's activities, proposing new and modified data elements in the domain submodel for use in multiple specifications. A representative from a NIEM domain will act as a voting member of the NIEM Business Architecture Committee (NBAC). The NBAC governs the NIEM core model; work here often involves harmonizing data definitions among the domain models to create elements that can be used in any domain. An individual may also be invited to participate in the NBAC as a non-voting observer.

Summary

This page describes those aspects of the NIEM organization, model, process, and technical specifications that are most relevant to NIEM JSON. Much more information about NIEM is available at the niem.gov website.

Why Use NIEM with JSON?

JSON data often appears as a simple, flat list of (key, value) pairs, as in the example on the left. JSON data can also be complex, with nested objects and arrays of objects or values, as in the example on the right.

```
{
  "lname": "Wimsey",
  "fname": "Peter",
  "date": "1893-05-04",
  "lat": 51.835,
  "lon": -0.417,
  "cdate": "1921-01-05",
  "charge": "Wild Driving"
}
```

```
{
  "name": {
    "lname": "Wimsey",
    "gname": [ "Peter","Death","Bredon" ],
  },
  "date": "1893-05-04,
  "lat": 51.835,
  "lon": -0.417,
  "cdate": "1921-01-05",
  "charge": "Wild Driving"
}
```

Regardless of whether the JSON data is simple or complex, the data scientists and software developers who work with it need to understand what it means. This involves answers to questions like:

- What is the meaning of the `date` key string? Birthday or something else?
- What is the format of the `date` value? Is it always an ISO 8601 date field, or might we sometimes see `Tuesday`, or `Fred`? Is it the same format as `cdate`, or something different?
- Does `lname` have the same meaning in both resources?
- Will `gname` always be found within a `name` object, or is it optional?

Such questions are easily answered when the JSON data is only exchanged between instances of the same program, or only processed by software written by a single small team of developers, or is one of a few sources accessed by the data scientists. In these cases there is little difficulty in understanding the JSON data, and little need for NIEM.

However, understanding the data resource becomes much more difficult when the developer team is large, or when there are many independent teams, or when the members of a team change over time – in short, when *developer cohesion* is low. Also, data understanding takes more work when scientists and developers work with many data resources (all at once, or over time) that have different data element definitions for the same shared concept (`fname`, `gname`, `givenname`) – in short, when *data resource overlap* is high. These conditions – low cohesion, high overlap – are almost always found in large enterprises with many participants or extensive data sharing needs.

NIEM is especially useful in those circumstances. The NIEM approach applied to JSON data offers:

- A mapping between each JSON key string and a global identifier for the corresponding data element.

- A framework for machine-readable documentation of each data element that is used in a data resource.

- A standards-based convention for canonical data element names used as JSON key strings (while still permitting local names like `lat` within a data resource).

These three combine to make NIEM JSON data *self-documenting*, at least to the extent that any data scientist or software developer can look at any key-value pair and find out how it is defined. In addition, the NIEM approach also offers:

- A simple way for developers who are creating a resource to reuse data elements defined by others, instead of creating new defintions. This encourages a degree of harmonization between different resources.

- A NIEM data model, partitioned into a NIEM core of concepts that might be reused in any exchange, plus several submodels extending the core, each defined by a subject-area or functional NIEM domain community. This also encourages harmonization among resources partly defined in terms of that model.

- A governance approach in which the NIEM core is controlled by the entire NIEM community, while each submodel is independently controlled by its domain, and which allows any community to create and control its own extension of the NIEM model.

- A version-control architecture that encourages reuse by insulating each community from external changes; this allows each community to change at its own pace, deciding if and when it will adopt changes made by others.

- A framework for data resource specifications that contain all the artifacts required to document the required and optional elements that may appear in a conforming resource, together with the meaning of those elements.

These features of NIEM combine to form a standards-based approach for constructing any number of data resource specifications, which document the data resources found at runtime and allow a compatible understanding of that data among the scientists and developers who work with it.

Although NIEM was first created to work with XML-based data exchange, it can now be applied to JSON data, without requiring knowledge of the XML technology originally used. Data exchange designers can use NIEM to create a resource specification that is all-JSON, or one that is all-XML, or one that defines both a JSON and an XML serialization, depending on the needs of the producers and consumers.

What Is NIEM JSON data?

A summary answer, one that is unpacked below, is:

1. It is JSON data. It follows the rules of the JSON specification.
2. It is NIEM data. It has a content model, which provides its meaning. That content model is based on the NIEM conceptual model.
3. It is JSON-LD data. It has a JSON-LD context, which connects key strings in the data to the data elements that provide their meaning.

NIEM JSON data might look like *Example 1* below, with simple *ad hoc* key strings:

```
{
  "person": {
    "name": {
      "lname": "Wimsey",
      "gname": [ "Peter","Death","Bredon" ]
    },
    "height": {
      "val": 163,
      "units": "CMT"
    },
    "hair": "BLD"
  }
}
```

The same information might also look like *Example 2* instead, with key strings that are names of NIEM data elements: (More on that information equivalence later.)

```
{
  "nc:Person": {
    "nc:PersonName": {
      "nc:PersonSurName": "Wimsey",
      "nc:PersonGivenName": [ "Peter","Death","Bredon" ]
    },
    "nc:PersonHeightMeasure": {
      "nc:MeasureIntegerValue": 163,
      "nc:LengthUnitCode": "CMT"
    },
    "j:PersonHairColorCode": "BLD"
  }
}
```

Informally, both of those JSON objects have the following meaning:

- There is a person
  - Who has a hair color that is *blond*
  - Who has a height that is *163 centimeters*

    - Who has a name
        - That has a surname of *Wimsey*
        - That has three given names: *Peter*, *Death*, and *Bredon*.

That meaning is supplied in three steps:

1. The NIEM conceptual model relates the components of NIEM JSON data to statements that make claims about the world.

2. The NIEM conceptual model also maps each key string in NIEM JSON data to a global identifier for a data element that defines its meaning and value domain.

3. The content model for each NIEM JSON data resource provides those data elements, and specifies which are mandatory and optional.

The NIEM Conceptual Model

According to the NIEM conceptual model, each object in the JSON data is

> a collection of statements that make claims about some object in 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.
> – *NIEM Naming and Design Rules (section 5)*

For each pair in the JSON object, the key string represents an object property, and the corresponding value is the value of that property. The order of the pairs is not significant. The values can be:

- A string. In the top example, the value of the `lname` property is the string "Wimsey".

- A number. In the top example, the value of the `val` property is the number 163.

- An object, like the value of the `name` property. This object has two properties of its own, `lname` and `fname`.

- An array, like the value of the `gname` property. Arrays are used when a property has more than one value, because keys may not be repeated in a NIEM JSON object. The `gname` property has three values: the strings "Peter", "Death", and "Bredon". The order of the array elements is not significant.

- The special tokens `true` and `false`.

- The special token `null` is not used in NIEM JSON data.

So far, so good. However, most of the meaning is bound up in the meaning of the key strings. The conceptual model does not supply that meaning. What the conceptual model does supply is a mapping between a name in the instance data and the URI for the data element with the meaning. For NIEM XML, element and attribute names are mapped to URIs through a rule based on XML namespace declarations. For NIEM JSON, the key strings are mapped to URIs through the context mechanism of

JSON-LD.

All NIEM JSON data is JSON-LD data, and follows the rules of the JSON-LD specification. In JSON-LD, a *context* is a JSON object with the reserved term `@context`as the only key string. It is used to map terms in key strings (and in certain data values) to IRIs. NIEM JSON always has a context that maps key strings to data element URIs. The context for *Example 2* is:

```
{
  "@context": {
    "nc": "http://release.niem.gov/niem/niem-core/4.0/#",
    "j":  "http://release.niem.gov/niem/domains/jxdm/6.0/#"
  }
}
```

Interpreted against that context, the key string `nc:PersonSurName` expands to the absolute URI:

`http://release.niem.gov/niem/niem-core/4.0/#PersonSurName`

Which is the global identifier for the data element defined as "A last name or family name of a person".

NIEM Content Models

NIEM data always has a content model, which includes all of the data elements that may appear in the data resource. These elements will be a subset of the NIEM core and domain models, possibly augmented by data elements defined specially for the resource. In addition to providing the meaning of each object property, the content model defines which elements are mandatory and optional in the resource; NIEM JSON uses JSON Schema for this.

The data elements for the content model of *Example 2* include:

Data Element Name	Definition
nc:Person	A human being
nc:PersonName	A combination of names and/or titles by which a person is known
nc:PersonSurName	A last name or family name of a person
nc:PersonGivenName	A first name of a person
nc:PersonHeightMeasure	A measurement of the height of a person
j:PersonHairColorCode	A color of the hair of a person

Every NIEM data element belongs to a namespace. Each namespace is uniquely identified by an absolute URI. Each namespace has a commonly-used *namespace prefix*, such as `nc`. The name of each data element is composed of the namespace prefix plus a *local name*, such as `PersonSurName`. The namespaces used in this example are:

Namespace	Prefix	Namespace URI
NIEM Core	nc	http://release.niem.gov/niem/niem-core/4.0/
Justice	j	http://release.niem.gov/niem/domains/jxdm/6.0/

The rule for the URIs in a NIEM JSON context is to append the octothorp/hash/pound "#" character to

any namespace URI that does not already end with that character. With this rule, the data element's local name becomes the *fragment identifier* for the machine-readable data element documentation within the resource identified by the namespace URI. The normative syntax for the namespace resource is XML Schema; however, equivalent alternate encodings in JSON Schema or RDF are possible.

Equivalence Of *Ad Hoc* Key Strings

NIEM JSON data will often use the names of NIEM data elements (such as `nc:PersonGiveName`) for key strings. However, sometimes people prefer simple *ad hoc* key strings such as `gname` instead. These simple keys may be used in NIEM JSON data if their mapping is included in the JSON-LD context. For example, the context for *Example 1* is:

```
"@context": {
  "nc":    "http://release.niem.gov/niem/niem-core/4.0/#",
  "j":     "http://release.niem.gov/niem/domains/jxdm/6.0/",
  "person": "nc:Person",
  "name":   "nc:PersonName",
  "lname":  "nc:PersonSurName",
  "gname":  "nc:PersonGivenName",
  "height": "nc:PersonHeightMeasure",
  "val":    "nc:MeasureIntegerValue",
  "units":  "nc:LengthUnitCode",
  "hair":   "j:PersonHairColorCode"
}
```

Using that context, the property with the key string `lname` is expanded to the URI:

```
http://release.niem.gov/niem/niem-core/4.0/#PersonSurName
```

Which is the URI for the data element definition. This approach allows the use of short *ad hoc* JSON keys while preserving the meaning provided by the NIEM conceptual model and the resource content model. Producers and consumers do not have to guess at the meaning of `lname`; it is precisely defined.

In addition, some development tooling do not support JSON names that include a colon (`:`), In these cases, a context can be used to define equivalent local names similar to the NIEM data elements that workaround the limitations of the tooling.

For example, the context could be:

```
"@context": {
  "nc":    "http://release.niem.gov/niem/niem-core/4.0/#",
  "j":     "http://release.niem.gov/niem/domains/jxdm/6.0/",
  "Person": "nc:Person",
  "PersonName":   "nc:PersonName",
  "PersonSurName":  "nc:PersonSurName",
  "PersonGivenName":  "nc:PersonGivenName",
  "PersonHeightMeasure": "nc:PersonHeightMeasure",
  "MeasureIntegerValue":    "nc:MeasureIntegerValue",
  "LengthUnitCode":  "nc:LengthUnitCode",
```

```
  "PersonHairColorCode":   "j:PersonHairColorCode"
}
```

This approach, however, does not handle exchanges in which the same name is used in multiple domains (e.g. `nc:Person` and `j:Person`). Another approach useful in these cases is to include domain prefixes in the local names with the colons replaced with an underscore (_) or other special character supported by the tooling (e.g. `nc_PersonName`)

Associating NIEM JSON Data To Its JSON-LD Context

NIEM JSON data always has an associated JSON-LD context. Sometimes the context object is embedded within the NIEM JSON data. For *Example 2*, the result would look like:

```
{
  "@context": {
    "nc":    "http://release.niem.gov/niem/niem-core/4.0/#",
    "j":     "http://release.niem.gov/niem/domains/jxdm/6.0/",
  }
  "nc:Person": {
    "nc:PersonName": {
      "nc:PersonSurName": "Wimsey",
      "nc:PersonGivenName": [ "Peter","Death","Bredon" ]
    },
    "nc:PersonHeightMeasure": {
      "nc:MeasureIntegerValue": 163,
      "nc:LengthUnitCode": "CMT"
    },
    "j:PersonHairColorCode": "BLD"
  }
}
```

Sometimes, NIEM JSON data simply includes the URI of its context resource. (This is especially useful when the context object is large.) With this approach, the data in *Example 2* might be:

```
{
  "@context": "http://example.com/myIEPD",
  "nc:Person": {
    "nc:PersonName": {
      "nc:PersonSurName": "Wimsey",
      "nc:PersonGivenName": [ "Peter","Death","Bredon" ]
    },
    "nc:PersonHeightMeasure": {
      "nc:MeasureIntegerValue": 163,
      "nc:LengthUnitCode": "CMT"
    },
    "j:PersonHairColorCode": "BLD"
  }
}
```

Where `http://example.com/myIEPD` is the URI for the context resource:

```
{
  "@context": {
    "nc": "http://release.niem.gov/niem/niem-core/4.0/#",
    "j":  "http://release.niem.gov/niem/domains/jxdm/6.0/#"
  }
}
```

The JSON-LD specification recommends that JSON-LD contexts be cached by systems that use them, reducing security risks and load times:

> JSON-LD contexts that are loaded from the Web over non-secure connections, such as HTTP, run the risk of being altered by an attacker such that they may modify the JSON-LD active context in a way that could compromise security. It is advised that any application that depends on a remote context for mission critical purposes vet and cache the remote context before allowing the system to use it.

This is also best practice for NIEM JSON. The context for NIEM JSON data should be treated as part of an information exchange specification, inspected at build time and retained for use, instead of relying on any kind of run-time network resolution of contexts.

Self-Describing Data

NIEM JSON is self-describing because:

- It is always associated with a JSON-LD context.
- That context maps each key string to a URI for a data element.
- The resource for that URI is machine-readable documentation of the data element's meaning.

This means that developers and data scientists can always tell whether a key in one resource has the same meaning as a key in another resource. It also means they can learn that meaning by obtaining the data element resource. These are valuable properties in any large enterprise with many different data resources.

A NIEM-JSON Reference Guide

Use of NIEM JSON does not require any use of XML; NIEM JSON exchanges are pure JSON, with JSON-LD providing some features of XML and JSON Schema providing some features of XML Schema. For users already familiar with XML, the NTAC produced Non-Normative Guidance in Using NIEM with JSON. This document lays out how a NIEM IEPD describes a JSON message, by showing that XML elements, attributes, and other features can be expressed as JSON objects. These guidelines highlight the use of JSON-LD for namespace-qualified names, the RDF conceptual model used by NIEM, and linked data methodologies.

The NIEM data model contains many community-agreed data definitions suitable for reuse. The names of JSON object properties can reuse the terms and namespaces of NIEM data definitions, allowing for reuse of data definitions across XML and JSON, and supporting JSON exchanges for data previously defined for XML.

NIEM has published a JSON-LD context for the NIEM 4.0 release, which provides a Uniform Resource Identifier (URI) mapping for each conformant namespace used in NIEM. This provides JSON users an easy way for JSON messages to use and reference the model, providing an easy-to-use linked data approach to NIEM.

Names for JSON data

An object within a JSON document is composed of keys with corresponding values, for example:

```
{
  "person" : {
    "age" : 14,
    "hair color" : "brown",
    "name" : "Morty Smith"
  }
}
```

In this example, each key is a string. When exchanging data in JSON, each participant needs to agree on basic facts about the data:

- What are the allowed names for values and objects within the JSON document?
- What do the various names mean?
- What are the allowed values for a given name?

NIEM defines a methodology for defining and reusing names for data.

NIEM data names follow a specific syntax, based on the ISO 11179 standard.

NIEM names are grouped by namespaces, distinguishing names that would otherwise be ambiguous or conflicting, and identifying names from specific versions of vocabularies.

Reuse of names and definitions

NIEM defines a methodology for broad communities to agree on:

- The structure of data
- Names for data
- Value constraints on data

Every name, structure, and set of data constraints has a single authoritative representation, defined by a set of XML Schema documents. See http://niem.github.io/niem-releases/ for new and old NIEM releases. These schemas define base definitions, which can be refined, specialized, and extended, to define specific exchanges.

NIEM has distributed governance, meaning that different sets of data definitions are defined by different groups. This enables groups to focus on their areas of interest, with deliberate version control, and minimal conflict across different groups.

NIEM defines the following kinds of data components:

- **Simple types** are the formats of unstructured values, like strings, numbers, and enumerations.

- **Complex types** are the structure of objects. Each complex type defines a class of object (such as a person or vehicle), along with a list set of characteristics it has, or relationships that it participates in.

- **Elements** describe uses of an object, or relationships that objects participate in, or characteristics of objects.

- **Attributes** describe characteristics of objects, with values that are always simple types.

Each type of component has its own purpose, and each supports a different aspect of data, including JSON data and XML data. However, every NIEM data component has a few common characteristics:

- A **local name**, which is a meaningful term that concisely identifies what a thing is, what it is for, and what it might hold. NIEM has a syntax for local names.

- A **definition**, which is a descriptive statement that provides a detailed meaning for a thing. The criteria of what makes a good definition are identified in the NIEM Naming and Design Rules, section 7.4.

- A **namespace**, which is a URL that acts as a universal identifier under which a set of data components are grouped. Each governance body, and each NIEM version, has unique namespaces that are assigned to it.

The **qualified name** of a data component is the **namespace** and **local name** together.

The reuse of these qualified names in exchange definitions, extensions, and messages allows the parts of the messages to be understood, and for similar components to be distinguished.

JSON, by itself, does not provide a mechanism for defining or using anything like XML namespaces. This is handled JSON-LD, which defines a way to use URIs for JSON keys, which works very similar to the way namespaces work.

Syntax of NIEM names

Each data component in NIEM has a single local name, which is paired with a namespace to form a universally unique qualified name.

Each local name follows a common structure. The main parts of a NIEM component name are:

- An **object class term** identifies a category of real-world things:
    - For a property (an element or attribute), the object class term identifies the category of thing that can have this characteristic, or which is the subject of this relationship.
    - For a type (simple or complex), the object class term identifies the category of things that the type represents.
- A **property term** describes a characteristic, relationship, or subpart of something.
- A **representation term** describes the nature of the value of an element or type. Representation terms in NIEM are listed in the Naming and Design Rules.

Any of the above terms may have any number of **qualifier terms**, which modify the term, to increase semantic precision or to reduce ambiguity. A qualifier term usually precedes the term it modifies.

For example, the parts of the NIEM element `nc:PersonPhysicalDisabilityText` are:

- `nc` - a **namespace prefix** (see NIEM Namespaces)
- `Person` - an **object class term**
- `Physical` - a **qualifier term**
- `Disability` - a **property term**
- `Text` - a **representation term**

Next: NIEM Namespaces

NIEM namespaces

NIEM maintains a set of governing bodies for data definitions. Each body is responsible for its own data definitions, and is encouraged to reuse the definitions of others. As described earlier, NIEM data definitions are organized into namespaces. These bodies, and their namespaces, include:

- The NIEM Business Architecture Committee (NBAC) governs the central namespace of NIEM, called NIEM Core. NIEM Core covers a large number of broadly-applicable data definitions which need to be managed across the entire NIEM community.

- The NIEM Technical Architecture Committee (NTAC) governs the technical architecture and manages the foundational namespaces **structures** and **appinfo**.

- Domains are communities of interest, each managing components addressing a specific topic or line of business.

Core and domains, and their namespaces, are outlined elsewhere on this site.

Each namespace has:

- an **XML namespace name**, which is a URI.
- a commonly-used **namespace prefix**, which is a short label that stands for the full namespace URI in instances and schemas.

Examples:

- NIEM core uses the prefix `nc`, and in NIEM 4.0 has the XML namespace name`http://release.niem.gov/niem/niem-core/4.0/`.

- The Justice domain uses the prefix `j`, and in NIEM 4.0 has the XML namespace name `http://release.niem.gov/niem/domains/jxdm/6.0/`.

The **namespace prefix** plus the **local name** for a component is the **qualified name** for a component. For example:

- The `PersonBirthDate` element defined by NIEM Core would be expressed as a qualified name, with the commonly-used prefix, as `nc:PersonBirthDate`.

- The `ChargeDisposition` element defined by the Justice domain would be expressed as a qualified name, with the commonly-used prefix, as `j:ChargeDisposition`.

JSON-LD

NIEM JSON documents are expressed using JSON-LD, a method of using JSON to express linked data, including Resource Description Framework (RDF) and Web Ontology Language (OWL) data. There are tools available that transform JSON-LD data in various ways.

JSON-LD is a pattern of using JSON that maps to RDF. NIEM has a mapping to RDF, as described by the NIEM Naming and Design Rules, section 5, *The NIEM conceptual model*. The expression of NIEM data in JSON-LD is, basically, the JSON-LD form of the RDF data that NIEM XML data would represent.

For most people, that's too much information, and it's sufficient to know that NIEM JSON data follows the structure and names of NIEM XML data.

JSON-LD allows a message to use short names as object keys, but still have those keys represent full URIs, which can represent the qualified names used in NIEM XML. The short names can have used XML qualified name syntax (e.g., `nc:Vehicle`), or can have a short syntax defined by an exchange developer (e.g., `vehicle`). These names are specified by a JSON-LD context, which is a JSON document (or part of a JSON-LD document).

JSON-LD provides a simple way of expressing the identifier of an object, using a construct called `@id`. These identifiers handle the identification and referencing that are handled by NIEM's `structures:id`, `structures:ref`, and `structures:uri` attributes.

JSON-LD contexts

JSON-LD provides **JSON-LD contexts** which enable JSON documents to use simple, locally-meaningful names, while still enabling the messages to be meaningful beyond the initial exchange partners. A JSON-LD message can be meaningful to anyone anywhere, even when mixed with other data.

The JSON-LD context has many more capabilities than this section will highlight. Here, we focus only on the use of contexts to define aliases, to enable the use of namespaces and short local terms.

Although you may not want to use colon-separated qualified names in your JSON data, it's easier to explain starting with that style. This example uses NIEM names with the standard prefixes for the NIEM namespaces:

```
{
  "nc:PersonAgeMeasure": {
    "nc:MeasureIntegerValue": 14,
    "nc:TimeUnitCode": "ANN"
  },
  "j:PersonHairColorCode": "BRO",
  "nc:PersonName": {
    "nc:PersonGivenName": "Mortimer",
    "nc:PersonSurName": "Smith",
    "nc:PersonNameSuffixText": "Sr",
    "nc:PersonPreferredName": "Morty"
  }
}
```

This may be interpreted against the following JSON-LD context:

```
{
  "nc": "http://release.niem.gov/niem/niem-core/4.0/#",
  "j": "http://release.niem.gov/niem/domains/jxdm/6.0/#"
}
```

This example and expanded and compacted version can be seen at the JSON-LD Playground.

The page on JSON-LD processing discusses expansion and compaction, two of the standard JSON-LD processing algorithms that can transform JSON-LD data in useful ways.

Compaction

When compacted with an empty context, the structure of the data, and the complete URIs for keys are plainly visible:

```
{
  "http://release.niem.gov/niem/domains/jxdm/6.0/#PersonHairColorCode": "BRO",
  "http://release.niem.gov/niem/niem-core/4.0/#PersonAgeMeasure": {
```

```
    "http://release.niem.gov/niem/niem-core/4.0/#MeasureIntegerValue": 14,
    "http://release.niem.gov/niem/niem-core/4.0/#TimeUnitCode": "ANN"
  },
  "http://release.niem.gov/niem/niem-core/4.0/#PersonName": {
    "http://release.niem.gov/niem/niem-core/4.0/#PersonGivenName": "Mortimer",
    "http://release.niem.gov/niem/niem-core/4.0/#PersonNameSuffixText": "Sr",
    "http://release.niem.gov/niem/niem-core/4.0/#PersonPreferredName": "Morty"
    "http://release.niem.gov/niem/niem-core/4.0/#PersonSurName": "Smith"
  }
}
```

This example can be seen at [the JSON-LD Playground](#).

Again, expanded and otherwise processed data shows the fundamental meaning of data in a universally-understandable way. It's likely that exchanges will instead utilize shorter, locally-preferred terminology.

The above data can be compacted against another JSON-LD context:

```
{
  "nc": "http://release.niem.gov/niem/niem-core/4.0/#",
  "j": "http://release.niem.gov/niem/domains/jxdm/6.0/#",
  "age": "nc:PersonAgeMeasure",
  "value": "nc:MeasureIntegerValue",
  "units": "nc:TimeUnitCode",
  "hairColor": "j:PersonHairColorCode",
  "name": "nc:PersonName",
  "given": "nc:PersonGivenName",
  "surname": "nc:PersonSurName",
  "suffix": "nc:PersonNameSuffixText",
  "nickname": "nc:PersonPreferredName"
}
```

This yields the JSON-LD data that uses only short, local terms:

```
{
  "age": {
    "value": 14,
    "units": "ANN"
  },
  "hairColor": "BRO",
  "name": {
    "given": "Mortimer",
    "surname": "Smith",
    "suffix": "Sr",
    "nickname": "Morty"
  }
}
```

This example can be seen at [the JSON-LD Playground](#).

When interpreted against the right JSON-LD contexts, both JSON documents evaluate to the same expanded data.

Developers of exchanges may use contexts to greatly simplify their JSON files, while maintaining the meaning of the data across all users and communities.

Using JSON-LD contexts to define keys

JSON-LD contexts can define universal identifiers for keys several different ways. This section highlights a few. The context can do much more than define key URIs; techniques are detailed in the JSON-LD specification's sections on contexts and advanced concepts.

Compact IRIs: mapping qualified names to IRIs/URIs

Contexts use Internationalized Resource Identifiers (IRIs) to uniquely identify resources.

The use of contexts to define prefixes is described by the JSON-LD specification's section on compact IRIs, which is JSON-LD's name for colon-separated qualified names.

The proper value to use for a prefix defined in a context can be identified by the NIEM Naming and Design Rules, section 5.6.1, *Resource IRIs for XML Schema components and information items*, which describes the mapping from an XML qualified name to a URI. In short, any namespace that does not end in an octothorp/hash/pound sign ("#") has a "#" character appended to it. So, since the namespace URI for the NIEM Core namespace (nc) is `http://release.niem.gov/niem/niem-core/4.0/`, it would be defined in the following part of a context:

```
{
  "nc": "http://release.niem.gov/niem/niem-core/4.0/#"
}
```

With this context, a value `nc:PersonAgeMeasure` becomes, after processing, the IRI `http://release.niem.gov/niem/niem-core/4.0/#PersonAgeMeasure`.

Mapping local names to IRIs

Within a context, a local name, like `age` can be aliased to an IRI. Either of the two following contexts would suffice:

```
{
  "nc": "http://release.niem.gov/niem/niem-core/4.0/#",
  "age": "nc:PersonAgeMeasure"
}
```

```
{
  "age": "http://release.niem.gov/niem/niem-core/4.0/#PersonAgeMeasure"
}
```

Use of `@vocab` to handle uniform namespaces

If much of your data is from a single namespace, the `@vocab` directive within a context will map otherwise-undefined local names to a single namespace, as if each was given an assigned prefix.

The context, …

```
{
  "@vocab": "http://release.niem.gov/niem/niem-core/4.0/#",
  "j": "http://release.niem.gov/niem/domains/jxdm/6.0/#"
}
```

… used to process the following instance, results in the same data as the above example.

```
{
  "PersonAgeMeasure": {
    "MeasureIntegerValue": 14,
    "TimeUnitCode": "ANN"
  },
  "j:PersonHairColorCode": "BRO",
  "PersonName": {
    "PersonGivenName": "Mortimer",
    "PersonSurName": "Smith",
    "PersonNameSuffixText": "Sr",
    "PersonPreferredName": "Morty"
  }
}
```

This example is available at [the JSON-LD Playground](#).

Linking a JSON file to a JSON-LD context

The [JSON-LD specification](#) describes how to attach a JSON-LD context to a JSON file, so that the file may be properly processed as JSON-LD. Examples provided on this website *mostly* maintain separate contexts from the JSON files.

Use `@context`

Any JSON object may carry a key "`@context`", which indicates a context that is active at the scope of that object, and within to nested objects.

Provide a context inline

A key "`@context`" may have, as a value, a JSON object, which acts as a context.

```
{
  "@context" : {
```

```
    "nc": "http://release.niem.gov/niem/niem-core/4.0/#"
  },
  "nc:PersonPreferredName": "Morty"
}
```

Link to a context

A key "`@context`" may have, as a value, a URI, which is a name for a JSON-LD context object.

```
{
  "@context" : "https://example.org/my/context.json",
  "nc:PersonPreferredName": "Morty"
}
```

Provide a context in an HTTP Link header

A method for attaching a JSON context to a plain JSON object via an HTTP Link header is described by the JSON-LD specification, section 4.9, *Interpreting JSON as JSON-LD*. In short, a plain JSON file that is passed via HTTP can be accompanied by a separate JSON-LD context document. The URI of the JSON-LD context document is passed via an HTTP Link header, using the link relation `http://www.w3.org/ns/json-ld#context`.

Cache contexts

The JSON-LD specification recommends that JSON-LD contexts be cached by systems that use them, reducing security risks and load times:

> JSON-LD contexts that are loaded from the Web over non-secure connections, such as HTTP, run the risk of being altered by an attacker such that they may modify the JSON-LD active context in a way that could compromise security. It is advised that any application that depends on a remote context for mission critical purposes vet and cache the remote context before allowing the system to use it.

A best practice for using JSON-LD contexts with NIEM is to treat contexts as part of an information exchange specification, and inspect them, vet them, and cache them for use, rather than relying on any kind of run-time network resolution of contexts.

JSON-LD identifiers

One of the ways NIEM supports data reuse is through object references which eliminate the need to describe the same object multiple times. JSON-LD supports references to objects with a reserved key `@id` which can be used either as an object identifier or a reference to an object identifier. The value of `@id` must be a string.

For example, a person object may described and identified as “P01” as follows:

```
{
  "nc:Person" : {
    "@id" : "P01",
    "nc:PersonBirthDate": {
      "nc:Date": "1893-05-04"
    },
    "nc:PersonName": {
      "nc:personNameCommentText" : "copied",
      ...
    },
    "exch:PersonFictionalCharacterIndicator": true
  }
}
```

An role object referencing that person object would be:

```
{
  "nc:RoleOfPerson" : {
    "@id" : "P01"
  }
}
```

An example of JSON-LD expansion

This example is available at the JSON-LD Playground.

When the JSON data…

```
{
  "nc:PersonAgeMeasure": {
    "nc:MeasureIntegerValue": 14,
    "nc:TimeUnitCode": "ANN"
  },
  "j:PersonHairColorCode": "BRO",
  "nc:PersonName": {
    "nc:PersonGivenName": "Mortimer",
    "nc:PersonSurName": "Smith",
    "nc:PersonNameSuffixText": "Sr",
    "nc:PersonPreferredName": "Morty"
  }
}
```

…is processed using the JSON-LD expansion algorithm, against the following JSON context…

```
{
  "nc": "http://release.niem.gov/niem/niem-core/4.0/#",
  "j": "http://release.niem.gov/niem/domains/jxdm/6.0/#"
}
```

…it yields the expanded JSON data…

```
[
  {
    "http://release.niem.gov/niem/domains/jxdm/6.0/#PersonHairColorCode": [
      {
        "@value": "BRO"
      }
    ],
    "http://release.niem.gov/niem/niem-core/4.0/#PersonAgeMeasure": [
      {
        "http://release.niem.gov/niem/niem-core/4.0/#MeasureIntegerValue": [
          {
            "@value": 14
          }
        ],
        "http://release.niem.gov/niem/niem-core/4.0/#TimeUnitCode": [
          {
            "@value": "ANN"
          }
        ]
      }
```

```
    ],
    "http://release.niem.gov/niem/niem-core/4.0/#PersonName": [
      {
        "http://release.niem.gov/niem/niem-core/4.0/#PersonGivenName": [
          {
            "@value": "Mortimer"
          }
        ],
        "http://release.niem.gov/niem/niem-core/4.0/#PersonNameSuffixText": [
          {
            "@value": "Sr"
          }
        ],
        "http://release.niem.gov/niem/niem-core/4.0/#PersonPreferredName": [
          {
            "@value": "Morty"
          }
        ],
        "http://release.niem.gov/niem/niem-core/4.0/#PersonSurName": [
          {
            "@value": "Smith"
          }
        ]
      }
    ]
  }
]
```

JSON-LD processing

JSON-LD provides a specification, JSON-LD 1.1 Processing Algorithms and API, whic provides standard operations for processing JSON-LD. These operations allow JSON data to be transformed to various formats for different purposes, and enables the meaning of data that uses local terminology to be widely comprehended.

The specification describes operations for expansion, compaction, flattening, and converting to or from RDF. This page highlights only the expansion and compaction algorithms; see the specification for information about the other operations.

Expansion

The expansion algorithm removes JSON-LD contexts from a message, cooking any alises defined in the context directly into the data. In addition, anywhere an array could occur, it places an array. An overview is provided in section 2.1, Expansion.

For example:

```
{
  "@context": {
    "nc": "http://release.niem.gov/niem/niem-core/4.0/#"
  },
  "nc:PersonPreferredName": "Morty"
}
```

This data, processed by the expansion algorithm, is expanded to:

```
[
  {
    "http://release.niem.gov/niem/niem-core/4.0/#PersonPreferredName": [
      {
        "@value": "Morty"
      }
    ]
  }
]
```

Compaction

The compaction operation is the inverse of the expansion algorithm, encoding JSON-LD data based on a given context. For example, given the expanded data above, compacted against the context:

```
{
    "nc": "http://release.niem.gov/niem/niem-core/4.0/#"
}
```

…yields the JSON data:

```
{
  "@context": {
    "nc": "http://release.niem.gov/niem/niem-core/4.0/#"
  },
  "nc:PersonPreferredName": "Morty"
}
```

When compacted against an empty context:

```
{
}
```

…it yields JSON data that does not depend on any context, but which has a simplified form:

```
{
  "http://release.niem.gov/niem/niem-core/4.0/#PersonPreferredName": "Morty"
}
```

The data could be compacted against a context that provides a short local name for the term:

```
{
  "nick": "http://release.niem.gov/niem/niem-core/4.0/#PersonPreferredName"
}
```

…yielding compacted data that uses the short term:

```
{
  "nick": "Morty"
}
```

NIEM Content Models

Content models define the structure and content of data objects in a formal way so that the content of information exchanges can be understood, used and validated in a consistent and repeatable way.

The structure and content of data objects is defined in **schema** which are defined when an information exchange is designed (or redesigned).

The messages in an information exchange contain objects that are **instances** generated from the **schema** by the sender of the message. At any time, if needed, **instances** may be validated against the **schema** by either the sender or receiver of the message.

In NIEM information exchanges, **instances** are referred to as **Information Exchange Packages (IEPs)** and **schemas** are essential artifacts of an **Information Exchange Package Documentation (IEPD)** that describes an information exchange. Other artifacts in an **IEPD** typically include sample **instances**, use cases, business rules, and other business and technical documentation guiding the implementation of the information exchange. Each **IEPD** should also include metadata about the exchange including the creator, the current version and the change history.

NIEM supports JSON information exchanges including **IEPs** in JSON format and **IEPDs** that include JSON Schema

More detail on NIEM models and their implementations in XML and JSON, including NIEM concepts, content, and releases, is available.

JSON Schema

JSON schema provides a JSON vocabulary for defining and validating JSON **instances**.

Each schema includes one or more key-value pairs or **keywords**, including a `type` keyword set to a JSON simple value (`null`, `boolean`, `number`, `integer` or `string`), `array` or `object` as defined in JSON data:

All schemas may include the following keywords:

- A`description` keyword to provide a definition for this type or property.
- An `$id` keyword to provide a base URL for referencing this schema, type or property.
- A `$ref` keyword to provide a JSON pointer to another schema, type or property,

Other keywords are specific to the `type` as follows:

- Numbers and integers may be restricted with the following keywords:
  - `multipleOf`: limit the value to be a multiple of a given number
  - `minimum`,`exclusiveMinimum`, `maximum`,`exclusiveMaximum`: restrict the range of values
- Strings may be restricted with the following keywords:
  - `minLength`,`maxLength`: limit the length of the string
  - `pattern`: require the value to satisfy a regular expression
  - `format`: require the value to match a certain format (`e.g. date-time`, `email`, `ipv4`, `uri`)
- Arrays may contain one or more `items` restricted with the following keywords:
  - `type`: limit the type of each item
  - `enum`: define the allowable values of each item with an array
  - `minItems`, `maxItems`: limit the number or items
  - `uniqueItems`: require each item to be unique
- Objects may contain one or more **properties** restricted with the following keywords:
  - `properties`: list properties that MAY be included in the object
  - `required`: list properties that MUST be included in the object
  - `additionalProperties`: allow properties that are not listed in the type definition
  - `minProperties`, `maxProperties`: limit the number of properties in the object
  - `patternProperties`: define types for properties based on their name
  - `dependencies`: add restrictions if certain conditions are met

In addition, schemas may be combined with the following keywords:

- `allOf`: must be valid against *all* of the subschemas
- `anyOf`: must be valid against *any* of the subschemas
- `oneOf`: must be valid against *exactly one* of the subschemas

An simple example of a JSON schema to request a court hearing is:

```
{
```

```
"$id": "https://example.com/request",
"$schema": "http://json-schema.org/draft-04/schema#",
"type": "object",
"additionalProperties": false,
"definitions": {
  "request:RequestMessage": {
    "$ref": "#/definitions/request:RequestMessageType",
    "description": "A request for the schedule of upcoming events in a court
  },
  "request:RequestMessageType": {
    "additionalProperties": false,
    "description": "A request for the schedule of upcoming events in a court
    "properties": {
      "request:CaseTypeCode": {
        "description": "A certain case type.",
        "type": "string",
        "enum": ["civil", "criminal", "traffic"]
      },
      "request:CourtID": {
        "description": "The identifier for a court of law in which the case
        "type": "integer",
        "minimum": 1,
        "maximum": 9
      },
      "request:CourtName": {
        "description": "A court of law in which the case is being tried.",
        "type": "string"
      },
      "request:HearingDate": {
        "description": "The requested date of the hearing.",
        "type": "string",
        "format": "date"
      },
      "request:InitialApperanceIndicator": {
        "description": "Indicates whether this will be an initial appearance
        "type": "boolean"
      }
    },
    "required": [
      "request:CaseTypeCode",
      "request:CourtID",
      "request:CourtName",
      "request:HearingDate",
      "request:InitialApperanceIndicator"
    ],
    "type": "object"
  }
},
"properties": {
  "request:RequestMessage": {
```

```
      "$ref": "#/definitions/request:RequestMessageType",
      "description": "A request for the schedule of upcoming events in a court
    },
    "@context": { }
  },
  "required": [
    "@context",
    "request:RequestMessage"
  ]
}
```

An example of a JSON request based on the schema is:

```
{
  "@context": {
    "request": "https://example.com/request#"
  },
  "request:RequestMessage": {
    "request:CaseTypeCode": "civil",
    "request:CourtID": 3,
    "request:CourtName": "Springfield Circuirt Court",
    "request:HearingDate": "2019-01-02",
    "request:InitialApperanceIndicator": true
  }
}
```

A complete NIEM-conforming example of a JSON court scheduling exchange adapted from the LegalXML Electronic Court Filing (ECF) 5.0 specification is also available

- JSON References
- Example JSON Schema

JSON References

In a JSON schema, a `$ref` keyword is a JSON Pointer to a schema, or a type or property in a schema.

A JSON pointer takes the form of *A*#*B* in which:

- *A* is the relative path from the current schema to a target schema. If *A* is empty, the reference is to a type or property in the same schema, an **in-schema reference**. Otherwise, the reference is to a different schema, a **cross-schema reference**.
- *B* is the complete path from the root of the schema to a type or property in the schema. If # in not included or *B* is empty, the reference is to an entire schema.

Both *A* and *B* consist of successive components of the path (folder, type or property names) each separated by a `/`.

To demonstrate, here is a example JSON schema `xs.schema.json` that will be the target of **cross-schema references**:

```
{
  "$id": "http://www.w3.org/2001/XMLSchema",
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "additionalProperties": false,
  "definitions": {
    "xs:decimal": {
      "type": "number"
    },
    "xs:string": {
      "type": "string"
    },
    "xs:token": {
      "type": "string",
      "pattern": "^\\S*$"
    }
  },
  "properties": {
  },
  "required": [
  ]
}
```

Here is an example JSON schema `ns.schema.json` including **cross-schema references** to types in `xs.schema.json`, a schema in the same directory, and **in-schema references** to types and properties in the same schema:

```
{
  "$id": "http://release.niem.gov/niem/niem-core/4.0/",
  "$schema": "http://json-schema.org/draft-04/schema#",
```

```
  "type": "object",
  "additionalProperties": false,
  "definitions": {
    "nc:Amount": {
      "$ref": "./xs.schema.json#/definitions/xs:decimal",
      "description": "An amount of money."
    },
    "nc:AmountType": {
      "additionalProperties": false,
      "description": "A data type for an amount of money.",
      "properties": {
        "nc:Amount": {
          "description": "An amount of money.",
          "$ref": "#/definitions/nc:Amount"
        },
        "nc:CurrencyCode": {
          "description": "A unit of money or exchange.",
          "$ref": "#/definitions/nc:CurrencyCode"
        },
        "nc:CurrencyText": {
          "description": "A unit of money or exchange.",
          "$ref": "#/definitions/nc:CurrencyText"
        }
      }
    },
    "nc:CurrencyCode": {
      "$ref": "./xs.schema.json#/definitions/xs:token",
      "description": "A unit of money or exchange."
    },
    "nc:CurrencyText": {
      "$ref": "./xs.schema.json#/definitions/xs:string",
      "description": "A unit of money or exchange."
    }
  },
  "properties": {
    "nc:ItemValueAmount": {
      "$ref": "#/definitions/nc:AmountType",
      "description": "A monetary value of an item."
    }
  },
  "required": [
  ]
}
```

Example JSON Schema

The OASIS LegalXML Electronic Court Filing (ECF) 5.0 specification defines a number of NIEM 4.0 information exchanges with courts.

Here is the ECF 5.0 GetCourtSchedulingRequestMessage to request a court hearing implemented in JSON schema:

```
{
  "$id": "https://docs.oasis-open.org/legalxml-courtfiling/ns/v5.0/schedulerec
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "additionalProperties": false,
  "definitions": {
    "schedulerequest:GetCourtScheduleRequestMessage": {
      "$ref": "#/definitions/schedulerequest:GetCourtScheduleRequestMessageTyp
      "description": "A request for the schedule of upcoming events in a court
    },
    "schedulerequest:GetCourtScheduleRequestMessageType": {
      "additionalProperties": false,
      "description": "A request for the schedule of upcoming events in a court
      "properties": {
        "ecf:CaseTypeCode": {
          "description": "Indicates whether the e-filing system supports elect
          "$ref": "./ecf.schema.json#/definitions/ecf:CaseTypeCode"
        },
        "ecf:CourtEventTypeCode": {
          "description": "Filter criterion indicating that only calendar entri
          "$ref": "./ecf.schema.json#/definitions/ecf:CourtEventTypeCode"
        },
        "ecf:DocumentAugmentation": {
          "description": "The pleading, motion or order that is the main docum
          "items": {
            "$ref": "./ecf.schema.json#/definitions/ecf:DocumentAugmentation"
          },
          "minItems": 0,
          "type": "array"
        },
        "ecf:ParticipantID": {
          "description": "A unique identifier for an entity participating in a
          "$ref": "./ecf.schema.json#/definitions/ecf:ParticipantID"
        },
        "ecf:SendingMDELocationID": {
          "description": "Location for the MDE to which asynchronous and servi
          "$ref": "./ecf.schema.json#/definitions/ecf:SendingMDELocationID"
        },
        "ecf:ServiceInteractionProfileCode": {
          "description": "An identifier, from a list of allowed values defined
          "$ref": "./ecf.schema.json#/definitions/ecf:ServiceInteractionProfil
```

```
    },
    "j:CaseCourt": {
      "description": "A court of law in which the case is being tried.",
      "$ref": "./niem/j.schema.json#/definitions/j:CaseCourt"
    },
    "j:CourtAdministrativeUnitText": {
      "description": "A unit within a court system responsible for record
      "$ref": "./niem/j.schema.json#/definitions/j:CourtAdministrativeUnit
    },
    "j:Judge": {
      "description": "An official who hears and decides a case or who rule
      "$ref": "./niem/j.schema.json#/definitions/j:Judge"
    },
    "nc:DateRange": {
      "description": "A time period measured by a starting and ending poir
      "$ref": "./niem/nc.schema.json#/definitions/nc:DateRange"
    },
    "nc:DocumentCategoryText": {
      "description": "A kind of document; a nature or genre of the content
      "items": {
        "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentCategoryTex
      },
      "minItems": 0,
      "type": "array"
    },
    "nc:DocumentDescriptionText": {
      "description": "A description of the content of a document.",
      "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentDescriptionTe
    },
    "nc:DocumentEffectiveDate": {
      "description": "A date in which the content or action becomes enforc
      "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentEffectiveDate
    },
    "nc:DocumentFileControlID": {
      "description": "An identifier assigned to a document to locate it wi
      "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentFileControlID
    },
    "nc:DocumentFiledDate": {
      "description": "A date a document was officially filed with an organ
      "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentFiledDate"
    },
    "nc:DocumentIdentification": {
      "description": "An identification that references a document.",
      "items": {
        "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentIdentificat
      },
      "minItems": 1,
      "type": "array"
    },
    "nc:DocumentInformationCutOffDate": {
```

```
        "description": "A date after which contributions to the content of a
        "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentInformationCu
      },
      "nc:DocumentLanguageCode": {
        "description": "A language of the content of the resource.",
        "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentLanguageCode"
      },
      "nc:DocumentPostDate": {
        "description": "A date a document is entered or posted to an informa
        "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentPostDate"
      },
      "nc:DocumentReceivedDate": {
        "description": "A date a transmitted document was received.",
        "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentReceivedDate"
      },
      "nc:DocumentSequenceID": {
        "description": "An identifier that determines the document order in
        "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentSequenceID"
      },
      "nc:DocumentSoftwareName": {
        "description": "A name of a computer application used to process a c
        "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentSoftwareName"
      },
      "nc:DocumentStatus": {
        "description": "A status of a document.",
        "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentStatus"
      },
      "nc:DocumentSubmitter": {
        "description": "An entity responsible for making the resource availa
        "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentSubmitter"
      },
      "nc:DocumentTitleText": {
        "description": "A name given to a document.",
        "$ref": "./niem/nc.schema.json#/definitions/nc:DocumentTitleText"
      }
    },
    "required": [
      "ecf:ParticipantID",
      "ecf:SendingMDELocationID",
      "ecf:ServiceInteractionProfileCode",
      "j:CaseCourt",
      "nc:DocumentIdentification",
      "nc:DocumentPostDate"
    ],
    "type": "object"
  }
},
"properties": {
  "schedulerequest:GetCourtScheduleRequestMessage": {
    "$ref": "#/definitions/schedulerequest:GetCourtScheduleRequestMessageTyp
```

```
      "description": "A request for the schedule of upcoming events in a court
    }
  },
  "required": [
    "schedulerequest:GetCourtScheduleRequestMessage"
  ]
}
```


An example of a JSON request based on the JSON schema is:


```
{
  "@context": {
    "clsa": "http://reference.niem.gov/niem/specification/code-lists/1.0/code-
    "ct": "http://release.niem.gov/niem/conformanceTargets/3.0/#",
    "ecf": "https://docs.oasis-open.org/legalxml-courtfiling/ns/v5.0/ecf#",
    "j": "http://release.niem.gov/niem/domains/jxdm/6.0/#",
    "nc": "http://release.niem.gov/niem/niem-core/4.0/#",
    "schedulerequest": "https://docs.oasis-open.org/legalxml-courtfiling/ns/v5
    "structures": "http://release.niem.gov/niem/structures/4.0/#",
    "term": "http://release.niem.gov/niem/localTerminology/3.0/#",
    "xs": "http://www.w3.org/2001/XMLSchema#"
  },
  "schedulerequest:GetCourtScheduleRequestMessage": {
    "ecf:ParticipantID": {
      "nc:IdentificationID": "10"
    },
    "ecf:SendingMDELocationID": {
      "nc:IdentificationID": "https://efilingprovider.com:8000"
    },
    "ecf:ServiceInteractionProfileCode": "urn:oasis:names:tc:legalxml-courtfil
    "j:CaseCourt": {
      "nc:OrganizationIdentification": {
        "nc:IdentificationID": "10"
      },
      "j:CourtName": "Circuit Court"
    },
    "j:Judge": {
      "nc:RoleOfPerson": {
        "nc:PersonName": {
          "nc:PersonFullName": "John Smith"
        }
      },
      "j:JudicialOfficialRegistrationIdentification": {
        "nc:IdentificationID": "100"
      }
    },
    "ecf:CaseTypeCode": "civil",
    "ecf:CourtEventTypeCode": "hearing",
    "nc:DocumentIdentification": [
      {
```

```
        "nc:IdentificationID": "001-f786e6d6-558c-448f-b249-067002d716a0"
      }
    ],
    "nc:DocumentPostDate": {
      "nc:Date": "2018-12-01"
    },
    "nc:DateRange": {
      "nc:StartDate": {
        "nc:Date": "2019-01-01"
      },
      "nc:EndDate": {
        "nc:Date": "2019-01-03"
      }
    }
  }
}
```

Advanced NIEM Concepts

In addition to objects and properties, NIEM provides a few other ways to define the contents of a data exchange including:

- Associations - relationships between objects
- Augmentations - extensions (additional properties) added to an object
- Metadata - information about objects or properties (e.g. the source or security level of the information).
- Roles - Temporary functions played by objects (e.g. the person that is the subject of a case).

Associations

Associations are relationships between objects. Each association references the associated objects with the `@id` keyword from JSON-LD.

Here is an example of a NIEM association `j:OffenseLocationAssociation` in JSON schema:

```
{
  "$id": "http://release.niem.gov/niem/domains/jxdm/6.0/",
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "additionalProperties": false,
  "definitions": {
    "j:Offense": {
      "$ref": "#/definitions/j:OffenseType",
      "description": "An act or a course of action which may constitute a viol
    },
    "j:OffenseLocationAssociation": {
      "$ref": "#/definitions/j:OffenseLocationAssociationType",
      "description": "A relationship between an offense and a location at whic
    },
    "j:OffenseLocationAssociationType": {
      "additionalProperties": false,
      "description": "A data type for a relationship between an offense and a
      "properties": {
        "j:Offense": {
          "description": "An act or a course of action which may constitute a
          "$ref": "#/definitions/j:Offense"
        },
        "nc:AssociationDescriptionText": {
          "description": "A description of an association.",
          "$ref": "./nc.schema.json#/definitions/nc:AssociationDescriptionText
        },
        "nc:Location": {
          "description": "A geospatial location.",
          "$ref": "./nc.schema.json#/definitions/nc:Location"
        }
      },
      "type": "object"
    },
    "j:OffenseType": {
      "additionalProperties": false,
      "description": "A data type for an alleged violation of a statute, ordin
      "properties": {
        "nc:ActivityDate": {
          "description": "A date of an activity.",
          "$ref": "./nc.schema.json#/definitions/nc:ActivityDate"
        },
```

```
        "nc:ActivityDescriptionText": {
          "description": "A description of an activity.",
          "$ref": "./nc.schema.json#/definitions/nc:ActivityDescriptionText"
        }
      },
      "type": "object"
    }
  },
  "properties": {
    "j:Offense": {
      "$ref": "#/definitions/j:Offense",
      "description": "An act or a course of action which may constitute a viol
    },
    "nc:Location": {
      "$ref": "./nc.schema.json#/definitions/nc:LocationType",
      "description": "A geospatial location."
    },
    "j:OffenseLocationAssociation": {
      "$ref": "#/definitions/j:OffenseLocationAssociationType",
      "description": "A relationship between an offense and a location at whic
    }
  },
  "required": [
  ]
}
```

This schema references 2 other schemas: nc and xs

Here is a JSON instance of that association:

```
{
  "@context": {
    "j": "http://release.niem.gov/niem/domains/jxdm/6.0/#",
    "nc": "http://release.niem.gov/niem/niem-core/4.0/#",
    "xs": "http://www.w3.org/2001/XMLSchema#"
  },
  "j:Offense": {
    "@id": "Offense1",
    "nc:ActivityDate": {
      "nc:DateTime": "2018-01-01"
    },
    "nc:ActivityDescriptionText": "Vandalism"
  },
  "nc:Location": {
    "@id": "Location1",
    "nc:Address": {
      "nc:AddressFullText": "100 Main St.",
      "nc:LocationCityName": "Springfield",
      "nc:LocationStateName": "Illinois"
    },
```

```
    "nc:LocationDescriptionText": "Residence"
  },
  "j:OffenseLocationAssociation": {
    "j:Offense": {
      "@id": "Offense1"
    },
    "nc:Location": {
      "@id": "Location1"
    },
    "nc:AssociationDescriptionText": "The location the offense occured"
  }
}
```

Augmentations

Augmentations are extensions (collections of additional properties) added to an object. In JSON schema, you must include the augmentation as a property in the definition of the object it extends.

Here is an example of a NIEM augmentation `j:CaseAugmentation` in JSON schema:

```
{
  "$id": "http://release.niem.gov/niem/domains/jxdm/6.0/",
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "additionalProperties": false,
  "definitions": {
    "j:CaseAugmentation": {
      "$ref": "#/definitions/j:CaseAugmentationType",
      "description": "Additional information about a case."
    },
    "j:CaseAugmentationType": {
      "additionalProperties": false,
      "description": "A data type for additional information about a case.",
      "properties": {
        "j:CaseCourt": {
          "description": "A court of law in which the case is being tried.",
          "$ref": "#/definitions/j:CaseCourt"
        },
        "j:CaseLineageCase": {
          "description": "A description of a case at a previous stage.",
          "$ref": "#/definitions/j:CaseLineageCase"
        },
        "j:CaseOfficial": {
          "description": "A justice official's involvement in a court case.",
          "items": {
            "$ref": "#/definitions/j:CaseOfficial"
          },
          "minItems": 0,
          "type": "array"
        }
      },
      "type": "object"
    },
    "j:CaseLineageCase": {
      "$ref": "./nc.schema.json#/definitions/nc:CaseType",
      "description": "A description of a case at a previous stage."
    },
    "j:CaseCourt": {
      "$ref": "#/definitions/j:CourtType",
      "description": "A court of law in which the case is being tried."
    },
    "j:CaseOfficial": {
```

```
      "$ref": "#/definitions/j:CaseOfficialType",
      "description": "A justice official's involvement in a court case."
    },
    "j:CaseOfficialType": {
      "additionalProperties": false,
      "description": "A data type for an official's involvement in a case.",
      "properties": {
        "j:JudicialOfficialRegistrationIdentification": {
          "description": "An identification assigned to a judicial official af
          "$ref": "#/definitions/j:JudicialOfficialRegistrationIdentification'
        }
      },
      "required": [
      ],
      "type": "object"
    },
    "j:CourtName": {
      "$ref": "./xs.schema.json#/definitions/xs:string",
      "description": "A name of a unit of a court."
    },
    "j:CourtType": {
      "additionalProperties": false,
      "description": "A data type for a court or a unit of a court responsible
      "properties": {
        "j:CourtName": {
          "description": "A name of a unit of a court.",
          "$ref": "#/definitions/j:CourtName"
        }
      },
      "type": "object"
    },
    "j:JudicialOfficialRegistrationIdentification": {
      "$ref": "./nc.schema.json#/definitions/nc:IdentificationType",
      "description": "An identification assigned to a judicial official after
    }
  },
  "properties": {
  },
  "required": [
  ]
}
```

Here is an example of another JSON schema with the augmentation extending`nc:CaseType`:

```
{
  "$id": "http://release.niem.gov/niem/niem-core/4.0/",
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "additionalProperties": false,
  "definitions": {
```

```
"nc:ActivityDate": {
  "$ref": "#/definitions/nc:DateType",
  "description": "A date of an activity."
},
"nc:ActivityDescriptionText": {
  "$ref": "./xs.schema.json#/definitions/xs:string",
  "description": "A description of an activity."
},
"nc:CaseTitleText": {
  "$ref": "./xs.schema.json#/definitions/xs:string",
  "description": "An official name of a case."
},
"nc:CaseType": {
  "additionalProperties": false,
  "description": "A data type for an aggregation of information about a se
  "properties": {
    "j:CaseAugmentation": {
      "description": "Additional information about a case.",
      "items": {
        "$ref": "./j.schema.json#/definitions/j:CaseAugmentation"
      },
      "minItems": 0,
      "type": "array"
    },
    "nc:ActivityDate": {
      "description": "A date of an activity.",
      "$ref": "#/definitions/nc:ActivityDate"
    },
    "nc:ActivityDescriptionText": {
      "description": "A description of an activity.",
      "$ref": "#/definitions/nc:ActivityDescriptionText"
    },
    "nc:CaseTitleText": {
      "description": "An official name of a case.",
      "$ref": "#/definitions/nc:CaseTitleText"
    }
  },
  "type": "object"
},
"nc:Date": {
  "$ref": "./xs.schema.json#/definitions/xs:date",
  "description": "A full date."
},
"nc:DateTime": {
  "$ref": "./xs.schema.json#/definitions/xs:dateTime",
  "description": "A full date and time."
},
"nc:DateType": {
  "additionalProperties": false,
  "description": "A data type for a calendar date.",
```

```
      "properties": {
        "nc:Date": {
          "description": "A full date.",
          "$ref": "#/definitions/nc:Date"
        },
        "nc:DateTime": {
          "description": "A full date and time.",
          "$ref": "#/definitions/nc:DateTime"
        }
      },
      "type": "object"
    },
    "nc:IdentificationID": {
      "$ref": "./xs.schema.json#/definitions/xs:string",
      "description": "An identifier."
    },
    "nc:IdentificationType": {
      "additionalProperties": false,
      "description": "A data type for a representation of an identity.",
      "properties": {
        "nc:IdentificationID": {
          "description": "An identifier.",
          "$ref": "#/definitions/nc:IdentificationID"
        }
      },
      "type": "object"
    }
  },
  "properties": {
    "nc:Case": {
      "$ref": "#/definitions/nc:CaseType",
      "description": "An aggregation of information about a set of related act
    }
  },
  "required": [
  ]
}
```

These schemas reference another schema: xs

Here is a JSON instance of `nc:Case` extended with the augmentation:

```
{
  "@context": {
    "aamva_d20": "http://release.niem.gov/niem/codes/aamva_d20/4.0/#",
    "clsa": "http://reference.niem.gov/niem/specification/code-lists/1.0/code-
    "ct": "http://release.niem.gov/niem/conformanceTargets/3.0/#",
    "j": "http://release.niem.gov/niem/domains/jxdm/6.0/#",
    "mmucc": "http://release.niem.gov/niem/codes/mmucc/4.0/#",
    "nc": "http://release.niem.gov/niem/niem-core/4.0/#",
```

```
    "ncic": "http://release.niem.gov/niem/codes/fbi_ncic/4.0/#",
    "niem-xs": "http://release.niem.gov/niem/proxy/xsd/4.0/#",
    "structures": "http://release.niem.gov/niem/structures/4.0/#",
    "term": "http://release.niem.gov/niem/localTerminology/3.0/#",
    "ucr": "http://release.niem.gov/niem/codes/fbi_ucr/4.0/#",
    "xs": "http://www.w3.org/2001/XMLSchema#"
  },
  "nc:Case": {
    "j:CaseAugmentation": [
      {
        "j:CaseCourt": {
          "j:CourtName": "Springfield Circuit Court"
        },
        "j:CaseLineageCase": {
          "j:CaseAugmentation": [
            {
              "j:CaseCourt": {
                "j:CourtName": "Springfield Circuit Court"
              }
            }
          ],
          "nc:ActivityDate": {
            "nc:Date": "2018-01-31"
          },
          "nc:ActivityDescriptionText": "original case",
          "nc:CaseTitleText": "Smith v. Jones"
        },
        "j:CaseOfficial": [
          {
            "j:JudicialOfficialRegistrationIdentification": {
              "nc:IdentificationID": "100"
            }
          }
        ]
      }
    ],
    "nc:ActivityDate": {
      "nc:Date": "2018-02-01"
    },
    "nc:ActivityDescriptionText": "counter claim",
    "nc:CaseTitleText": "Jones v. Smith"
  }
}
```

Metadata

Metadata is information about objects or properties (e.g. the source or security level of the information).

Here is an example of NIEM metadata associated with the object `nc:Case` in a JSON schema:

```
{
  "$id": "http://release.niem.gov/niem/niem-core/4.0/",
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "additionalProperties": false,
  "definitions": {
    "nc:CaseTitleText": {
      "$ref": "./xs.schema.json#/definitions/xs:string",
      "description": "An official name of a case."
    },
    "nc:CaseType": {
      "additionalProperties": false,
      "description": "A data type for an aggregation of information about a se
      "properties": {
        "structures:metadata": {
          "$ref": "./structures.schema.json#/definitions/structures:metadata"
        },
        "structures:relationshipMetadata": {
          "$ref": "./structures.schema.json#/definitions/structures:relationsh
        },
        "structures:uri": {
          "$ref": "./structures.schema.json#/definitions/structures:uri"
        },
        "nc:CaseTitleText": {
          "description": "An official name of a case.",
          "$ref": "#/definitions/nc:CaseTitleText"
        }
      },
      "type": "object"
    },
    "nc:Date": {
      "$ref": "./xs.schema.json#/definitions/xs:date",
      "description": "A full date."
    },
    "nc:DateTime": {
      "$ref": "./xs.schema.json#/definitions/xs:dateTime",
      "description": "A full date and time."
    },
    "nc:DateType": {
      "additionalProperties": false,
      "description": "A data type for a calendar date.",
      "properties": {
        "nc:Date": {
```

```
      "description": "A full date.",
      "$ref": "#/definitions/nc:Date"
    },
    "nc:DateTime": {
      "description": "A full date and time.",
      "$ref": "#/definitions/nc:DateTime"
    }
  },
  "type": "object"
},
"nc:EffectiveDate": {
  "$ref": "#/definitions/nc:DateType",
  "description": "A date that information take effect."
},
"nc:ExpirationDate": {
  "$ref": "#/definitions/nc:DateType",
  "description": "A date after which information is no longer valid."
},
"nc:LanguageCode": {
  "$ref": "./xs.schema.json#/definitions/xs:token",
  "description": "A system of words or symbols used for communication."
},
"nc:LastUpdatedDate": {
  "$ref": "#/definitions/nc:DateType",
  "description": "A date information was last modified."
},
"nc:MetadataType": {
  "additionalProperties": false,
  "description": "A data type for information that further qualifies prima
  "properties": {
    "structures:id": {
      "$ref": "./xs.schema.json#/definitions/xs:ID"
    },
    "nc:EffectiveDate": {
      "description": "A date that information take effect.",
      "$ref": "#/definitions/nc:EffectiveDate"
    },
    "nc:ExpirationDate": {
      "description": "A date after which information is no longer valid.",
      "$ref": "#/definitions/nc:ExpirationDate"
    },
    "nc:LanguageCode": {
      "description": "A system of words or symbols used for communication.
      "$ref": "#/definitions/nc:LanguageCode"
    },
    "nc:LastUpdatedDate": {
      "description": "A date information was last modified.",
      "$ref": "#/definitions/nc:LastUpdatedDate"
    },
    "nc:SensitivityText": {
```

```
          "description": "A sensitivity level of the information.",
          "$ref": "#/definitions/nc:SensitivityText"
        }
      },
      "type": "object"
    },
    "nc:SensitivityText": {
      "$ref": "./xs.schema.json#/definitions/xs:string",
      "description": "A sensitivity level of the information."
    }
  },
  "properties": {
    "nc:Case": {
      "$ref": "#/definitions/nc:CaseType",
      "description": "An aggregation of information about a set of related act
    },
    "nc:Metadata": {
      "$ref": "#/definitions/nc:MetadataType",
      "description": "Information that further qualifies primary data; data ab
    }
  },
  "required": [
  ]
}
```

This schema references other schema: xs and structures

Here is a JSON instance of that metadata:

```
{
  "@context": {
    "nc": "http://release.niem.gov/niem/niem-core/4.0/#",
    "structures": "http://release.niem.gov/niem/structures/4.0/#",
    "xs": "http://www.w3.org/2001/XMLSchema#"
  },
  "nc:Case": {
    "@id": "Case1",
    "structures:metadata": "Metadata1 ",
    "nc:CaseTitleText": "Smith v Jones"
  },
  "nc:Metadata": {
    "@id": "Metdata1",
    "nc:EffectiveDate": {
      "nc:Date": "2018-01-31"
    },
    "nc:LanguageCode": "en",
    "nc:LastUpdatedDate": {
      "nc:Date": "2018-01-31"
    },
    "nc:SensitivityText": "Public"
```

```
    }
}
```

Roles

Roles are temporary functions played by objects (e.g. the person that is the arresting officer or the subject in a law enforcement case).

Each role references the base object with the `@id` keyword from JSON-LD.

Here is an example of a NIEM role `j:ArrestOfficial` in JSON schema:

```
{
  "$id": "http://release.niem.gov/niem/domains/jxdm/6.0/",
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "additionalProperties": false,
  "definitions": {
    "j:EnforcementOfficialBadgeIdentification": {
      "$ref": "./nc.schema.json#/definitions/nc:IdentificationType",
      "description": "An identification used to refer to an enforcement offici
    },
    "j:EnforcementOfficialType": {
      "additionalProperties": false,
      "description": "A data type for a person involved in the enforcement of
      "properties": {
        "j:EnforcementOfficialBadgeIdentification": {
          "description": "An identification used to refer to an enforcement of
          "$ref": "#/definitions/j:EnforcementOfficialBadgeIdentification"
        },
        "nc:RoleOfPerson": {
          "description": "A person of whom the role object is a function.",
          "$ref": "./nc.schema.json#/definitions/nc:RoleOfPerson"
        }
      },
      "required": [
        "nc:RoleOfPerson"
      ],
      "type": "object"
    }
  },
  "properties": {
    "j:ArrestOfficial": {
      "$ref": "#/definitions/j:EnforcementOfficialType",
      "description": "A peace official who makes an arrest."
    },
    "nc:Person": {
      "$ref": "./nc.schema.json#/definitions/nc:PersonType",
      "description": "A human being."
    }
  },
```

```
  "required": [
  ]
}
```

This schema references other schemas: nc and xs

Here is a JSON instance of that role:

```
{
  "@context": {
    "j": "http://release.niem.gov/niem/domains/jxdm/6.0/#",
    "nc": "http://release.niem.gov/niem/niem-core/4.0/#",
    "xs": "http://www.w3.org/2001/XMLSchema#"
  },
  "j:ArrestOfficial": {
    "j:EnforcementOfficialBadgeIdentification": {
      "nc:IdentificationID": "100",
      "nc:IdentificationSourceText": "Springfield Police Department"
    },
    "nc:RoleOfPerson": {
      "@id": "Person1"
    }
  },
  "nc:Person": {
    "@id": "Person1",
    "nc:PersonName": {
      "nc:PersonFullName": "John Smith"
    }
  }
}
```

Non-Normative Guidance in Using NIEM with JSON

Abstract

This document provides guidance from the NIEM Technical Architecture Committee (NTAC) for using the National Information Exchange Model (NIEM) with JavaScript Object Notation (JSON). NIEM provides a well-established standard for defining information exchanges. JSON is specified by RFC4627.

Status

This guidance is non-normative. It discusses possible NIEM conformance rules for JSON data, but it does not establish any specification for such conformance.

This document is a working draft; revisions and additions are expected. Readers are invited to provide feedback on this document by entering an issue at https://github.com/NIEM/NIEM.github.io/issues or sending email to niem-comments@lists.gatech.edu.

Authors

- Scott Renner sar@mitre.org, @iamdrscott
- Webb Roberts webb.roberts@gtri.gatech.edu, @webb
- Leila Tite leila.tite@hennepin.us, @leilatite
- Tom Carlson tom@tomcarlsonconsulting.com

Contents

- Introduction
  - Audience
  - Overview
  - The NIEM data model and RDF
  - Use of JSON-LD
  - NIEM XML and NIEM JSON carry the same data
- JSON-LD template for NIEM IEP
- JSON-LD representation of NIEM XML
  - IEP XML instance document
  - Namespaces and JSON-LD context
  - Document element (A.K.A. root element)
  - Element content of an object or association type
  - Repeated Elements
  - Element with Complex Content and Attributes
  - Element with Simple Content and Attributes
  - Element with Simple Content and No Attributes

    - Element with Numeric or Boolean Content
    - Elements with empty and nilled content
    - ID Attributes
    - References and IDREF attributes
    - Abstract Elements and Substitution Groups
    - Augmentations
    - Metadata
    - Adapter Elements
        - Pretend the external content is NIEM conforming
        - Create a custom mapping to JSON-LD
        - Represent the external content with GeoJSON
        - Represent the external content as an XMLLiteral blob
- Implementing Translators
- Advanced Concerns for JSON-LD Serialization
    - JSON-LD as plain JSON
    - Order of keys not significant
    - Arrays may be omitted
- Additional guidance
    - Expressing types
    - Linking contexts via HTTP headers
- Road map for future work
    - NIEM Conformance
    - Support for transforming between XML and JSON
- References
- Full example:
    - Full example: XML instance document
    - Full example: JSON data
        - RDF

Introduction

This document provides guidance from the NIEM Technical Architecture Committee (NTAC) for using the National Information Exchange Model (NIEM) with JavaScript Object Notation (JSON). NIEM provides a well-established standard for defining information exchanges. JSON is specified by RFC4627.

This document is provided as guidance, not as a normative specification. It discusses possible NIEM conformance rules for JSON data, but it does not establish any specification for such conformance. This document is the first step in establishing a repeatable and interoperable methodology for using JSON to represent NIEM-conformant information exchanges. The guidelines presented by this document should be in alignment with later normative specifications that define NIEM-conformant JSON.

Audience

The primary target of this guidance is the developer who has a NIEM IEPD that defines one or more XML messages, and who wishes to exchange JSON representations of those messages, instead of XML. For these users, this document supplies patterns for converting an NIEM XML document (i.e., an IEP, an information exchange package) into a semantically-equivalent JSON serialization.

This document also provides guidance for the developer who wishes to use NIEM definitions in a JSON object design. Guidance for developers who wish to map existing JSON objects to NIEM XML may be provided at a later date.

Overview

This document is one step in the NTAC technical road map for JSON, Using JSON with NIEM. It describes NIEM's methodology for creating consistent, interoperable JSON messages that are based on NIEM-conformant XML schemas. Core aspects of this guidance are:

1. The guidance describes JSON messages based on NIEM Schemas.
2. The JSON messages are expressed as JSON-LD, a scalable JSON framework for linked data.
3. The names, relationships and structures in JSON messages are based on names, relationships and structures described by NIEM-conformant XML schemas.
4. The details of how JSON-LD represents NIEM XML data is based on the NIEM Naming and Design Rules's mapping between XML (and XML Schema) and RDF.

The NIEM data model and RDF

The NIEM Naming and Design Rules (NDR) is the main document that explains the meaning of NIEM–conformant XML schemas and XML instance documents. The framework that the NIEM NDR relies on for meaning is the Resource Description Framework (RDF), which defines a data model that is the basis for Semantic Web technologies. NIEM defines the meaning of conformant XML schemas and XML instance documents in terms of RDF. This document leverages the NDR's RDF to map NIEM XML schemas and XML instance documents to JSON-LD.

Although NIEM is, at its core, defined in terms of RDF, most users of NIEM do not have to understand much about RDF; NIEM is primarily discussed in terms of XML and XML schema: rules about elements, attributes, simple and complex types, etc. Similarly, users of this guidance document, and of NIEM's JSON-LD representation, will not need to understand much about RDF. The few RDF concepts this document explicitly requires are explained without requiring deep understanding of the underlying RDF concepts.

The NIEM NDR Section 5, "The NIEM conceptual model" describes the conceptual model of NIEM, and provides a mapping between NIEM-conformant XML schemas and XML instance documents, and RDF. This section is the basis for much of what appears in this document. For example:

- Section 5.6.1, "Resource IRIs for XML Schema components and information items" describes how to find an internationalized resource identifier (IRI) for a schema component or XML element or attribute, using the namespace name and local name of the component's qualified name (QName). This is the basis for the translation of element names into JSON-LD terms.
- Section 5.6.5.4, "Element as a simple property of an object or association", describes the RDF for an element of an object or association type. This is the basis for how a NIEM-conformant XML element is translated into corresponding JSON-LD.

These, and other sections of the NDR are the basis of the translation from NIEM XML to JSON-LD. A goal of this document is to explain the translations from NIEM to JSON-LD sufficiently, so that the reader does not need to understand the NDR's mappings to RDF, nor the JSON-LD's RDF basis.

Use of JSON-LD

This document describes the translation of NIEM-conformant XML instance documents (informed by certain features of schemas, such as component types) into JSON, using JSON-LD as the target JSON representation. JSON-LD (JavaScript Object Notation for Linking Data) is a language-independent data format for representing Linked Data, expressed as JSON. JSON-LD provides a lightweight mechanism that allows data to be linked across websites. The syntax is easy for humans to read and easy for machines to parse and generate.

One reason for choosing JSON-LD is its context mechanism, which allows IRIs in JSON-LD to look like XML qualified names (QNames). As described above, the NDR defines how to translate element QNames to IRIs. These IRIs can be represented in a short form using a JSON-LD context.

Take, for example, the following JSON data:

```
{
  "http://release.niem.gov/niem/niem-core/3.0/#quantityUnitText" : "dozen"
}
```

The key in this JSON object is the IRI defined by NIEM for the `nc:quantityUnitText` attribute. Using the JSON-LD context mechanism, that IRI can be compacted. as follows:

```
{
  "@context": {
    "nc": "http://release.niem.gov/niem/niem-core/3.0/#"
  },
  "nc:quantityUnitText": "dozen"
}
```

The benefits of a JSON-LD representation are apparent. The context object preserves the information in the XML namespace declarations. The JSON object key is a familiar NIEM QName. That QName can be converted into the full IRI by an expansion algorithm defined for JSON-LD by JSON-LD 1.0 Processing Algorithms and API, Section 2.1, “Expansion”.

A second reason for choosing JSON-LD is that JSON-LD is a concrete RDF syntax, like RDF/XML or Turtle. This means that NIEM JSON-LD data may be processed using RDF techniques, such as SPARQL queries. This also means that there are now two methods of generating RDF from NIEM-conforming XML documents and schemas: the first defined in NIEM NDR Section 5, the second described by this guidance. The NTAC’s intention is that the two methods produce consistent RDF; realizing this intention may require future changes to the NDR, this guidance, or both.

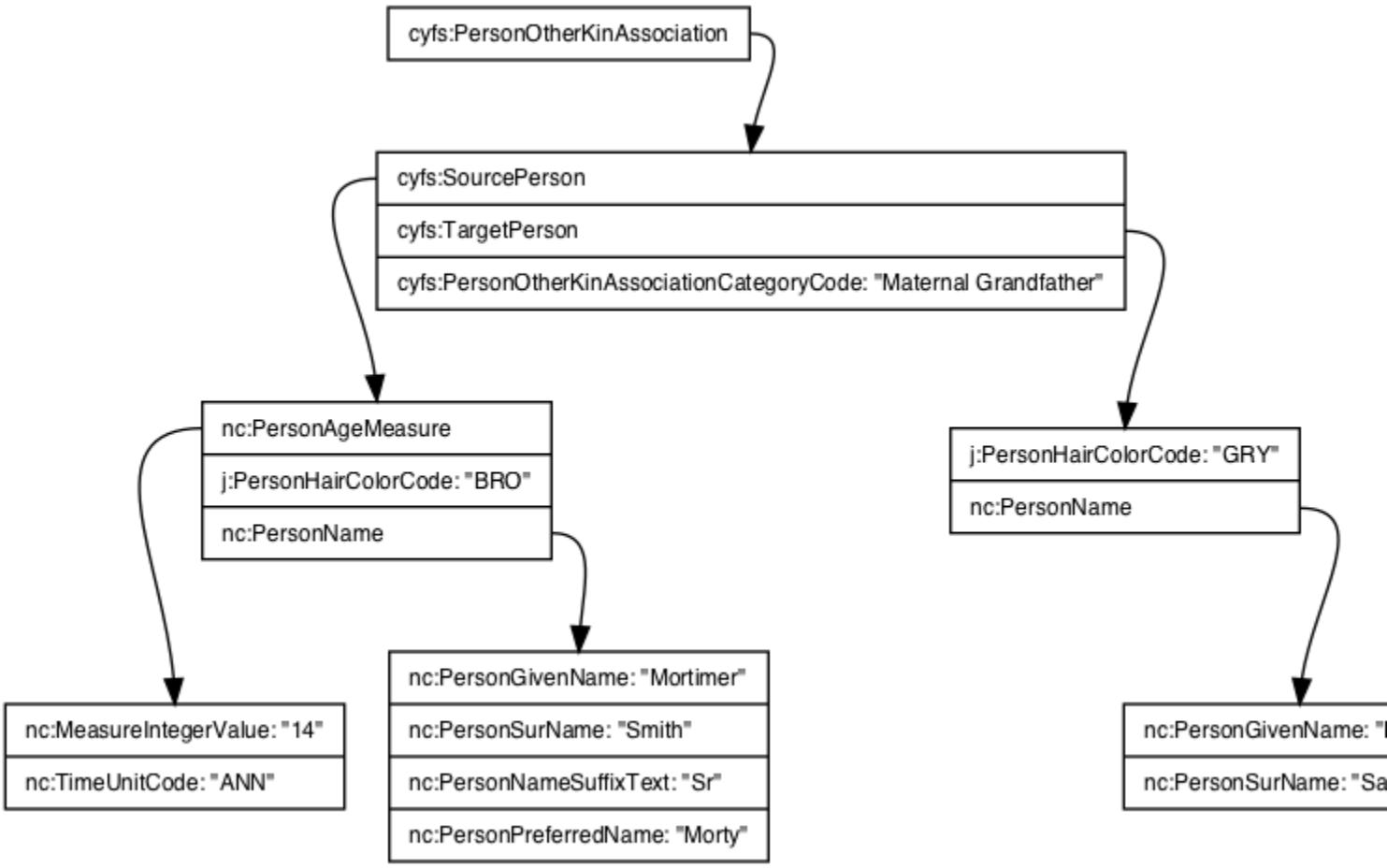
Based on the above, all three of these methods are legitimate ways to access NIEM JSON-LD data:

1. A developer can work with the data as if it is plain JSON, without worrying about details of JSON-LD.
2. A developer can use JSON-LD tools as part of the data processing, including using compaction against a known context, expansion, flattening, or framing.
3. A developer can process the data as RDF, possibly first converting it to RDF/XML or Turtle syntax.

The guidance in the next major section is intended to describe a JSON-LD serialization of NIEM-conforming XML that is easy to generate, and convenient to process, for all three of these consumer use cases. Certain advanced concerns for that serialization may be found in section 4. This document does not attempt to teach best practices for JSON or JSON-LD; for that, the reader may turn to http://json-ld.org/spec/latest/json-ld-api-best-practices, or other similar resources.

NIEM XML and NIEM JSON carry the same data

A NIEM-conformant schema defines a simple object model. Here is an example of data that is defined by a NIEM schema:



This data can be represented in either of two forms: XML or JSON. Here are an XML and a JSON representation for this data, side-by-side:

```
<?xml version="1.0" encoding="UTF-8"?>
<cyfs:PersonOtherKinAssociation
  xmlns:cyfs="http://release.niem.gov/niem/domains/cyfs/3.2/"
  xmlns:j="http://release.niem.gov/niem/domains/jxdm/5.2/"
  xmlns:nc="http://release.niem.gov/niem/niem-core/3.0/">


  <cyfs:SourcePerson>
```

```
{
  "@co
    "c
    "j
    "n
  },
  "cyf
```

```
    <nc:PersonAgeMeasure>
      <nc:MeasureIntegerValue>14</nc:MeasureIntegerValue>
      <nc:TimeUnitCode>ANN</nc:TimeUnitCode>
    </nc:PersonAgeMeasure>
    <j:PersonHairColorCode>BRO</j:PersonHairColorCode>
    <nc:PersonName>
      <nc:PersonGivenName>Mortimer</nc:PersonGivenName>
      <nc:PersonSurName>Smith</nc:PersonSurName>
      <nc:PersonNameSuffixText>Sr</nc:PersonNameSuffixText>
      <nc:PersonPreferredName>Morty</nc:PersonPreferredName>
    </nc:PersonName>
  </cyfs:SourcePerson>
  <cyfs:TargetPerson>
    <j:PersonHairColorCode>GRY</j:PersonHairColorCode>
    <nc:PersonName>
      <nc:PersonGivenName>Rick</nc:PersonGivenName>
      <nc:PersonSurName>Sanchez</nc:PersonSurName>
    </nc:PersonName>
  </cyfs:TargetPerson>
  <cyfs:PersonOtherKinAssociationCategoryCode
    >Maternal Grandfather</cyfs:PersonOtherKinAssociationCategoryCode>
</cyfs:PersonOtherKinAssociation>
```

```
},
"c
},
"c
}
}
```

You can exchange NIEM data either as XML or as JSON. Either way, the data that is expected in a message, and the meaning of that data, is defined by NIEM schemas.

The NIEM JSON representation uses JSON-LD, which means that order of NIEM JSON data is flexible, that the names of keys is determined by JSON-LD contexts, that the JSON representation is also an RDF representation, and that there are additional tools available for working with NIEM JSON.

One method of facilitating NIEM JSON exchanges is for someone who understands NIEM schemas to construct a template JSON instance, which is a NIEM JSON instance for a NIEM schema, but with data values replaced with `null`. This method is described by the next section.

Another method of facilitating NIEM JSON exchanges is to translate between NIEM XML and NIEM JSON. This is described in a section below. That section also describes how NIEM JSON represents data that is more complex. Although NIEM XML is used in that section describe how NIEM JSON works in various situations, you do not have to work with NIEM XML or translate between JSON and XML to work with NIEM JSON.

JSON-LD template for NIEM IEP

One way to make it simpler for the JSON developer to create JSON-LD documents for a NIEM exchange is to provide a fully completed JSON-LD document structure with all “null” values. The IEP designer provides an iep-sample-template.jsonld document in the iep-samples directory that describes the context and structure of the document for the exchange. The developer populates the fields for which they have values and submits the document. All elements with null values are ignored during processing

at the receiving end.

The following JSON example is a JSON IEP template for the simple object model from Section 1.5, above.

```
{
  "@context": {
    "exch": "http://example.com/SimpleExchange/1.0/#",
    "j": "http://release.niem.gov/niem/domains/jxdm/5.1/#",
    "nc": "http://release.niem.gov/niem/niem-core/3.0/#",
    "cyfs": "http://release.niem.gov/niem/domains/cyfs/3.2/"
  },
  "cyfs:PersonOtherKinAssociation": {
    "cyfs:SourcePerson": {
      "nc:PersonAgeMeasure": {
        "nc:MeasureIntegerValue": null,
        "nc:TimeUnitCode": null
      },
      "j:PersonHairColorCode": null,
      "nc:PersonName": {
        "nc:PersonGivenName": null,
        "nc:PersonSurName": null,
        "nc:PersonNameSuffixText": null,
        "nc:PersonPreferredName": null
      }
    },
    "cyfs:TargetPerson": {
      "j:PersonHairColorCode": null,
      "nc:PersonName": {
        "nc:PersonGivenName": null,
        "nc:PersonSurName": null
      }
    },
    "cyfs:PersonOtherKinAssociationCategoryCode": null
  }
}
```

JSON-LD representation of NIEM XML

This section describes the translation of an example IEP (information exchange package, an XML instance document), defined by an IEPD (information exchange package description), into the corresponding JSON-LD data. Each section within highlights the transformation of a NIEM or XML concept into corresponding JSON-LD. The full source XML appears in an appendix below. The resulting JSON-LD also appears in an appendix below.

This section makes simplifying assumptions, which may not apply to every NIEM IEP. If your IEP is more complicated, then you may have to extend the guidelines to cover your data.

Throughout these examples, at each stage, content is omitted or held back for later sections. When content is left out, an ellipsis ("…") appears in place of the omitted text.

IEP XML instance document

The IEP as a whole is represented by a JSON object, evident in an instance by an outside set of curly braces. The sub-objects for the root element and `@context` go within that object:

```
{
    "@context" : {
        ...
    },
   ...
}
```

Rules for what goes in this object follow below.

Namespaces and JSON-LD context

NIEM uses XML namespaces to distinguish components with similar names, to identify the authorities responsible for managing a set of XML schema components, and to organize data definitions among NIEM Core, domains, IEPD extensions, etc. Every NIEM IEP has namespace declarations for its content. To uniquely specify a NIEM element, attribute, or schema component, its local name must be combined with the namespace associated with its namespace prefix.

The context mechanism in JSON-LD can serve a purpose similar to XML namespaces. JSON-LD uses International Resource Identifiers (IRIs) as keys within JSON objects, and to uniquely identify objects. A JSON-LD context can define parts of IRIs, which can be used in short-form terms which expand into full IRIs. So, XML namespaces in the IEP are mapped to JSON-LD context entries, to support short-form terms.

ASSUMPTIONS: All of the IEP's namespace declarations are at or above the IEP's root element (if there is an envelope around the IEP's root element). There is no content using the default (no namespace prefix) namespace.

These are the namespace declarations in the sample IEP:

```
<exch:CrashDriverInfo
    xmlns:exch="http://example.com/CrashDriver/1.0/"
    xmlns:j="http://release.niem.gov/niem/domains/jxdm/5.1/"
    xmlns:nc="http://release.niem.gov/niem/niem-core/3.0/"
    xmlns:geo="http://release.niem.gov/niem/adapters/geospatial/3.0/"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:structures="http://release.niem.gov/niem/structures/3.0/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <nc:Person structures:id="P01">
  ...
</exch:CrashDriverInfo>
```

Each declared namespace prefix is converted into an `@context` entry:

- The namespace prefix becomes a short term.
- The namespace name assigned to the prefix is converted into an IRI root.

The rules for constructing IRIs from QNames are provided by NDR section 5.6.1, "Resource IRIs for XML Schema components and information items". This leads to the definition of IRI roots as follows:

- If the namespace name ends with "#", then the IRI root is the namespace name.
- Otherwise: concatenate(namespace name, "#").

For the QName `nc:Person` this yields the IRI `http://release.niem.gov/niem/niem-core/3.0/#Person`.

In addtion, we apply the following guidelines:

- Omit the `xsi` namespace, `http://www.w3.org/2001/XMLSchema-instance`, which is reserved for XML Schema-specific concepts, and is not carried over into JSON-LD.
- Include a declaration of `rdf` to `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. This is used for `rdf:value` and `rdf:XMLLiteral`.

This yields the following `@context` entry:

```
{
    "@context" : {
        "exch" : "http://example.com/CrashDriver/1.0/#",
        "j" : "http://release.niem.gov/niem/domains/jxdm/5.1/#",
        "nc" : "http://release.niem.gov/niem/niem-core/3.0/#",
        "geo" : "http://release.niem.gov/niem/adapters/geospatial/3.0/",
        "gml" : "http://www.opengis.net/gml/3.2#",
        "structures" : "http://release.niem.gov/niem/structures/3.0/#",
        "rdf" : "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    },
    "exch:CrashDriverInfo" : {
        ...
    }
}
```

Document element (A.K.A. root element)

Each element within the IEP is converted to a key in a JSON-LD node object.

The root element (or document element) of a NIEM IEP is also converted to a node object. It is given the JSON key corresponding to the QName (XML qualified name) of the element. This may be expanded into an IRI by a JSON-LD processor, but this guidance recommends using the QName, supported by a context entry for the IRI root. Within the full example, this yields:

```
{
  "@context": {
```

```
    "exch": "http://example.com/CrashDriver/1.0/#",
    ...
  },
  "exch:CrashDriverInfo": {
    ...
  }
}
```

Seems to me we need only one of sections 2.1 and 2.3. I'd keep 2.3. —@iamdrscott

Element content of an object or association type

Since the top level element is a NIEM object type, we convert most elements within to a key for the top-level element's JSON-LD object. In the sample instance, `exch:CrashDriverInfo` has the following child elements:

- `nc:Person`
- `j:Crash`
- `j:PersonChargeAssociation`
- `j:Charge`
- `j:JusticeMetadata`

These are each converted into keys within the `exch:CrashDriverInfo` object described above:

```
...
"exch:CrashDriverInfo": {
  "nc:Person": {
      ...
  },
  "j:CrashDriver": {
      ...
  },
  "j:PersonChargeAssociation": {
      ...
  },
  "j:Charge": {
      ...
  }
  "j:JusticeMetadata": {
      ...
  }
}
```

This is the most common case; this is how object types and association type are translated to JSON-LD.

Note that there is only one pair for each child element *name*, no matter how many times that element appears in the content. It is now time to discuss…

Repeated Elements

In the full example XML IEP, The element `nc:PersonMiddleName` is repeated within `nc:PersonName`:

```
<nc:PersonName nc:personNameCommentText="copied">
    <nc:PersonGivenName nc:sequenceID="1">Peter</nc:PersonGivenName>
    <nc:PersonMiddleName nc:sequenceID="2">Death</nc:PersonMiddleName>
    <nc:PersonMiddleName nc:sequenceID="3">Bredon</nc:PersonMiddleName>
    <nc:PersonSurName>Wimsey</nc:PersonSurName>
</nc:PersonName>
```

Since `nc:PersonMiddleName` is repeated, it is represented using the key `nc:PersonMiddleName`, with a value that is an array of objects:

```
"nc:PersonName": {
  "nc:PersonGivenName": <!-- content of element nc:PersonGivenName -->,
  "nc:PersonMiddleName": [
    <!-- content of first element nc:PersonGivenName -->,
    <!-- content of second element nc:PersonGivenName -->
  ],
  "nc:PersonSurName": <!-- content of element nc:PersonSurName -->,
  "nc:personNameCommentText": "copied"
}
```

We see that occurrences of element `nc:PersonMiddleName` are bundled together as an array, and that array is the value for key `nc:PersonMiddleName`. A repeated element is converted into a JSON object with an array value. (This is done for elements with simple content as well as the complex elements shown.) The array contains one JSON object for the content of each element instance. Note that order of data within an array should not be considered significant, as described below.

Consumers of NIEM JSON-LD data must be aware that the value of a repeatable element may be an array (if the element is in fact repeated), or an object (if it is not), and code accordingly.

Observe that with this guidance, the same JSON is produced for these two `Parent` elements:

```
|<Parent>          |    <Parent>          |
|    <Repeated/>   |        <Repeated/>   |
|    <Repeated/>   |        <Child/>      |
|    <Child/>      |        <Repeated/>   |
|</Parent>         |    </Parent>         |
```

Therefore it is up to the developer to ensure that these two elements have the same meaning in the IEP. If the meaning depends on the difference in element ordering, then the guidance in this document does not apply, and the developer is on his own. However, in that case, the IEPD design is bad, conflicting with the NIEM Conceptual Model.

Element with Complex Content and Attributes

The content of an element with complex content is converted to a JSON object with one pair for the name of each attribute and one pair for the name of each child element in that content. For example, in the sample IEP we have

```
<nc:PersonName nc:personNameCommentText="copied">
    <nc:PersonGivenName nc:sequenceID="1">Peter</nc:PersonGivenName>
    <nc:PersonMiddleName nc:sequenceID="2">Death</nc:PersonMiddleName>
    <nc:PersonMiddleName nc:sequenceID="3">Bredon</nc:PersonMiddleName>
    <nc:PersonSurName>Wimsey</nc:PersonSurName>
</nc:PersonName>
```

which is converted to the following JSON

```
"nc:PersonName": {
  "nc:personNameCommentText": "copied",
  "nc:PersonGivenName": {
    ...
  },
  "nc:PersonMiddleName": [
    ...
  ],
  "nc:PersonSurName": {
    ...
  }
}
```

Observe that for a NIEM-conforming IEP, there can't be a collision between attribute and child element names, because of the camel-case rule. For an IEP that is not NIEM conforming, it is possible to have an attribute and a child element with the same name. This document has no guidance for such non-conforming IEPs; developers are on their own.

Element with Simple Content and Attributes

The `nc:PersonGivenName` element has simple content and attributes. Its JSON representation is

```
  "nc:PersonGivenName" : {
    "nc:sequenceID" : 1,
    "rdf:value" : "Peter"
  }
```

The element's simple content can't be represented by `{"nc:PersonGivenName" : "Peter" }`, because then there is no place to put the attributes. Instead, the representation must be an object, with ordinary keys for the attributes and a special key for the simple content. That special key could have a magic syntax, such as `"."`, but this would only work for plain JSON consumers; to process the data as JSON-LD or RDF, the key must be a term mapped to an IRI.

According to the RDF Primer (2004), `rdf:value` is customarily used in this situation where a

property has one main value and one or more additional values providing context that qualifies the main value. Therefore, this guidance chooses `rdf:value` for the special key. This choice also makes the JSON-LD representation consistent with NDR content of an element, which specifies that non-empty simple values are mapped in this way.

Obviously this will break if an element in the IEP has `rdf:value` as an attribute. Fortunately, there is no good reason to do that in a NIEM IEP.

Element with Simple Content and No Attributes

The `nc:Date` element has simple content and no attributes. Its JSON-LD representation is

```
"nc:Date" :"1893-05-04"
```

With no attributes, there is no need for the `rdf:value` key, and so it is not used. Consumers of NIEM JSON-LD data must be aware that the value of a simple element may be an object (if the element has attributes), or a value (if it does not), and code accordingly.

Element with Numeric or Boolean Content

When the IEPD schema defines a numeric type for a simple element, the value of the JSON pair is a number. Likewise, when the schema defines a boolean type, the value of the JSON pair is `true` or `false`. For example, the representations of `nc:MeasureDecimalValue` and `exch:PersonFictionalCharacterIndicator` are:

```
"nc:MeasureDecimalValue" : 9.7

"exch:PersonFictionalCharacterIndicator" : true
```

Elements with empty and nilled content

There are three situations in an XML instance where an element may have empty content:

1. The element has no simple content, and any element content is optional. For example, its type may be a complex type with complex content, which has no simple value, and its element children may have `minOccurs="0"`.
2. The element is defined to have simple or complex content, but the simple content is nilled using `xsi:nil="true"`.
3. The element has simple content, but that simple content is the empty string.

Cases 1 and 2 are represented the same way in JSON-LD: There is no value associated with the element. Take this XML example (case 1):

```
<nc:Person/>
```

…or the equivalent (case 1):

```
<nc:Person></nc:Person>
```

Both of these omit any content. To be a valid XML instance, the schema for this must either define `nc:PersonType` (the type defined by NIEM for element `nc:Person`) with no child elements, or with child elements that have `minOccurs="0"`. A similar instance may use `xsi:nil` (case 2):

```
<nc:Person xsi:nil="true"/>
```

This XML instance may be valid with mandatory element children, as long as `nillable="true"` is set for the element. The JSON for all of these is the same:

```
"nc:Person" : { }
```

This expresses that `nc:Person` is a node object, but does not assert any other properties.

An element carrying an empty string (case 2) is represented differently. In NIEM, element `nc:PersonGivenName` is defined to have simple content based on `xs:string`. So, if that element is empty, it represents the empty string. The XML instance (case 2):

```
<nc:PersonGivenName></nc:PersonGivenName>
```

…and the equivalent XML (case 2):

```
<nc:PersonGivenName/>
```

…are both carrying the empty string as children of `nc:PersonGivenName`. This may be represented with the JSON:

```
"nc:PersonGivenName" : ""
```

ID Attributes

NIEM defines the attribute `structures:id` to carry ID values. `structures:id` is the only ID-typed attribute allowed in NIEM-conformant content (except for externally-defined content). An ID attribute in XML defines a unique document-relative unique identifier: An ID value may appear in at most one element within a single XML document; the same ID value may appear in any number of different documents.

The NIEM-defined `structures:id` attribute is represented by the JSON-LD reserved key `@id`. The value for `@id` is the value of `structures:id`. For example, the XML:

```
<nc:Person structures:id="P01">
  <nc:PersonBirthDate>
    <nc:Date>1893-05-04</nc:Date>
  </nc:PersonBirthDate>
  <nc:PersonName nc:personNameCommentText="copied">
    ...
  </nc:PersonName>
  <exch:PersonFictionalCharacterIndicator>true</exch:PersonFictionalCharacterI
</nc:Person>
```

…is represented by the JSON-LD:

```
"nc:Person" : {
  "@id" : "P01",
  "nc:PersonBirthDate": {
    "nc:Date": "1893-05-04"
  },
  "nc:PersonName": {
    "nc:personNameCommentText" : "copied",
    ...
  },
  "exch:PersonFictionalCharacterIndicator": true
}
```

The JSON-LD processor will process `@id` values against a base IRI. This might be automatically generated by a system (for example, the JSON-LD Playground uses a base IRI of `http://json-ld.org/playground/`). The base IRI may also be set by `@base` within an `@context`, as described by JSON-LD 1.0, Section 6.1, "Base IRI".

References and IDREF attributes

The value of a `structures:ref` attribute and any other IDREF attribute is converted into a reference using the @id of the corresponding JSON-LD node. For example, the representation for `<nc:RoleOfPerson structures:ref="P01" xsi:nil="true"/>` is

```
"nc:RoleOfPerson" : {
  "@id" : "P01"
}
```

The representation of a reference element has no content from the element value, either complex or simple; that is, the only content is the `"@id":"idref"` pair and zero or more attributes. In NIEM JSON-LD data created according to this guidance, this will be true of every object representing a reference element. In addition, there must be exactly one object containing the `"@id":"value"` pair that does have child elements or simple element content. That will be the object representing the element with the `structures:id` attribute, as described above. These constraints are not part of JSON-LD; it is created by this guidance.

Observe that the `xsi:nil` attribute, which is useful only for schema validation, does not appear in the JSON-LD representation.

Abstract Elements and Substitution Groups

Because this guidance does not seek to replicate in JSON the underlying XML Schema powering NIEM, substitution groups are simply represented by replicating the XML instance structure, where the substitution has already taken place. Abstract elements do not appear at all. For example,

```
<nc:Person>
    <!-- Date substituted for DateRepresentation -->
    <nc:PersonBirthDate>
```

```
        <nc:Date>1893-05-04</nc:Date>
    </nc:PersonBirthDate>
</nc:Person>
```

becomes

```
"nc:Person" : {
    "nc:PersonBirthDate" : {
        "nc:Date"  : "1893-05-04"
    }
}
```

Augmentations

NIEM has two kinds of augmentation elements. The most common is a container element that is derived from `AugmentationType` and has a name ending in `Augmentation`; for example, `LicenseAugmentation` in the sample IEP. These augmentation container elements are important for schema validation, but have no role in the conceptual model, and so they do not appear in the JSON representation. So, for example, the representation of `j:DriverLicense` is

```
"j:DriverLicense" : {
  "j:DriverLicenseCardIdentification" : {
      "nc:IdentificationID" : "A1234567"
  },
  "nc:ItemLengthMeasure" : {
      "nc:MeasureDecimalValue" : 9.7,
      "nc:LengthUnitCode" : "CMT"
  }
}
```

The other kind of augmentation element is an element declared in the substitution group of an `AugmentationPoint`. For example, the IEPD schema includes the following element declaration:

```
<xs:element name="PersonFictionalCharacterIndicator" type="niem-xs:boolean"
  substitutionGroup="nc:PersonAugmentationPoint">
    <xs:annotation><xs:documentation>
        True if this person is a fictional character in a literary work.
    </xs:documentation></xs:annotation>
</xs:element>
```

Augmentation elements of this kind are treated the same as any other child element.

Metadata

> We are still working on figuring out how to represent metadata in JSON-LD. The NDR says to hang metadata on RDF statements, but it is not clear what would be the best way to do that. One option is if the XML contains metadata, then create an RDF statement and hang the metadata off of that. Metadata on objects can be handled with rdf:value, but metadata on associations is harder. —@webb, —@leilatite

One way to do that is through reification. The JSON-LD would look something like example from stack overflow:

```
{
  "@context": {
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "subject": { "@id": "rdf:subject", "@type": "@id" },
    "predicate": { "@id": "rdf:predicate", "@type": "@id" },
    "object": { "@id": "rdf:object", "@type": "@id" },
    "j": "http://release.niem.gov/niem/domains/jxdm/5.1/#",
    "structures": "http://release.niem.gov/niem/structures/3.0/#"
  },
  "@type": "rdf:Statement",
  "subject": "j:Charge",
  "predicate": "structures:metadata",
  "object": { "@id": "j:JusticeMetadata" },
  "j:CriminalInformationIndicator": true
}
```

Adapter Elements

IEPD developers sometimes want to reuse schema components defined in an existing standard via a schema that does not conform to the NDR. The adapter elements defined in NDR section 10.2.3, "External adapter types and external components" are the NIEM mechanism for including these external components in the IEPD. For instance, in the following section of the sample IEP, the `geo:LocationGeospatialPoint` element is a NIEM-conforming adapter. The external content of that adapter element (`gml:Point`) is defined by the Open Geospatial Consortium (OGC) in the Geographic Markup Language (GML).

```
<geo:LocationGeospatialPoint>
    <gml:Point gml:id="PT01" srsName="urn:ogc:def:crs:EPSG::4326">
        <gml:pos>51.835 -0.417</gml:pos>
    </gml:Point>
</geo:LocationGeospatialPoint>
```

Because the external content of an adapter element does not follow the NDR, the guidance in this document may not apply. Developers must decide how to convert external content to JSON on a case-by-case basis, and must also decide which of the three consumer use cases (plain JSON, JSON-LD, RDF) they will support. For the adapter element in the sample IEP, the developer might apply any of the following solutions:

Pretend the external content is NIEM conforming

The developer could simply apply the guidance in this document as if the external content were NIEM conforming. In this case, the adapter element is converted to

```
"geo:LocationGeospatialPoint": {
  "gml:Point": {
    "@id": "PT01",
```

```
    "srsName": "urn:ogc:def:crs:EPSG::4326",
    "gml:pos": "51.835 -0.417"
  }
}
```

That will work when processed as plain JSON. When processed as JSON-LD, the resulting IRIs in the expanded form are reasonable. However, because the `srsName` attribute does not have a namespace in GML, it will be dropped by the JSON-LD processor. That will probably cause difficulties for the JSON-LD and RDF consumer use cases.

Create a custom mapping to JSON-LD

The difficulty with the `srsName` attribute could be resolved with a custom mapping, one which extends the guidance in this document by supplying a special `@context` pair for the content of the adapter element. The resulting JSON-LD would be

```
"geo:LocationGeospatialPoint": {
  "@context": {
    "srsName": "http://www.opengis.net/gml/3.2#srsName"
  },
  "gml:Point": {
    "@id": "PT01",
    "srsName": "urn:ogc:def:crs:EPSG::4326",
    "gml:pos": "51.835 -0.417"
  }
}
```

This approach produces a plausible IRI for the `srsName` attribute in the expanded JSON-LD, and so should work for the JSON-LD and RDF consumer use cases. This is the approach followed for the [full example]

Represent the external content with GeoJSON

GeoJSON is a geospatial data interchange format based on JSON. The Internet Engineering Task Force (IETF), in conjunction with the original specification authors, has formed the [Geographic JSON WG] to standardize the format. Although GeoJSON is not a specification of the OGC, it is still a plausible choice for encoding GML content in JSON. With this solution, the adapter element is converted to

```
"geo:LocationGeospatialPoint": {
  "@context": {
    "geometry": "https://datatracker.ietf.org/doc/draft-ietf-geojson/#geometry
    "type": "https://datatracker.ietf.org/doc/draft-ietf-geojson/#type",
    "coordinates": "https://datatracker.ietf.org/doc/draft-ietf-geojson/#coord
  },
  "geometry": {
    "type": "Point",
    "coordinates": [51.835, -0.417]
  }
}
```

That will work when processed as plain JSON, ignoring the `@context` pair. Defining the context as shown will produce plausible IRIs in the expanded JSON-LD and converted RDF. A better base IRI would be desirable, of course.

Represent the external content as an XMLLiteral blob

```
"geo:LocationGeospatialPoint": {
  "@type": "rdf:XMLLiteral",
  "@value":
    "<gml:Point gml:id=\"PT01\"
                srsName=\"urn:ogc:def:crs:EPSG::4326\">
        <gml:pos>51.835 -0.417</gml:pos>
     </gml:Point>"
}
```

Implementing Translators

> TODO: Discuss an **easy button** transfrom from NIEM XML to JSON-LD, treating input like a *canned query* with very little optionality, transforming to JSON-LD using XSLT3's JSON capability. —@webb

Advanced Concerns for JSON-LD Serialization

JSON-LD as plain JSON

A software developer may wish to work with JSON-LD data instances with vanilla JSON tools, which aren't JSON-LD aware. Although developers are encouraged to use JSON-LD tools, using vanilla JSON tools is straightforward, although there are caveats. These include:

1. The meaning of JSON-LD is defined its **context**.
2. JSON-LD expanded syntax may be much more *regular*, simplifying software that uses data.
3. The order of keys in an object is not significant, so don't rely on it.
4. Arrays may only appear as needed.
5. Order of data inside an array should not be considered signficant (as described by *Building JSON-LD APIs: Best Practices*, "Best Practice 6: Assume arrays are unordered")

Vanilla JSON processes that use JSON-LD are encouraged to carefully control the organization and JSON-LD context of the data. This may be done by using the expansion, compaction, and framing algorithms provided as part of the JSON-LD specification products.

Order of keys not significant

In JSON and JSON-LD, the order of keys within an object are *not* considered significant. This means that the two following pieces of JSON-LD are equivalent:

```
{
  "ns:key1": "value",
```

```
  "ns:key2": "value"
}
```

is equivalent to:

```
{
  "ns:key2": "value",
  "ns:key1": "value"
}
```

The order of keys within a JSON-LD instance may be determined by the writer of the JSON-LD, or may be handled automatically by a JSON-LD library.

Arrays may be omitted

In JSON-LD, a single object is equivalent to an array that contains a single object. So, the two following pieces of JSON-LD are equivalent:

```
{
  "ns:key": "value"
}
```

is equivalent to:

```
{
  "ns:key" : [
    "value"
  ]
}
```

A system that is generating JSON-LD could optionally generate the second form instead of the first.

Additional guidance

Expressing types

The `@type` keyword is used to associate a type with a node. The concept of a node type and a value type are different. A *node type* specifies the type of thing that is being described, like a Person, Location, or Event. A *value type* specifies the data type of a particular value, such as an integer, a floating point number, or a date. Using `@type` keywords is optional. In general, the NIEM IEPD uses XML Schema to define node types or to constrain value types so it would not be necessary for the JSON-LD to repeat all of that.

If you need to specify type information for handling the iep data correctly, you can add it where needed. This example specifies that the node type of Person in the iep is a NIEM PersonType and the value type of their PersonBirthDate is an xsd date type.

```
{
```

```
  "@context": {
    "nc": "http://release.niem.gov/niem/niem-core/3.0/#",
    "xs": "http://www.w3.org/2001/XMLSchema#",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    },
      "nc:Person": {
        "@id": "P01",
        "@type" : "nc:PersonType",
        "nc:PersonBirthDate": {
          "nc:Date": {
            "rdf:value": {
              "@value" : "1893-05-04",
              "@type"  : "xs:date"
            }
          }
        }
      }
    }
```

Here is an example of a typed literal, identifying the type of the simple content of `nc:MeasureDecimalValue` as being type `xs:decimal`:

```
{
  "@context" : {
    "nc": "http://release.niem.gov/niem/niem-core/3.0/#",
    "xs": "http://www.w3.org/2001/XMLSchema#"
  },
  "nc:MeasureDecimalValue": {
    "rdf:value": {
      "@value": "9.7",
      "@type" : "xs:decimal"
    }
  }
}
```

If you don't specify the value type as xs:decimal, it is interpreted by JSON as being of type http://www.w3.org/2001/XMLSchema#double and the value becomes "9.699999999999999E0".

See JSON-LD Section 6.4 Typed Values for more information on how to express typed values.

Linking contexts via HTTP headers

If you have existing JSON data that you want to expose as JSON-LD, you can do that by supplying a separate JSON-LD context for it. This provides an upgrade path for developers who need to be able to continue to support regular JSON. You do this by adding a HTTP Link Header as specified by RFC5988 using the `http://www.w3.org/ns/json-ld#context` link relation.

A separate context file could be served at URL `http://example.com/contexts/iepd-context.jsonld`:

```
{
  "@context": {
    "nc": "http://release.niem.gov/niem/niem-core/3.0/#",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    ...
    "givenName" : "nc:PersonGivenName",
    "additionalName" : "nc:PersonMiddleName",
    "familyName" : "nc:PersonSurName",
    "text" : "rdf:value"
  }
}
```

A link header can be added to the plain JSON response that includes:

```
Link: <http://example.com/contexts/iepd-context.jsonld>; rel="http://www.w3.or
```

The JSON-LD processor prepends the context to the JSON data and interprets the original plain JSON as JSON-LD. These definitions can then be used by a JSON data instance, without the instance explicitly referring to the context. Since the context defined the short terms as aliases to full NIEM terms, expanding this JSON data will convert it to use the full NIEM terms.

```
{
  "givenName": { "text" : "Peter" },
  "additionalName": [ { "text": "Death"},
                      { "text" : "Bredon" } ],
  "familyName": { "text" : "Wimsey" }
}
```

A benefit of separating the context from the JSON is that if you're storing the JSON data in a database you can add new terms to the JSON-LD context without having to replace the inline context in all the data that has already been stored.

See JSON-LD Specification Section 6.8, "Interpreting JSON as JSON-LD" for more information and examples of how to implement this.

Road map for future work

NIEM Conformance

1. Provide a normative specification on a way to generate a JSON-LD instance from a NIEM-conformant XML schema and/or XML instance, and such a JSON-LD instance could be NIEM-conformant.
2. Provide a normative specification on verifying NIEM-conformance of a JSON-LD instance to a NIEM-conformant XML schema

Support for transforming between XML and JSON

Describe approaches or tools for transforming.

Provide helper functions to check for repeatable elements.

References

- JSON-LD: Manu Sporny, Gregg Kellogg, Markus Lanthaler, Editors. 16 January 2014. W3C Recommendation. "A JSON-based Serialization for Linked Data." Available from https://www.w3.org/TR/json-ld/
- RDF-Concepts: Richard Cyganiak, David Wood, Markus Lanthaler, Editors. 09 January 2014. W3C Proposed Recommendation. "RDF 1.1 Concepts and Abstract Syntax." Available from http://www.w3.org/TR/rdf11-concepts/
- RDF Primer: Frank Minola, Eric Miller, Brian McBride, Editors. 10 February 2004. W3C Recommendation. 10 February 2004. "RDF Primer."
- RFC4627: D. Crockford. "The application/json Media Type for JavaScript Object Notation (JSON) (RFC 4627)." July 2006. RFC. Available from http://www.ietf.org/rfc/rfc4627.txt
- RFC5988: M. Nottingham. Web Linking. October 2010. Internet RFC 5988. Available from: http://www.ietf.org/rfc/rfc5988.txt

Full example:

The walkthrough of how to transform an XML instance document IEP into a JSON-LD data instance is demonstrated using the full example documents within this section. These examples were constructed to show many aspects and features of NIEM XML instance documents.

Full example: XML instance document

The following XML document is the XML form for the full example from Section 2, above.

```
<?xml version="1.0" encoding="UTF-8"?>
<exch:CrashDriverInfo
 xmlns:exch="http://example.com/CrashDriver/1.0/"
 xmlns:j="http://release.niem.gov/niem/domains/jxdm/5.1/"
 xmlns:nc="http://release.niem.gov/niem/niem-core/3.0/"
 xmlns:geo="http://release.niem.gov/niem/adapters/geospatial/3.0/"
 xmlns:gml="http://www.opengis.net/gml/3.2"
 xmlns:structures="http://release.niem.gov/niem/structures/3.0/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <nc:Person structures:id="P01">
        <nc:PersonBirthDate>
            <nc:Date>1893-05-04</nc:Date>
        </nc:PersonBirthDate>
        <nc:PersonName nc:personNameCommentText="copied">
            <nc:PersonGivenName nc:sequenceID="1">Peter</nc:PersonGivenName>
            <nc:PersonMiddleName nc:sequenceID="2">Death</nc:PersonMiddleName>
            <nc:PersonMiddleName nc:sequenceID="3">Bredon</nc:PersonMiddleNam
            <nc:PersonSurName>Wimsey</nc:PersonSurName>
        </nc:PersonName>
        <exch:PersonFictionalCharacterIndicator>true</exch:PersonFictionalCha
    </nc:Person>
```

```
    <j:Crash>
        <nc:IncidentLocation>
            <geo:LocationGeospatialPoint>
                <gml:Point gml:id="PT01" srsName="urn:ogc:def:crs:EPSG::4326">
                    <gml:pos>51.835 -0.417</gml:pos>
                </gml:Point>
            </geo:LocationGeospatialPoint>
        </nc:IncidentLocation>
        <j:CrashVehicle>
            <j:CrashDriver>
                <nc:RoleOfPerson structures:ref="P01" xsi:nil="true"/>
                <j:DriverLicense>
                    <j:DriverLicenseCardIdentification>
                        <nc:IdentificationID>A1234567</nc:IdentificationID>
                    </j:DriverLicenseCardIdentification>
                    <exch:LicenseAugmentation>
                        <nc:ItemLengthMeasure>
                            <nc:MeasureDecimalValue>9.7</nc:MeasureDecimalValu
                            <nc:LengthUnitCode>CMT</nc:LengthUnitCode>
                        </nc:ItemLengthMeasure>
                    </exch:LicenseAugmentation>
                </j:DriverLicense>
            </j:CrashDriver>
        </j:CrashVehicle>
    </j:Crash>
    <j:PersonChargeAssociation>
        <nc:Person structures:ref="P01" xsi:nil="true"/>
        <j:Charge structures:ref="CH01" xsi:nil="true"/>
        <j:JuvenileAsAdultIndicator>true</j:JuvenileAsAdultIndicator>
    </j:PersonChargeAssociation>
    <j:Charge structures:id="CH01" structures:metadata="MD01">
        <j:ChargeDescriptionText>Wild Driving</j:ChargeDescriptionText>
        <j:ChargeFelonyIndicator>false</j:ChargeFelonyIndicator>
    </j:Charge>
    <j:JusticeMetadata structures:id="MD01">
        <j:CriminalInformationIndicator>true</j:CriminalInformationIndicator>
    </j:JusticeMetadata>
</exch:CrashDriverInfo>
```

Full example: JSON data

The following JSON data is a compact JSON-LD form of the full example from Section 2, above.

```
{
  "@context": {
    "exch": "http://example.com/CrashDriver/1.0/#",
    "j": "http://release.niem.gov/niem/domains/jxdm/5.1/#",
    "nc": "http://release.niem.gov/niem/niem-core/3.0/#",
    "geo": "http://release.niem.gov/niem/adapters/geospatial/3.0/#",
```

```
    "gml": "http://www.opengis.net/gml/3.2#",
    "structures": "http://release.niem.gov/niem/structures/3.0/#",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  },
  "exch:CrashDriverInfo": {
    "nc:Person": {
      "@id": "P01",
      "nc:PersonBirthDate": {
        "nc:Date": "1893-05-04"
      },
      "nc:PersonName": {
        "nc:personNameCommentText": "copied",
        "nc:PersonGivenName": {
          "nc:sequenceID": 1,
          "rdf:value": "Peter"
        },
        "nc:PersonMiddleName": [
          {
            "nc:sequenceID": 2,
            "rdf:value": "Death"
          },
          {
            "nc:sequenceID": 3,
            "rdf:value": "Bredon"
          }
        ],
        "nc:PersonSurName": "Wimsey"
      },
      "exch:PersonFictionalCharacterIndicator": true
    },
    "j:Crash": {
      "nc:IncidentLocation": {
        "geo:LocationGeospatialPoint": {
          "@context": {
            "srsName": "http://www.opengis.net/gml/3.2#srsName"
          },
          "gml:Point": {
            "@id": "PT01",
            "srsName": "urn:ogc:def:crs:EPSG::4326",
            "gml:pos": "51.835 -0.417"
          }
        }
      },
      "j:CrashVehicle": {
        "j:CrashDriver": {
          "nc:RoleOfPerson": {
            "@id": "P01"
          },
          "j:DriverLicense": {
            "j:DriverLicenseCardIdentification": {
```

```
              "nc:IdentificationID": "A1234567"
            },
            "nc:ItemLengthMeasure": {
              "nc:MeasureDecimalValue": 9.7
            },
            "nc:LengthUnitCode": "CMT"
          }
        }
      }
    },
    "j:PersonChargeAssociation": {
      "nc:Person": {
        "@id": "P01"
      },
      "j:Charge": {
        "@id": "CH01"
      },
      "j:JuvenileAsAdultIndicator": true
    },
    "j:Charge": {
      "@id": "CH01",
      "j:JusticeMetadata": {
        "@id": "MD01",
        "j:CriminalInformationIndicator": true
      },
      "j:ChargeDescriptionText": "Wild Driving",
      "j:ChargeFelonyIndicator": false
    }
  }
}
```


RDF

The NIEM conceptual model is based on the RDF data model described in RDF-Concepts. NIEM defines a mapping from the XML components in IEPs and IEPD schemas to equivalent RDF triples. As of this writing, there is no automated translator for that mapping.

Another reason for choosing JSON-LD is that it is a concrete RDF syntax as described in RDF-Concepts. There is a mapping between JSON-LD objects and equivalent RDF tuples. This mapping is implemented in open-source translators. This means that the JSON-LD serialization of an IEP can be automatically translated to Turtle or RDF/XML and processed in that form.

This establishes two paths from a NIEM IEP to RDF

- Direct, using the mapping in the NDR
- Indirect, first following the guidance in this document to produce a JSON-LD serialization, then converting that JSON-LD to RDF

The NTAC intent is that the RDF created by both paths will be consistent. This may entail future revisions to the NDR.

Community Activities

- Biometrics Domain
  A dedicated group that supports information sharing and promotes interoperability between mission-based organizations engaged in activities such as homeland security, national defense, border management, immigration benefits, and global law enforcement through the joint development and alignment of Extensible Markup Language (XML) Biometric Standards.
  Learn more
- A common understanding of grant data
  There are approximately 24 federal agencies that manage grants ranging from health to disasters to education and many other issue areas. A team within the NIEM community is working to identify the common terms, standardize them, and harmonize/unify the grants management systems.
  Learn more
- Health Care and NIEM
  The NIEM Health Community's primary objective is to identify ways to use existing health IT standards to meet the needs of the NIEM community, all while providing education and awareness about the specific requirements for protecting sensitive health data.
  Learn more
- State, Local, Tribal, and Territorial Tiger Team
  A dedicated group to advance information sharing at the state, local, tribal, and territorial level to help protect, support, and respond to community needs.
  Learn more

Biometrics Domain

The NIEM Biometrics Domain (NBD) is a data model of agreed upon terms, definitions, and formats. It supports information sharing and promotes interoperability between mission-based organizations engaged in activities such as homeland security, national defense, border management, immigration benefits, and global law enforcement through the joint development and alignment of Extensible Markup Language (XML) Biometric Standards. The NBD was launched in July 2012 and functions under the stewardship of the Office of Biometric Identity Management (OBIM) within the Department of Homeland Security (DHS). OBIM transitioned to the DHS Management Directorate after passage of the Cybersecurity and Infrastructure Security Agency Act of 2018.

OBIM's Assistant Director, John Boyd, is the Chair of the of the NBD. As a Domain Steward, OBIM's role is to advance the Biometrics Domain of NIEM by providing active support to the NIEM community and collaborating with Domain Co-Chairs including the Department of Defense (DoD) and Department of Justice (DOJ) with National Institute of Standards and Technology (NIST) as the Ombudsman which forms the NBD Executive Committee (NBDEC). OBIM manages all domain related activities, including biometrics schema development, harmonizing the schema elements across the NIEM domains, and we also contribute to the community of interest (COI) through outreach. OBIM provides input to the NIEM Business Architecture Committee (NBAC) updates, supports NBAC harmonization, and the core synchronization processes for inclusion in future NIEM releases.

OBIM contributes to NIEM Technical Architecture Committee (NTAC) as a member and provides input during the review and editing of various NIEM technical specifications. We participate in tiger teams like the Information Exchange Package Documentation (IEPD) Registry, and the Emerging Technologies Tiger Team (ET3) as well.

- Biometrics Executive Committee
  Learn more
- Biometrics Newsletters
  Learn more
- Biometrics Working Groups
  Learn more
- Biometrics Community Documents
  Learn more

- Biometrics Membership/Feedback
  Learn more

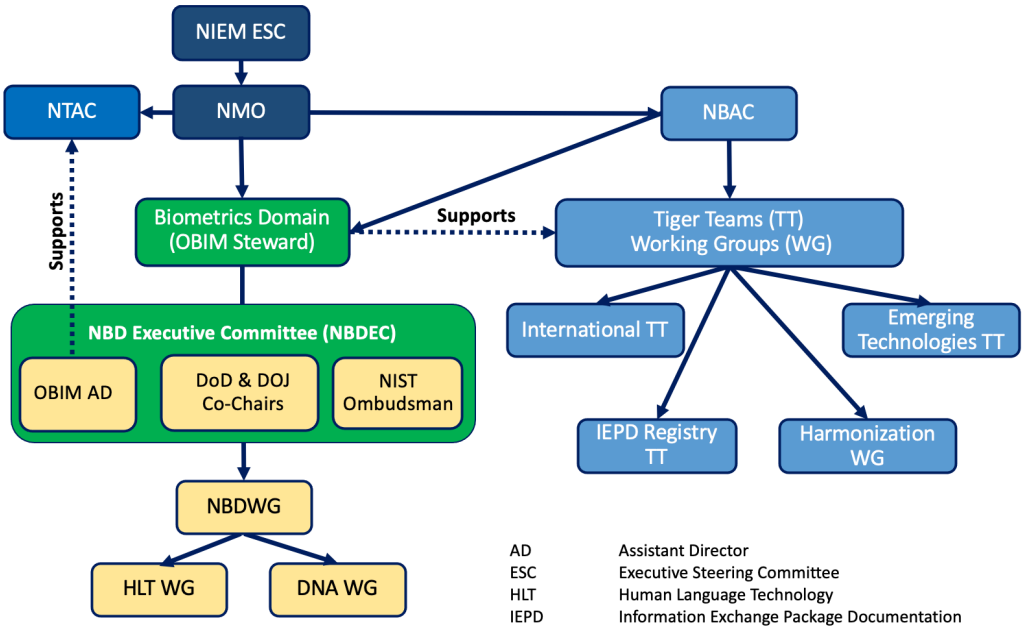
Biometrics Executive Committee

Executive Committee

OBIM is the Domain Steward and works with the NBDEC members which include the Biometrics Domain Chair, two Co-Chairs (DoD and DOJ), and the National Institute of Standards and Technology (NIST) as Ombudsman to provide guidance to the Biometrics Domain and the community.

The NBDEC communicates the information concerning the Biometrics Domain and associated issues of domain management to the NIEM Management Office (NMO) and NIEM Business Architecture Committee (NBAC) to apprise stakeholders of developments and activities.

NBDEC	Role	Organization
John Boyd	Chair	DHS OBIM
Jennifer Stathakis	Co-Chair	DoJ Criminal Justice Information Services (CJIS)
William Graves	Co-Chair	DoD
Diane Stephens	Ombudsman	NIST

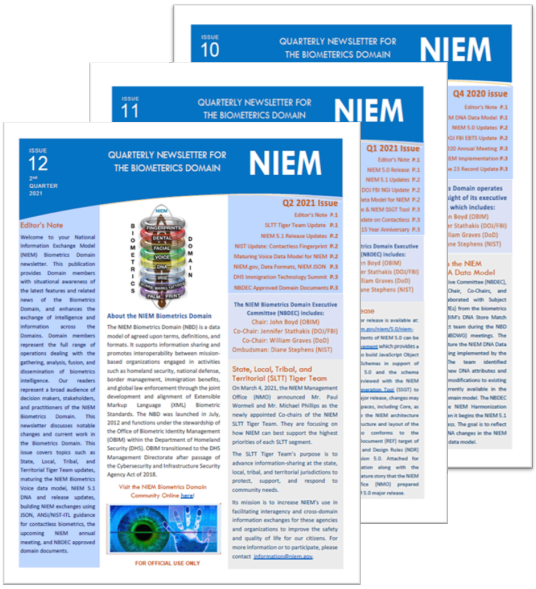


AD	Assistant Director
ESC	Executive Steering Committee
HLT	Human Language Technology
IEPD	Information Exchange Package Documentation
NBD	NIEM Biometrics Domain
NBAC	NIEM Biometric Architecture Committee
NBDWG	NIEM Biometrics Domain Working Group
NMO	NIEM Management Office
NTAC	NIEM Technical Architecture Committee

Please contact the NIEM Biometrics Team to become a member and receive updates and newsletters.

Biometrics Newsletters

Newsletters



The NBD produces Quarterly Newsletters to support the community. The publication provides Domain members with situational awareness of the latest features and related news of the Biometrics Domain, and enhances the exchange of intelligence and information across the Domains. Domain members represent the full range of operations dealing with the gathering, analysis, fusion, and dissemination of biometrics intelligence. Our readers represent a broad audience of decision makers, stakeholders, and practitioners of the NIEM Biometrics Domain.

Time Period	Date Posted
Q2 2021	Newsletter Q2 2021
Q1 2021	Newsletter Q1 2021
Q4 2020	Newsletter Q4 2020
Q3 2020	Newsletter Q3 2020
Q2 2020	Newsletter Q2 2020
Q1 2020	Newsletter Q1 2020
Q4 2019	Newsletter Q4 2019
Q3 2019	Newsletter Q3 2019
Q2 2019	Newsletter Q2 2019
Q1 2019	Newsletter Q1 2019

Please contact the NIEM Biometrics Team to become a member and receive updates and newsletters.

Biometrics Working Groups

- Working Groups
- Human Language Technology
- DNA
    - DNA Record Structure
    - DNA Match
    - Pedigree
    - Height Open Source Project

Working Groups

The NEIM Biometrics Domain (NBD) has a standing working group, the NBDWG. The Chair, Co-Chairs, and Ombudsman of the NIEM Biometrics Domain participates in the NBDWG and the meetings are specific to potential NBD model updates. The Co-Chairs lead the NBDWG based on their respective need for potential NBD model updates, or for updates that align with ANSI/NIST-ITL updates.

The working group is administered by the DHS OBIM who is domain steward in conjunction with the NIEM Biometrics Domain Executive Committee (NBDEC), which communicates issues and resolutions among the working group, the NIEM Management Office (NMO), and the Biometrics Community of Interest (COI).

Changes to the NBD model are first confirmed and approved by the NBDWG before making it into the NIEM Domain release.

Human Language Technology

DHS OBIM is developing an Human Language Technology (HLT) prototype and utilizes the NIEM Biometrics Domain Working Group (NBDWG) to bring together DHS, DoD, DOJ, HLT SME's, NIEM, and NIST to mature the voice standard. The goal of this effort is to perform an initial gap analysis by mapping the HLT subject areas to NIST Type 11 and the current NIEM Biometrics Domain which will feed into the later stages of the Map and Model phase of the NIEM Information Exchange Package Documentation (IEPD) lifecycle.

If you are interested in collaborating in this effort, learning more information about it, or other

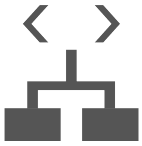
NBDWG discussions, please contact [the NIEM Biometrics Team](#).

DNA

DHS OBIM has developed a DNA prototype and utilized the NIEM Biometrics Domain Working Group (NBDWG) to bring together DHS, DoD, DOJ, DNA SME's, NIEM, and NIST to mature the DNA standard.

Outcomes of the NBDWG DNA effort including updates to NIEM 4.2 and 5.1 RC, some of which are described further below. Continued efforts are ongoing which may lead to updates to NIEM 5.2.

DNA Record Structure



- Restructuring DNA record so that it contains one to many DNA Typing Technology Data.
- Sample information should be optional metadata to DNA Typing Technology Data.

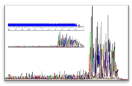
DNA Match

- Modify DNA match data structure to include the fields required for kinship matches.

Pedigree

- To have a separate type in the ANSI/NIST ITL Biometric Data Exchange Standard is recommended for exchanging the embedded pedigree information.
- Additional fields for pedigree information are also recommended to streamline the communication of complex familial relationships.

Height Open Source Project

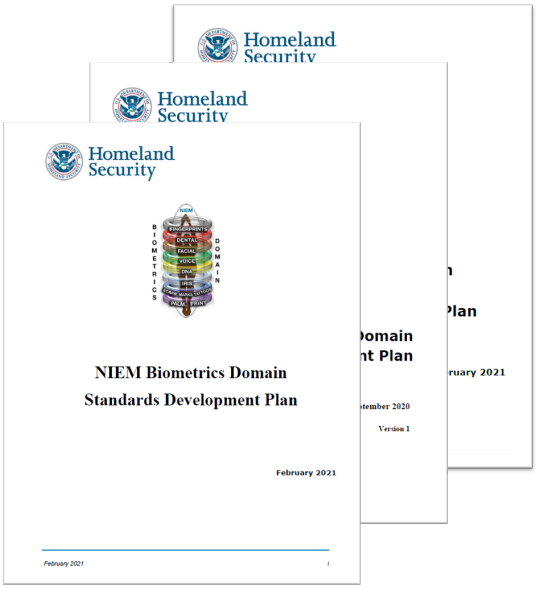


- Large peak height values must be “capped” at 32,767 in a .fsa file causing loss of data so the group discussed alternative options. Currently working to set up an open source project to collaboratively develop the “.fsax” data format to meet DNA Vendor requirements.
- The NBD team worked with the NIEM Management Office (NMO) to stand up a GitHub repo for the open source project to collaborate between NIST, NIEM, SME’s, and DNA Vendors to design the .fsax format to improve Peak Height representation in NIEM and American National Standards Institute (ANSI)/NIST-Information Technology Laboratory (ITL).

If you are interested in collaborating in this effort, learning more information about it, or other NBDWG discussions, please contact the NIEM Biometrics Team.

Biometrics Community Documents

Community Documents



OBIM supported the NIEM Biometrics Domain by developing biometrics domain related documents to benefit Biometrics community. The below items are available for download.

NIEM Document	Description
Governance Process Plan	The plan documents the governance processes related to release management and publication including a planned schedule, a unified strategy for Domain development and maintenance, as well an establishment of roles and responsibilities of the COI members as defined in the NIEM High Level Version Architecture (HLVA).
How-To Guide for Creating NIEM Biometric Exchanges	This details the process for creating NIEM exchanges necessary for Biometrics Domain development and maintenance. It also identifies a variety of support processes including IEPD development that are relevant to successful interoperability of NIEM exchanges.
Standards Development Plan	The plan contains information pertaining to the NIEM structure and future architecture. The plan also addresses how to ensure that the development of NIEM Domain content minimally disrupts OBIM architecture and maximizes the efficiency and effectiveness of OBIM plans and execution.
Enterprise Level Data Standards Execution	The plan covers the need, development, adoption, implementation, and maintenance of data standards required for the efficient interoperability between disparate systems across the federal government.
Domain Management Plan	The plan describes the approach to support the day-to-day management and operations of the Biometrics Domain and details of support necessary to provide data standards services for Biometric Domain management and a baseline of the biometrics domain.

Biometrics Membership/Feedback

Membership/Feedback

Domain members can contribute to efforts in the Biometrics domain including formal training, domain related best practices, working group participation and collaborative review and creation of biometric standards, data exchanges, information models, and code management support.

Biometrics domain standing working groups drive solutions on technical and business issues and tiger teams attend to pressing needs within the COI. Technical assistance and support includes a review of Information Exchange Package Documentation (IEPDs); Biometrics-focused NIEM training for organizations and at conferences, and organizational system and policy alignment.

Please contact the NBD Team the NIEM Biometrics Team if you are interested becoming a member.

A common understanding of grant data

There are approximately 24 federal agencies that manage grants ranging from health to disasters to education and many other issue areas. Each of these agencies may use one or more grant management systems. These systems use terms such as “grantor,” “grantee,” “award process,” “adjudication,” etc.; the definitions of these terms (and the terms themselves) may vary greatly between systems.

- Data Definitions for Grants
- Conceptual Model
- Learn More About NIEM

A team within the NIEM community, led by the NBAC co-chair Kamran Atri, is working to modernize grants management within a federal agency. They are identifying the common terms, standardizing them, and harmonizing/unifying the grants management systems. This is where NIEM comes.

To date, the team has reviewed over 34,000 grant-related data fields scattered across several government systems and narrowed the data fields down to approximately 3,200. This set has been further edited down and categorized into 114 of the most common data subjects. Of the 114 data subjects, some have been down-selected for consideration as common core for grant data.

The list of data definitions and the conceptual model on this page provide the data subjects that were reviewed with the aim of building initial consensus.

The team is ready to share their work to advance the grant reporting need as well as help other grant stakeholders work through a unified structure for building standardized data exchanges. If you are interested in joining a tiger team to work with stakeholders on creating a NIEM grant domain, please email Kamran Atri.

Data Definitions for Grants

To provide feedback, please see our related issue.

Entity Name	Definition
ADDRESSES	Addresses is the subject area collection of Address information for applicants, organizations and recipients. Address information is collected for physical location and mailing purposes.
ALLOCATIONS	Allocation is a financial transaction in which funds for a grant program or a grant become available. Establishes amount of funds that can be obligated for a project.
AMENDMENTS	The collection of information detailing post-award changes to be incorporated into a Grant or project.
APPEALS	Appeals are the formal requests from the applicant or recipient /sub-recipient that have been made for Disaster and Project related determinations. Substantiation appeals may be related to expense, income, lease or rentals. Project appeals may be associated with funds or expense payments.
APPLICANTS	Applicant is the non-Federal (Public Entity or Non-Profit), commercial organization or individual requesting mitigation or disaster assistance through

Entity Name	Definition
	one of the Federal grant programs.
APPLICATIONS	Applications is the subject area collection of required information for requesting assistance. Pre-Applications, Grant Application, Subgrant Application are types of Application.
ASSURANCES	Assurance is a listing of agreements to which applicants certify to maintain and uphold in writing before they are deemed eligible for grant funding.
AUDITS	Audits are reviews that occur after an Award is approved and reviews the details of contracted work, compliance, financial and performance efforts.
AUTHORITIES	The statutes, regulations and policy guidance used by federal agencies and States to establish processes for managing Disasters, Grants, Programs and Projects.
AWARD PACKAGE	Stores the information provided to a grantee upon award of their grant, including: An award letter, Signed application and Agreement Articles.
AWARDS	Awards is the phase of the Grant lifecycle consisting for determining and approving awards and notifying applicants of the award decision.
BUDGETS	The process by which programs and projects create and manage approved funds and expenditures within a fiscal timeframe. Federal Cost Share is the percentage of total application budget costs that will be provided by agency funding.
CASE MANAGEMENT	Case management is the activity used to manage applicant information, correspondence log, address book, projections and kickoff meeting notes.
CLOSEOUTS	Closeouts is the phase of the Grants life-cycle containing a set of actions conducted to verify and conclude an awarded application, project or Grant's activities and funding.
COMMITMENTS	Commitments is the collection of Commitment information. Commitment is a financial agreement or pledge of assistance for funding.
COMMUNICATIONS	Interactions between Grant Management and Customers or other Agencies for information provided by grant programs regarding disasters, emergencies and assistance efforts or responses from Customers or Providers.
CONTACTS	Contacts is the collection of Contact information. Contact information identifies individual persons or departments as a designated first contact source to an agency, organization or stakeholder.
CONTRACTORS	Contractors provide resources to programs and projects.
CUSTOMERS	Information about people and organizations (survivor, business, whole community, state/local government, and Indian tribal) affected by disaster/ incident, therefore receiving assistance from the providers or grantors.
DATA EXCHANGES	Information sent to or received from internal and external partners by various program functions using the Internet or Network connections.
DISBURSEMENTS	Disbursements are records of payments made to Sub-recipients by Recipients to reimburse them for work done on obligated projects. Financial Office makes disbursement payment to Recipient through Treasury.
ELECTRONIC MESSAGES	Information (emails, faxes, notices) sent to an Individual or Organization by various program functions using the Internet.
ELIGIBILITY DETERMINATIONS	Eligibility reviews and evaluations are required for applications and programs. Applications are scored against criteria defined by disaster and nondisaster grant programs.
FILES	Files contain associated information for a particular grants specific document

Entity Name	Definition
	and attachments, import and export information, backups and archivals needed for sending, receiving and processing grants related information.
FUNDING OPPORTUNITY	Funding Opportunities are the collection of Notice of Funding Opportunity announcement that applications for a federal grant may now be submitted.
FUNDS	Fund is the approved/appropriated collection of monies to be used to support portfolio and program execution. Financial resources allocated to or for a specific task.
GEOGRAPHIC LOCATIONS	Geographical Location is information about a designated physical geographic area within the business service, tribal or Non-US areas.
GEOSPATIAL LOCATIONS	Geospatial location is a position of things relative to the Earth's surface. Geospatial Locations subject area contains all Geospatial related entities and is based on capturing Latitude and Longnitude attribute usage.
GRANTS	Grants are the primary funding mechanism for building and sustaining national preparedness capabilities, assistance for rebuilding or mitigation efforts and relief funds for disaster survivors. Each grant has specific eligibility criteria for applicants to meet.
INSPECTIONS	Inspections may be required for Sub-recipient's facilities or for Registrants' real and personal property by assigned inspectors in order to receive certain types of assistance or confirm work completion and compliance.
INSURANCE INFORMATION	Insurance Information is any insurance policy and/or coverage that may apply to an item for which a grant is requested. The registrant's or applicant's insurance policy for their vehicle(s), personal property, real property or site/facility.
OBLIGATIONS	An obligation of funds is a financial transaction to disburse funds immediately or at a later date as a result of a series of actions. All of these actions must occur to obligate funds for the formula-based grant programs authorized by the Acts.
OFFICIAL CORRESPONDENCE	Official Correspondence (Letters and Notifications) sent to an Individual or Organization by various program functions using the US Mail Service or the Internet.
PAYMENT REQUESTS	Payment Request is a request for awarded funds and may include invoice details and specific line of accounting codes to apply charges or credits to. Information regarding requests for reimbursement of funds spent by Recipients and Sub-recipients on obligated projects.
PHONE CALLS	Phone calls is the collection of Cellular and landline communications with individuals, organizations or other resources in support of assistance efforts.
PLANS	Plans is a collection of Information specific to a project, program or Grant that details the goals, objectives and milestones.
POST-AWARDS	Post Award is the phase of the Grant lifecycle consisting of executing grant and subgrant award documents, funding, and monitoring. This phase is the operational process of executing grant award documents and obligations and releasing funds for recipient use.
PROGRAMS	Program is a grouping of predefined tasks and procedures managed in a coordinated way in response to a specified type of disaster or future hazard.
PROJECT COST ESTIMATES	Cost estimate Information supporting the eligibility requirements of a project, program or grant.
PROJECTS	Projects is the collection of information supporting project scoping, project setup, and project management.

Entity Name	Definition
PROPERTIES	Properties (real or personal property) owned by an individual or organization that may be impacted or damaged by a disaster or require mitigation assistance.
RECIPIENTS	Recipient/Awardee is the non-Federal (Public Entity, Private-Non-Profit, or Commercial organization or Individual) receiving mitigation or disaster assistance through a grant program.
REPORTS	A collection of information required by programs and awards to be gathered on a periodic basis.
REQUESTS	A request is submitted by a Recipient or Sub-recipient for changes to a Grant or Sub-grant, award, funds or disbursement.
RESOURCES	Resources include the people, materials, knowledge, studies, technologies, and money required to develop plans or implement mitigation strategies.
REVIEWS	Reviews are the activities that contain the review information provided by the various reviews at various grant lifecycle stages including Pre-Award, Award, Post-Award and Closeout.
SITES	Site is one or more facilities that were damaged by a disaster or require mitigation in some way.
STATES	Governmental entities and Tribes requesting and administering grants. Includes the 50 states identified by Federal Information Processing Standards (FIPS) codes along with their names and associations.
SUBGRANTS	Subgrants is the collection of information supporting subawards. A Subaward is provided by the pass-through entity to a Sub-recipient to carry out part of the Federal award received by the pass-through entity.
SYSTEMS AND SERVICES	Systems and services is the technology involving the development, maintenance, and use of computer systems, software, data stores, and networks for the processing and distribution of Grants data.
TECHNOLOGY	Technology is the hardware infrastructure, software processes and delivery services used to support business activities and the supporting data for each business activity.

Conceptual Model

The conceptual model is currently too large to display on this page. View the conceptual model.

Learn More About NIEM

- What is NIEM?
- The Value of NIEM

- View the NIEM model content using Movement, a user-friendly open source search tool.

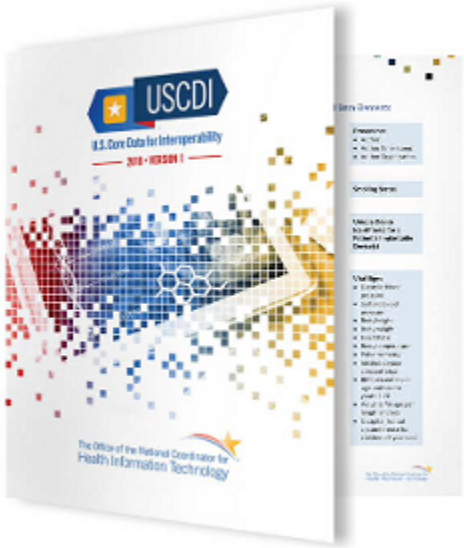
Health Care and NIEM

- The NIEM Health Community of Interest Overview
- Overlap with other NIEM Domains
- NIEM Health Community Goals
    - The NIEM Health Community of Interest's objectives
- Get Involved!

The NIEM Health Community of Interest Overview

With the adoption of the National Information Exchange Model (NIEM) at all levels of government for use cases ranging from immigration to intelligence, there's a growing and unmet demand to exchange clinical health elements outside the typical hospital/clinician setting. For example, domains such as justice, human services, screening and emergency management are finding a greater need to standardize how they handle healthcare elements and information. However, existing health information technology (IT) standards are distinctly different from those found in the NIEM.

> Federal organizations, led by the **Federal Health Architecture (FHA)**, are working to formalize the relationship between NIEM and commonly used health IT standards.



For the past two years, FHA has evaluated the NIEM community's need for health data, codified the specific information needed for the Health Community of Interest , then studied existing health IT standards and their associated data elements in order to determine which of the existing standards the NIEM community can use to help meet their need for health data. This mapping effort provided FHA with clear insight into which standards to draw upon, and as importantly, what gaps still exist that need to be filled in order to satisfy the NIEM health data requirements.

This work will result in a canonical and authoritative mapping that aligns NIEM with the standard fields, value sets and codes used in clinical standards as required by US Common Data Interoperability (USCDI) 2018. These fields, value sets, and codes are defined by Health Level Seven's (HL7) Consolidated-Clinical Document Architecture (C-CDA), Fast Healthcare Interoperability Resource (FHIR), the International Health Terminology Standards Development Organization's (IHTSDO) SNOMED, the National Library of Medicine's RxNorm, Regenstrief's LOINC, the International Statistical Classification of Diseases and Related Health Problems (ICD) ICD-10, and others.

Overlap with other NIEM Domains

Multiple NIEM domains may require access to health information. The NIEM Health Community of Interest aims to ensure standards are in place to make it possible to share health data efficiently and securely.

The specific domains that most need access to health data include:

- Emergency Management
- Human Services
- Justice
- Military Operations
- Screening

NIEM Health Community Goals

The NIEM Health Community's primary objective is to identify ways to use existing health IT standards to meet the needs of the NIEM community, all while providing education and awareness about the specific requirements for protecting sensitive health data [Health Insurance Portability and Accountability Act (HIPAA), the Federal IT Acquisition Reform Act (FITARA), etc.].

The NIEM Health Community of Interest's objectives

1. Catalog and prioritize identified use cases for exchanging health related data outside the healthcare system;
2. Determine health IT standards/frameworks most relevant to high priority exchanges and seek alignment with the NIEM exchange development process;
3. Inventory existing NIEM health data elements;
4. Model and map required health IT standard data elements to existing or new NIEM health elements;
5. Support a community of interest around the sharing of health data with non-traditional partners.

Through its efforts to harmonize standards and build relationships between the NIEM and clinical health communities, the NIEM Health community of interest will have a lasting impact by making health data available securely to a broader range of organizations that need to the data to better serve the American people.

Get Involved!

The Health Community of Interest is sponsored by the U.S. Department of Health and Human Services (HHS).

Point of Contact: Robert Tagalicod

NIEM Health COI Meeting Minutes

- Meeting Minutes - August 6, 2019

  Future of NIEM Health COI

- Meeting Minutes - April 9, 2019

  Need for Health IT in Disaster Response; Patient Unified Lookup System for Emergencies (PULSE)

- Meeting Minutes - March 22, 2019

  NIEM in Canada; Information Exchange Framework Applied to Structured Data

- Meeting Minutes - January 29, 2019

  General Update; Protecting Health Information

- Meeting Minutes - February 13, 2018

  Health IT, NIEM Health Recap, and Status Update

- Meeting Minutes - October 10, 2017

  FHA, FHIM, and NIEM alignment

Meeting Minutes - August 6, 2019

- NIEM COI Meeting
- Agenda
- Slides
    - FHA Sunset
    - The Future of NIEM Health Interoperability
    - The Future of NIEM Health
    - NIEM Health Deliverable Complete
    - Transition Outreach
    - Lesson Learned
    - NIEM Health Crosses Multiple Domains
    - Recommended Scenarios
    - COI Lead Responsibilities
    - Transition Next Steps
- Q&A Plus Closing Comments
- Upcoming Events
- Action Items

NIEM COI Meeting

Hosts / Presenters:

- Avinash Shanbhag (FHA)
- Cait Ryan (NIEM)
- Brian Handspicker (NIEM)
- Ryan Schultz (NIEM MO)

Participants:

- Avinash Shanbhag (ONC/FHA)
- Sherilyn Pruitt (ONC/FHA)
- John Forrester (FHA)
- Cait Ryan (NIEM)
- Brian Handspicker (NIEM)
- Linda Self (NIEM MO)
- Matt Endreny (NIEM MO)
- Ryan Schultz (NIEM MO)
- Unknown Number (NIEM MO)
- Chris Travers (ACF)
- Nikolay Lipskiy (CDC)
- Suzanne Hassell (Johns Hopkins Univ)
- Jean Hall (Kentucky CHFS/PMIE)
- Diane Graski (NCSC)

Length: 1:05:48 minutes

Venue: Zoom

Agenda

- Welcome and Roll Call (Cait Ryan)
- FHA Sunset (Avinash Shanbhag, Sherilyn Pruitt)
- NIEM MO Remarks for NIEM Health (Ryan Schultz)
- Future of NIEM Health COI (Cait Ryan, Brian Handspicker, All)
- Wrap Up and Next Steps (Cait Ryan)

Slides

FHA Sunset

Avinash Shanbhag

- Avinash clarifies the Office of the National Coordinator for Health IT (ONC) is a staff division within Health and Human Services (HHS)
- Avinash states he will share:
    - The background of the NIEM project including:
        - Hopes for the future of NIEM
        - Activities the NIEM MO is conducting to support the future of the NIEM Health domain
        - Opportunity for Q&A with him and ONC colleague Sherilyn Pruitt
- The Federal Health Architecture (FHA) program electorally funds work supporting the NIEM COI
    - This is the connection between the FHA and the COI
- The FHA was founded as an OMB government line of business
    - This is an activity directed by OMB as the lead agency within HHS and all things regarding Health IT coordination in terms of technology standards and policy
- ONC is the managing partner for FHA
    - Along with HHS, the FHA is funded by the DoD, VA, the main office of HHS, and the SSA
    - The FHA was created in 2004
    - Including NIEM, the FHA funded a substantial number of projects targeting Health IT Interoperability at the national level
    - In early 2018, the FHA Managing Board decided to sunset FHA
        - At that time Avinash Shanbhag and colleague Sherilyn Pruitt joined ONC/FHA and were tasked by the ONC Managing Board to “deliberately and methodically” start transitioning the various programs FHA was funding
    - The sunset of FHA will take place in September 2019 for these reasons:
        - That landscape changed as technologies evolved and new entities arose:
            - Interagency programs such as ONC now exist
            - New agencies and offices that better coordinate federal health IT programs are now established
- NIEM COI is one of the key programs funded by the FHA over the past 2.5 years
- For the past year, the FHA, along with the support of their contracting team led by Cait Ryan of IRIS Health Solutions and Technical Architect Brian Handspicker, worked on a two-pronged

approach:
    - Identify new stewards for the COI by:
        - Identifying use cases not currently handled by existing Health IT standards
            - There are many Health IT standards and a Standards Development Organization called HL7 (Health Level 7) that deal with health information and interoperability
            - NIEM has a clear value proposition in edge-cases Cait and Brian worked with the COI to identify:
                - The idea over the past year was to determine if there were valuable use cases identifying appropriate stakeholders and/or agency lead/s who would take on the stewardship of the NIEM COI after the FHA sunsets
    - Make sure COI activity, management, and administration are done in a way that support to keep it going is kept to a minimum
        - Make sure the "lift" and resources needed by federal agencies who may take on stewardship would be kept to a bare minimum
        - Identify key use cases to make sure future stewardship of the COI and possible NIEM Health Domain would require only modest effort
- ONC funded NIEM in the past and looked at internal appropriations to determine if it could continue supporting NIEM but determined:
    - The constraints of funding limited staff
    - The fact that ONC's limited staff must support the work, including "the purest rule" that has taken on a lot of ONC's internal efforts, dictates ONC cannot take on the work
- Looking at the work Cait and Brian have done, there are many use cases where NIEM can be used
- Contract support for NIEM ends September 9, 2019
    - The NIEM COI is useful and should continue
    - Due to the program sun setting, it is important the critical work that has been done by the FHA continues
    - Avinash asks anyone interested to please reach out as there is still time to transition
- August 15, 2019: NIEM Symposium hosted by the NIEM MO

The Future of NIEM Health Interoperability

Ryan Schultz

- Ryan emphasizes the importance with which the NIEM MO views the NIEM Health COI
    - Vital work has been done
    - The work is recognized within the NIEM community and leadership
    - Many have expressed the importance of NIEM Health Community
    - Ryan invites discussion to explore what "that way forward might be"
- NIEM Symposium Agenda and Overview:
    - Welcome and introductions
    - NIEM and NIEM Health Historical Overview
    - The Future of NIEM Health
        - Establishing NIEM Health's Leadership
        - Collaboration Opportunities
        - Prioritizing Initiatives
        - Funding Models

        - NIEM Health Moving Forward: Next Steps
        - NIEM Health Scenarios
        - Wrap-up
        - Adjourn
    - The NIEM Health Symposium will be held in Rosslyn, Virginia
        - DC participants can take the metro
        - There will be dial-in capacity for remote/virtual participants
        - The Symposium registration URL is shown
    - The Symposium will be thought provoking on the value of the Health Community and what it does for interoperability and will explore:
        - The future of NIEM Health
        - What can be worked on and the kinds of things NIEM can contribute
        - How the Symposium can be used as a transition discussion with the larger stakeholder community of NIEM and NIEM Health
        - Those who have raised their hand to talk through various scenarios of how NIEM Health can contribute to interoperability within their area
- Brian Handspicker asks if virtual attendees will have capability to share their screen
    - There will be some combination of a WebX, Zoom, etc.
    - Safest to submit slides to NIEM MO
    - If material needs to be presented, NIEM MO will make sure it gets shared
        - Let NIEM MO drive the slides locally and someone can speak to them over the phone or WebX
- Ryan offers teaming-up with NIEM MO so NIEM MO can do some of the "heavy lifting"
    - It would be inappropriate for NIEM MO to be the lead to represent the health interests
    - A lot can be accomplished as a team if somebody will raise their hand
    - Brian Handspicker explains if multiple agencies/organizations are interested in supporting the NIEM Health COI there are solid examples of governance systems to support and quickly stand up to that kind of collaboration
        - Example: Mill Ops domain is a tri-chair with three organizations that have come together
            - It is not as resource-intensive as a whole, but everybody is doing what they would do naturally from an organizational responsibility perspective
            - Joining forces eases the burden for all
- Nikolay Lipskiy from the CDC Office of Public Health Preparedness and Response expresses strong support for NIEM
    - Specific interest is to support cases for public health emergency preparedness response
    - Wants to talk to someone interested in a collaborative effort
- Happy to support this specific use case or domain and very much wants to participate

The Future of NIEM Health

Brian Handspicker

- Brian Handspicker states work on proof of concepts has begun with the National Interoperability Coalition
    - There are a many people, especially in San Diego, interested in the emergency response aspect of NIEM Health

NIEM Health Deliverable Complete

Cait Ryan and Brian Handspicker

- Cait Ryan confirms the three promised deliverables providing education were delivered to the NIEM office in July 2019 and will be uploaded to the NIEM Health COI Page after NIEM editing and branding:
    - NIEM 101: An Introduction to Health Information Exchange (HIE)
    - NIEM 102: An Introduction to Security and Privacy of Protected Healthcare Information
    - NIEM 201: Architecting NIEM IEPDs Using Health Information Models
- Additional slide commentary:
    - Existing standards
    - How to look up, identify, and use data elements needed for information exchange package documents for IEPDs to align with existing health standards
    - A repeatable process is in place
    - If no one can take this work on immediately, once the documents are reviewed and edited by the NIEM MO for branding, they will be a valuable resource for people to create Health Information Exchanges for NIEM
        - The Team produced a good foundation for this work to build upon further
        - Transition outreach includes discussions with the NIEM MO who have helped send email communications, notify the NIEM Community-at-large about the sun setting of the FHA, and the leadership opportunity for the NIEM COI

Transition Outreach

Cait Ryan

- Additional transition efforts include:
    - Attending business and technical architect meetings
    - Reaching out to government POCs identified as a possible fit to determine their level of interest in continuing the work and dialogue from an FHA perspective
    - Holding one-off conversations with select COI participant organizations
    - Email communications in addition to the Fact Sheet sent with the meeting invitation

Lesson Learned

Cait Ryan and Brian Handspicker

- Additional transition efforts include:
    - More than 300 “health-related” elements already modeled within NIEM and various NIEM Domains
        - They do not align with HIE standards
        - Without a true repository with provenance and IEPD element indexing it is nearly impossible to decipher provenance and intent of elements
        - There appears to be no coordination among the domains to have a cohesive definition of what those NIEM “health elements” look like and how they are defined
    - Only a subset of clinical health elements is required

        - NIEM is not expected to be used for the exchange of significant clinical information
        - NIEM Health can be used for the exchange of specific healthcare information such as:
            - Community of Care Document
        - Edge-case Scenarios
            - The Team is exploring edge-case scenarios that are not the typical exchange provider-to-provider they, along with Health Information Exchange and HL7, tend to look at and remain driven by
            - The edge-case scenarios create a funding opportunity for multiple organizations
        - NIEM Health overlaps with, touches, and affects different Stakeholders and multiple domains reflecting a true COI with multiple layers
            - FHA had a difficult time identifying only one organization to take charge because NIEM affects so many people

NIEM Health Crosses Multiple Domains

Brian Handspicker

- Brian shares that he joined the NIEM initiative as a trained NIEM Architect who spent time developing HL7 standards
    - He was perplexed that they chose 72.5 0.1 for meaningful use:
        - If a prisoner is released into the community and their medical records need to be sent to a local doctor, why not use HL7 to do that from the infirmary to the doctor's office?
        - There is real use of health-related information being exchanged by NIEM
            - It is often being done in environments in which organizations don't have an Electronic Health Records or Healthcare Information Exchange system, but they do have NIEM systems which is why it is mandated by their agencies
        - How do we coordinate the healthcare information being exchanged?
        - How do we rationalize it with clinical healthcare standards?
        - How do we make sure this information can be reasonably exchanged between HL7 FHIR-based environments and NIEM environments?
            - Example: For the Justice Department, healthcare information is not the core of their domain
                - It is the same for MilOps, emergency management, and Human Services
        - It is not the center of what Human Services organizations do, but they still must collect foster children medical records
- NIEM Health Information has been at the periphery of each of the NIEM domains
    - From a clinical healthcare and HL7 FHIR perspective, exchanging information for natural disasters is important; however, it is not at the center of supporting clinical information exchange for oncology patients, an average pediatrician, or general practitioner, etc.
    - It is at the edge of both the clinical health domain and all NIEM domains
    - There are many domains that need to use some bit of healthcare-related information in order to do their job

Recommended Scenarios

Brian Handspicker

- Edge-case scenarios discovered thus far include scenarios actively using NIEM for support as well as envisioned scenarios:
    - Disaster Response
        - Earthquake and tsunami
        - Tsunami, Drought, and wildfire
    - Emergency Response
        - Multi-vehicle accident with trauma and burns
        - Building collapse requiring search and rescue
    - Public Health Reporting
        - Ebola-infected flight arriving
        - Vaccine causing wide-spread allergic reactions
        - Pharmaceutical recall
    - Medical Report Exchange
        - Foster/juvenile records collection
        - Wounded warriors on the battlefield
        - Reentry into the community after incarceration
    - Substance monitoring including
        - Substance abuse rehabilitation with effective supervision
        - Tracking opiate abuse and illegal distribution
        - Law enforcement poisoned by drugs during mission

COI Lead Responsibilities

Cait Ryan

- What the NIEM Team and FHA led for the past few years is a COI
    - No modeling was done to be a true domain
    - To satisfy FHA's role, this is a call to action for someone to stand up and lead the NIEM COI
    - Leadership requires a minimal level of effort
    - A solid foundation is in place with a repeatable process being published on the COI page
- Minimal effort leadership from a meeting perspective:
    - The Team currently attends monthly NIEM Business Architects Council meetings (invite only, one hour)
    - Weekly NIEM Technical Architects Council meetings (one hour)
    - A four-to-five-day annual Face2Face meeting at the Rosslyn office
        - If participating in NIEM Business Architects Counsel calls, the requirement is to attend the first two days which focus on end-back discussions
    - Monthly NIEM COI meetings introducing those who have use cases or scenarios that have proven successful in the past, or those looking for collaboration
        - Monthly as opposed to quarterly meetings generate the most participation and interest
        - New Steward/s will determine what works best for them and their resources
- Minimal effort leadership from an administrative perspective:

        - Periodic questionnaires from the NIEM MO
        - Updating the NIEM Health Community of Interest page currently on GitHub
        - Updating meeting materials and webinar logistics for upcoming meetings
        - Updating announcements, relevant documentation referenced during calls, or deliverables being produced
- If there are multiple agencies excited about NIEM MO, work can be done mapping a healthcare data requirements document called US CDI from HL7 standards into NIEM
    - Coordination across multiple domains ensures new healthcare-related elements defined are done in collaboration with what has already been done, as well as other projects that could be developed to further extend NIEM Health

Transition Next Steps

Cait Ryan

- Interested agencies are instructed to send a letter of intent to the FHA at www.federal.healthathhs.gov (or to Cait Ryan or Brian Handspicker)
- After approval the FHA will introduce to the NIEM MO for onboarding
    - The Team will set up GitHub account/access/permissions to COI page
    - The Team will populate the new Steward's calendar with necessary meetings
    - The Team will identify a point of contact
- Before the FHA sunsets, the Team will make sure everything on the NIEM COI page is up to date including:
    - Documents already produced will be archived, including:
    - Lessons learned
    - Recommendations for NIEM Health
    - Meeting minutes
- A joint, formal announcement will me made by NIEM, FHA, and
    - Ryan Schultz commits NIEM MO resources to help with transition and set up
        - NIEM MO has worked with many domains
        - NIEM MO knows what works effectively

Q&A Plus Closing Comments

All participants

- Ryan Schultz asks if anyone wants to meet with him and Nikolay Lipskiy to confer on what a collaborative effort between the NIEM MO, CDC, and another organization/s would look like
    - No attribution
    - No commitment
- Chris from ACF shares the challenge may be that FHA has been playing multiple roles best suited to be addressed at different levels
    - In terms of the use case approach:
        - There will be specific domains, communities, or agencies that have a role to play in defining exchanges based on the fact they are implicated in that exchange
        - From the standpoint of Human Services as part of the discussion on foster care health records, ACF expresses a desire to be involved and help support the

implementation of that exchange
        - At the Domain or COI level, Chris shares that it is an aggregation of agencies that are part of the different conversations and all interact with health
            - They have a role to play in defining the data set that becomes part of the NIEM framework supporting various exchanges
            - It is a multi-dimensional, multi-agency type of challenge
        - At the highest level, there is a role to play in terms of "thought leadership" to ensure appropriate health IT experts and implementers are part of the conversation
            - That is critical
            - That will enable potential leaders to have important conversations
            - That role is hard for people outside the health space to initiate
        - If ONC is not able to support the work financially or put resources behind it, it is essential someone step forward with the "health bonafide" to speak as a representative of the Community and bring the right people to the table
- Ryan Schultz shares what Chris described is a "healthy functioning, fully operable Health Community of Interest"
- Nikolai states, based on the model of NIEM that is the core part in the domains, there is a clear picture of how this core needs to be connected to healthcare and other domains
- Ryan reiterates that he wants to address the "clever thoughts" being shared in a discussion, devising a plan to put them in motion
    - Everything need not fit perfectly into place just yet
    - Recognize it is a process of transition
    - Discussion must focus on a "leadership way forward"
    - Ideas and comments are encouraged
    - Ideas and comments can be shared via email, phone call, or in-person
    - Ryan emphasizes non attribution, no commitment discussions
- Ryan shares his contact number, (757) 203-8605
    - Cait Ryan agrees to share Ryan's email address with all participants

Upcoming Events

- August 15, 2019: NIEM Health Symposium hosted by NIEM MO

Action Items

- Cait will distribute slides from presentation to the NIEM COI
- Cait will share necessary contact information with the NIEM COI
- Cait will email a meeting transcription to the NIEM COI

Meeting Minutes - April 9, 2019

- NIEM COI Meeting
- Agenda
- Slides
    - Lessons in Health IT learned from Hurricane Katrina
    - Bridging the Gap for Disaster Response
    - PULSE Design, Patient Target Populations
    - History of PULSE
    - PULSE Purpose
    - What Does PULSE DO?
    - Sample Workflow
    - PULSE Pilot Use Case - - Based on CA fires
    - Just in Time Training
    - PULSE Advisory Council consists of experts from state and federal government and healthcare
    - PULSE Connects to Nationwide Networks
- Post Presentation Discussion and Questions
- PULSE Action Items
- NIEM Action Item
- Closing comments

NIEM COI Meeting

Hosts: Cait Ryan and Brian Handspicker

Guest presenter: Mariann Yeager, CEO of The Sequoia Project

Topic: PULSE - Supporting a Nationwide Disaster Response System Slide Presentation

Length: 39:35 minutes

Venue: Virtual

Agenda

- Need for Health IT in Disaster Response
- Overview of Patient Unified Lookup System for Emergencies (PULSE)
- Lessons Learned from PULSE in California
- Leveraging Health IT for Disaster Response
- Nationwide PULSE Deployment
- PULSE Resources

Slides

Lessons in Health IT learned from Hurricane Katrina

- Concept for PULSE derived from devastation following hurricane Katrina
- Healthcare system at the time was largely a paper environment
- Natural and manmade events occur causing up to thousands of people needing to be treated
    - Prescription refills
    - Managing chronic conditions
    - Injuries resulting from the event
- Little access to health records for those seeking care outside of their normal healthcare delivery system
    - Those seeking or in a shelter
    - Records system is beyond capacity
- Needs to be a national strategy enabling health information to be accessible in areas where patients are displaced

Bridging the Gap for Disaster Response

- Needs versus gaps
    - Even with advancing HIT systems, several key needs are not being met
- Three critical needs:
    - Volunteer access to patient information when treating in shelters
    - Volunteers must be authenticated against state's volunteer database to access information
        - Databases exist and are used; however, there is no way to tie them to mass-emergency situations where a volunteer needs authentication before accessing EHRs
    - Leverage existing health IT infrastructure such as existing HIT networks rather than developing a dedicated disaster response network
        - Disaster response currently operates outside of the current infrastructure and the information sharing that occurs
- Gap:
    - Disaster healthcare volunteers do not have access to EHRs, health IT systems, or health information networks
    - EHRs, health IT systems do not authenticate users against state volunteer databases
    - Disaster response is not tied into health IT infrastructure

PULSE Design, Patient Target Populations

- This is a simple start, not the totality of what is needed throughout the country
- Disaster victims who need medical refills (such as a diabetic for whom an insulin prescription refill is an urgent need)
- All victims experiencing a medical trauma or emergency
- Displaced victims of disasters that occur outside of their normal health system geographic area of service
- Health information systems may be beyond capacity or, as in the case of the CA fires, destroyed

History of PULSE

- April 2014: ONC evaluates use of HIE infrastructure for disaster preparedness and response ONC published report exploring health information sharing during disaster response and EMS exchange with hospitals Funding provided by HHS Idea Lab to look at core technical approach and requirements as well as use cases to support
- March 2015: PULSE use case and technical architecture published
- July 2015: ONC awards EMSA a grant to advance HIE during a disaster and regionally in daily EMS
- March 2016: EMSA begins PULSE development
- July 2017: PULSE Go-line in CA
    - Value of PULSE recognized
    - Sequoia became steward for PULSE in collaboration with ONC, HHS, and other government organizations
- November 2017: PULSE activated for CA fires
- January 2018: PULSE migrates to private sector; Advisory Council formed

PULSE Purpose

- What is PULSE intended to do and who is intended to use it?
    - Assure health information is accessible by volunteer healthcare professionals who work in:
        - Mobile field hospital for Alternate Care Facility (ACF)
        - Medical shelter such as a tent or stadium, etc.
- How does PULSE work and what information is available?
    - Enables authorized volunteers to access health records to treat displaced individuals
    - Simply provides instant access which is not to be documented
    - It is a C-CDA-based exchange basic summary care record such as allergies, medications, immunizations
    - Principal use is critical to prescription refills

What Does PULSE DO?

- "PULSE enables authorized disaster healthcare volunteers to assess health records to treat people injured or displaced due to disasters."

- Provides Access to Health Information:
    - Summary record with problems lists, immunizations, allergies
    - Mediation history to aid prescription refills
- It was found beneficial to have patients go through the triage process before being treated, using triage as an opportunity to search for the patient's health information. Records are located and queued, and PULSE then provides "view-only" access.

Sample Workflow

- Slide shown depicts actual workflow used during CA fires triaging patients seeking treatment at a field hospital.

- Volunteer greets patient at triage
- Volunteer searches for patients using state ID, insurance card, or last known encounter
- Broadcast patient to all PULSE participants
- Finds patient match
- Searches for documents
- CCD summary of care documents retrieved and utilized by a disaster health volunteer
- Pulse is intended to be activated and available only under authority of the Emergency Management Services Authority and other authorities authorized to make those decisions in their jurisdiction
- Sample log in and information gathering slides show that PULSE is intended to be intuitive and simple, sensitive to various mental states of victims who are in a state of shock or despair.
- Only information that is a match based on basic information provided individual can be selected.
- Information is now queued for the attending physician and housed only for a period provided by the state.

PULSE Pilot Use Case - - Based on CA fires

- Emergency or disaster occurs
- Request made to EMSA to activate PULSE
- EMSA activates PULSE through designated PULSE operator

- Disaster Healthcare Volunteer (DHV) can log into PULSE from DHV system via single sign-on

- In California, six types of licensed volunteers can log into PULSE when activated by EMSA: (There is a planned expansion of the following list.)
    - Physician
    - Nurse Practitioner
    - Pharmacist
    - Physician's Assistant
    - EMT/Paramedic
- The location of where the emergency occurs determines to what extent they need help.
- During first activation of PULSE concerning wildfires statewide from October through November 2017, EMSA partnered with the eHealth Exchange network to increase PULSE's geographic coverage for accessible patient records.
- Risk factor for CA was initially considered to be earthquakes but the fires have put it to more significant use.
- It is better for national preparedness to have the system be in a secure state of readiness rather than trying to get it up and running during a disaster.
- Second activation of PULSE was in 2018 during the Shasta and Lake County fires.
    - EMSA partnered with eHealth Exchange to broaden access to additional providers for patients who were impacted.
    - Pulse was in a state of readiness to use (activated) and healthcare volunteers were deployed.
    - Even with limited use, additional lessons were learned to make the system more effective.
- Third activation was PULSE was November 2018 during Butte County fires.
    - EMSA realized use by medical personnel needed active deployment procedures
    - EMSA provided Just-in-Time-Training to a small group so they could use PULSE,

including Scripps Medical Response Team, California National Guard, and different medical and support groups
    - EMSA deployed a three-member employee team to initiate PULSE and train personnel and trained six additional staff members
    - This was most extensive use up to that time and was used in nine locations
    - Disaster Medical Services Division (CAL-MAT) team used PULSE for patient history and medical refills

Just in Time Training

- 132 Medical staff were provided "just in time training" on PULSE and deployed to Butte County
    - 20 Members of the Disaster Medical Services Division (CAL-MAT)
    - 10 Scripps Medical Response Team
    - 36 California Air National Guard
    - 50 California Army National Guard
- PULSE was used as of December 2018 to compile information on hundreds of patients.
- In disasters, every record represents a life.
- Two to three hundred records affected the lives of two to three hundred patients.
- HIT focuses on the value and perfection of the data, which is needed as a litmus test in day-to-day care delivery; however, in emergencies, any information is valuable, and often it is life-saving.
- Pulse should be a public utility for the good of the public facilitated through an open and transparent process.
- The advisory group's discussion of lessons learned and sharing of ideas ha led to the formation of an expansion strategy and set of resources.

PULSE Advisory Council consists of experts from state and federal government and healthcare

- Nora Belcher, Texas eHealth Alliance (TEHA)
- Rim Cothren, California Association of Health Information Exchange (CAHIE)
- Tara Cramer, Georgia Regional Academic Community Health Information Exchange (GRACHIE)
- Kristen Finne, HHS Assistant Secretary of Preparedness and Response (ASPR)
- George Gooch, Executive Directive, Texas Health Services Authority (THSA)
- Katharyn Kryda, Department of Transportation (DoT)
- Dan Smiley, California Emergency Medical Services Authority (CalEMSA)
- Lee Stevens, HHS Intergovernmental and External Affairs
- Sean Turner, Dignity Health
- Leslie Witten-Rood, California Emergency Medical Services Authority (CalEMSA)
- Jeremy Wong, Audacious Inquiry (AI)
- Mariann Yeager, The Sequoia Project (facilitator)

PULSE Connects to Nationwide Networks

- PULSE is already connected to eHealth Exchange Network, with California connections, and ability to connect nationwide

- Connectivity will be expanded through Carequality

- Vulnerable patient populations include those on dialysis, veterans, etc. eHealth Exchange and the expansion to include interoperability connections through Carequality provide the principle way for care plans to be accessed and connects multiple thousands of pharmacies, medical groups, dialysis centers, government agencies, hospitals, providers, and more.

- PULSE resources include information and answers about:
    - How would a jurisdiction know if PULSE is right for them?
    - What are the characteristics of a jurisdiction where PULSE would be successful?
    - There are enumerable funds available and CMA is willing to cover up to 90% of the PULSE implementation.
    - Identify the roles and responsibilities.
    - Is that jurisdiction data savvy?
    - Creation of Privacy Guidance Template
- Access resources at https://sequoiaproject.org/pulse/

Post Presentation Discussion and Questions

- PULSE is currently a web portal, there is no mobile application. A mobile application could be addressed as part of the future scope through additional public funding if there arises a need for it.
- The exchange that is occurring is through Query for Documents using IHE profiles and consolidated CDA, not NIEM.
- Daily use of PULSE by EMS and first responders is not projected; however, there is another project in the works addressing that as the EMS community uses a different format. Their format would have to be translated to the format used by PULSE.
- Mariann accepted an offer to be connected to the stewards of the Emergency Management Domain in NIEM to help with the translation from one format to another; that is what NIEM does.
- Concerning the authentication process of volunteers, the question of volunteer impersonation arose. Mariann mentioned the credentialing process but was not sure of what other provisions California would have pursued. Regarding the portal itself, if one has the credentials, they can access PULSE. All data/information on PULSE is only Cached for a limited period. For instance, in California, the cache was 20 minutes. In certain situations, it could potentially be shorter or longer. It does not capture clinical information.

PULSE Action Items

- HHS has keen interest in pushing PULSE to be implemented for the June 2019 hurricane season
- Support continued efforts in California and the Gulf states
- Explore potential for collaboration with NIEM as suggested by Cait Ryan

NIEM Action Item

- Cait Ryan will advise others of next meeting which will likely be in one month

Closing comments

- Mariann invites participants to contact her with additional information or questions
- Cait Ryan invites questions regarding the FHA transitioning
- Cait Ryan invites participants to contact her or Brian Handspicker for additional information or questions

Meeting Minutes - March 22, 2019

- NIEM COI Meeting
- Agenda
- Slides
    - NIEM Health Deliverables
    - Mapping to US Core Data for Interoperability
    - COI: Coming Up
    - Care to Share
    - Information Exchange Framework Applied to Structured Data Environments
    - NIEM in Canada
    - Employment and Social Development Canada (slide 1 of 2 title)
    - Employment and Social Development Canada (slide 2 of 2 title)
    - Achieving Interoperability
    - Use of NIEM in Interoperability Solutions
    - Managing NIEM Subset and Extensions
    - Enterprise Information Exchange Model
    - Strategy – Commoditized Data Assets
    - Enterprise-wide Implementation Challenges
    - Defining and Validating
    - NIEM Models in Partner Development Environments
    - XML to JSON
    - Partner Specific Policy/Restrictions
    - Information Exchange Framework Applied to Structured Data Environments
    - Why Integrate Information Sharing and Safeguarding?
    - Why Share Information?
    - Data Collection, Processing, and Analysis
    - Data Sharing and Safeguarding
    - Modern Information Management on a Side
    - What is OMG IEF?
    - Metadata, Tags, and Labels
    - Information Exchange Framework (IEF) Use Case
    - IEF Reference Architecture (IEF-RA)
    - Keys to a Solutions Success
    - Architecture Ecosystem for Information Sharing and Safeguarding (slide title “series”)
    - Full Traceability IER to Data Element
    - Information Exchange Framework (IEF)
- Closing Remarks
- ACTION ITEM

NIEM COI Meeting

Host: Cait Ryan

Presenters:

- Michael Savoie

- Don Clysdale
- Mike Abramson

Topic: NIEM COI

Length: 1:08:54 minutes

Venue: Virtual

Agenda

- Welcome, Upcoming Events, and Activities (Cait Ryan)
- NIEM in Canada (Veejay Mara, Michell (?), and Don Clysdale: Advanced Systems Management Group and KYM Advisors)
    - Opening Remarks
    - Presentation
- Information Exchange Framework Applied to Structured Data (Mike Abramson)
- Q&A (All Participants)
- Closing Remarks (Cait Ryan)

Slides

NIEM Health Deliverables

Cait Ryan

- Three tutorial/educational documents are in final their draft stage:
    - NIEM 101: An Introduction to Health Information Exchange
    - NIEM 102: An Introduction to Security and Privacy of Protected Healthcare Information
    - NIEM 201: Architecting NIEM IEPDs using Health Information Models
- Communications team will do a final review and format the three documents
- Scheduled for April 12, the documents will go to FHA leadership for final FHA review and edits
- Documents will be published to the NIEM Health COI space on Get Hub
- Documents will be sent to ENTAC teams for their final review

ACTION ITEM: Send documents to FHA Leadership team by April 12 for final FHA review and edits

Mapping to US Core Data for Interoperability

Cait Ryan

- Mapping will produce canonical and authoritative mapping that will align the NIEM community with standard fields, value sets, and codes used in clinical standards as required by USCDI
- A thorough review of this project was conducted at the March 14 FHA Federal Architects Council

- Those not present on March 14 are encouraged to contact Cait for reporting information and to learn about that exercise

COI: Coming Up

Cait Ryan

- The Patient Unified Lookup System for Emergencies (PULSE) – 2017 and 2018 California Wildfires Use Case
    - When: April 16 from 2:00 PM to 3:00 PM ET
    - What: The Patient Unified Lookup System for Emergencies (PULSE) is a nationwide health IT disaster response platform that can be deployed at the city, county, or state level to authenticate disaster healthcare volunteer providers
        - PULSE allows disaster workers to query and view patient documents from all connected healthcare organization
    - Who: The Sequoia Project

Care to Share

Cait Ryan

- Meeting participants are invited to express use cases they are interested in hearing about
- Meeting participants are invited to express if they would like to give a presentation to the group

Information Exchange Framework Applied to Structured Data Environments

Brian Handspicker (Introduction)

- This session continues discussions surrounding security and privacy for NIEM Health Information Exchange
- Just in time authorization to access and modify protected health information is an ongoing challenge for NIEM Information Exchange and a concern with the healthcare community and healthcare standards
- Regarding a highly-responsive security and privacy information exchange framework, Veejay Mara, who is Vice President and Principal Consultant from KYM Advisors, Co-Chair of OMG IEF Working Group, former Co-Chair of the Threat and Risk Information Sharing ONG Workgroup, and a driver behind NIEM UML Technical work is introduced
- Veejay will help the group understand security and privacy surrounding NIEM information exchanges
- Mike Abramson, President and CEO of Event Systems Management Group of Canada, Co-Chair of the OMG C4I Task Force and IEG Working Group is introduced

NIEM in Canada

Veejay Mara (Additional Introductions)

Michell (?), Don Clysdale , and Mike Abramson (Presenters)

- Veejay Mara introduces Michael Savoie and Don Clysdale, and reintroduces Mike Abramson
- (audio difficulty with Veejay Mara)

Employment and Social Development Canada (slide 1 of 2 title)

Michell (?)

- The mission of Employment and Social Development Canada (ESDC), including the Labor Program and Service Canada, is to build a stronger and more inclusive Canada, support Canadians by in helping them live productive and rewarding lives
    - ESDC fulfils its mission by:
        - Developing policies that ensure Canadians can use their talent, skills, and resources to participate in learning, work, and their community
        - Delivering programs to help Canadians through life transition, from school to work, job to job, unemployment to employment, and workforce to retirement
        - Providing income support to seniors, families with children, and those unemployed due to job loss, illness, or caregiving responsibilities
        - Helping Canadians with distinct needs such as Indigenous people, those with disabilities, the homeless, travelers, and recent immigrants
        - Ensuring labor relations stability by providing mediation services
        - Promoting fair and healthy workplaces by enforcing minimum working conditions, promoting decent work and employment equity, and fostering respect for international labor standards
        - Delivering programs and services
- When people require government services, ESDC is involved one way or another

Employment and Social Development Canada (slide 2 of 2 title)

Michell (?)

- Services include:
    - Find a job
    - Hire a temporary foreign worker
    - Public pensions including Canada Pension Plan, Old Age pension and related benefits
    - Disability benefits
    - Training
    - Funding programs through grants and contributions that support jobs, training, and social development
    - Workplace standards
- The list above comprises the main services ESDC is involved in
- Internal lingo was revised last year to include semantics and the information aspect of accompany the already existing emphasis on technology and application
- Still in use is the “same old way” of mapping which created a lot of complexities and diminishing the benefit of the technology being used
- This was alleviated by addressing the information aspect and the NIEM model fulfilled that gap

Achieving Interoperability

Don Clysdale

- Goal was to achieve interoperability across departments so the business would be more in charge
- One of the key tenets is to have a common information exchange model
- They started trying to achieve this goal on their own and then decided to go with NIEM

Use of NIEM in Interoperability Solutions

Don Clysdale

- ESDC interoperability team began using NIEM as their enterprise information exchange model in November 2017
- Interoperability solutions have been developed in the following areas:
    - Finance
    - Human Resources
    - Labor/Employment
    - Case Management
    - Document Management
    - Interactive Voice Response
        - People getting information over the phone through interactive voice responses
- Interoperability Partners:
    - (Other) Federal government departments
    - Provinces, Territories and
    - First Nations, Inuit and Metis communities
    - Employers
- This slide shows evidence of a big spectrum of partners from a variety of areas

Managing NIEM Subset and Extensions

Don Clysdale

- Enterprise Information Exchange Model
    - Subset of NIEM core/domains (want list)
    - Extensions to complete requirements
- New Requirements
    - Augment NIEM subset (larger want list)
    - Augment/refactor extensions
- It was understood from the beginning the work they were going to do had substantial overlap from solution to solution
- NIEM has a second layer noted as “Optional”; however, ESDC felt it was mandatory to leverage the work they were doing
- That work was to develop and Enterprise Information Exchange Model that would be common across all interoperability solutions
- Extensions were created extensions or use standards already in place according to the NIEM approach and have an ever-growing model as more solutions were made
- From a team of 60 to 70 developing these solutions, four members are dedicated to information models

Enterprise Information Exchange Model

Don Clysdale

- Slide displays where ESDC stands as of two days prior to this meeting
- ESDC subset now includes 47% of NIEM Core–47% in use on one of ESDCs applications
- Of the 47% in use, 97% is being used for infrastructure protection
- Human services, immigration, and justice hits on ESDCs core business of managing people entering Canada as well as managing "people" data
- Several extensions were made, including:
    - Address
    - Documents
    - Employment
    - Finance
    - Personal
    - Technical

Strategy – Commoditized Data Assets

Don Clysdale

- If ESDC has four providers, they want those providers to "speak" NIEM using their version and extensions
- Why? That is where the most value comes from
- If some providers cannot "speak" NIEM (such as those with a Legacy system), ESDC wants to promptly convert them to NIEM so other consumers, departments, obligation can "speak" NIEM with ESDC
- This is where most of the money is spent

Enterprise-wide Implementation Challenges

Don Clysdale

- Defining and validating constraints
    - Enterprise Model is more open that each business use
        - The Model cannot have everyone's constraints built into it
        - For instance, middle name is mandatory but not everyone has a middle name
            - When someone needs middle name to be mandatory, there needs to be a way of defining those constraints in a common way
- Implementing NIEM models in partner development environments
    - NIEM XML schema use constructs in partner development environments
        - Substitution groups (across schema)
        - Type extensions (from base type)
        - Enumerations
- XML to JSON
    - More than one approach
- Partner-specific policy/restrictions
    - Data subset (sharing agreement)
    - Data redaction (data sensitivity)

- The true first challenge is nobody wants to go with one model
- The challenge of moving NIEM model into other environments is they don't "speak our language"
- There are certain areas of NIEM that that the partners don not support, for which cases SAP is being used even though they don't support certain advanced techniques the way NIEM does, such as substitution groups, in their schema model
- Issues can occur when going back and forth from XML to JSON
- There are partner-specific policy restrictions with HR departments and ESDC wants to create a way for HR systems to share certain data with the interoperability platform, and then have the interoperability platform share only specific pieces of information that the HR people want to share with the partners
- Certain sensitive data may not be protected as well as it should be, ESDC wants to be to redact such data

Defining and Validating

Don Clysdale

- Level 0 validation (local constraint): Schema Validation
    - Validation of a single element/attribute
        - Consistency with expected IT structural requirements
- It was decided the best way to add business constraints was to use XML Schema assertion
- Level 1 validation (co-constraint): Business Validation
    - Validation of more than one element/attribute within the same dataset
        - Consistency with other data in the dataset (document/message)
- Implement a standard approach to put in their platform [inaudible] that had the same business validation capability which would be implemented in a standard way
- If both the schema and business validation were good, ESDC would process those messages to documents in the common way
- If that could not be done, because it was invalid, or the business rules were not respected, it would be handled in the standard air-handling way; ESDC has created a standard way of doing this

NIEM Models in Partner Development Environments

Don Clysdale

- Commercials tools have problems implementing complex XML Schema
    - Don't support specific XML Schema constructs
        - Substitution groups, extension/restriction from base type, choice…
    - Import tools don't support specific XML Schema constructs
        - Especially substitution groups across schema files
- Converting XML Schema to reusable object model
    - JAXB is available for Java objects
        - Able to generate enterprise-wide object model, which was successful
        - One enumeration too large
    - .Net tools exist, but are no longer supported, this was not as successful
- Schema by example
    - Using project message examples to produce simplified XML Schema

        - Resulting schema is specific to interfaces within project and subject to change if interface changes
- Since this slide was made ESDC started using pure JAVA script using JSON as the base and will convert back and forth
- Several big projects done with SAP with 60 different messaging operations
- Example messages are created for SAP which are used to create a simpler, NIEM-compliant schema that support what ESDC is doing with that partner
- The partner is given the NIEM-compliant, simpler schema which they use to create NIEM messages with ESDC
- The difficulty with that is if the schema changes, the information must be recreated, and the partner must implement that again

XML to JSON

Don Clysdale

- More than one approach
    - XSLT
    - XML to Java object to JSON
- Challenges converting to other models
    - Class/property versus type/element/attribute
- Need associative process so transformations, or their order, do not change the original message structure in any way
- If the message is not identical, businesses rules cannot be applied to the XML message that was there when started, which something is not going to work properly, such as business rule checking

Partner Specific Policy/Restrictions

Don Clysdale

- Data minimization (to match data sharing agreement)
    - Sharing agreement defines for each partner which operations and subset of the data they can consume
    - Standard authorization policies available to enforce operation access
    - Develop standard processes (policy enforcement) to enforce the data subset
        - Work in progress
- Data Redaction (to match data protection regime)
    - Need to extend authorization policies for data sensitivity, classification, privacy
    - Develop standard processes (policy enforcement) to enforce the data redaction
        - Work in progress
        - Extend data subset approach

Information Exchange Framework Applied to Structured Data Environments

Mike Abramson

- Solution to Policy-driven Data-centric Information Sharing and Safeguarding January 2019
- Focus is on selective sharing of information between partners

Why Integrate Information Sharing and Safeguarding?

Mike Abramson

- "At the heart of the intelligence effort lies a paradox."
    - Intelligence is valuable only if it can be shared with consumers who need it
        - More sharing equals an increased risk of compromise
    - Need to find best balance between adequate sharing and effective information security
- "Sharing and safeguarding are two side of the same coin."
- This paradox exists in every domain where sensitive (Private, Confidential, Legally-significant, or classified) information needs to be shared
- Sharing and safeguarding priorities are often seen as mutually exclusive; they are mutually reinforcing
- By implementing mechanisms to strengthen protections for sensitive information one helps to build trust within the user and stakeholder communities and increase their willingness to share
- Achieving an effective balance between Sharing and Safeguarding
    - Targets responsible information sharing
    - Represents a data management challenge more than a technology limitation
- Could not find a solution allowing to define an interface that had variances between the different people that information is being shared with
- After working with NIEM and other common messaging protocols, individual interfaces would need to be defined for individual partners even though the same exchange protocol was being used

Why Share Information?

Mike Abramson

- Inform Decisions
    - Shared situational awareness (hindsight, insight)
    - Shared intelligence (foresight)
- Enable collaboration and collective action
- Improve operational posture – higher quality information
    - Timely, accurate, current, actionable, complete, concise, accessible, relevant, consumable, understandable, reliable, etc., trusted (quality of information)
    - Get better information to people to get better decisions
- Resource multiplier
    - If able to make better decision based on better information, limited resources can better be applied to "the things we have to deal with"
- Foundation and enabler of: situational awareness, intelligence, collaboration, planning, command (coordination), cyber, etc.

Data Collection, Processing, and Analysis

Mike Abramson

- Data is collected from all available sources and:
    - Tagged, labeled, and catalogued to facilitate discovery, processing, sharing, and safeguarding

    - Transitioned into a form (institutional standards) that enables and facilitates processing and analysis
    - Staged for analytics, machine learning, and business intelligence services analytics inform intelligence, situational awareness, and planning
- The ability to gather all-source data and create quality information for decision makers is the primary role of IM/IT
    - Provide better information to front-line workers
    - Support collaborative planning between different organizations and agencies

Data Sharing and Safeguarding

Mike Abramson

- Once created, situational awareness, intelligence, and planning data is only useful if it can be shared
    - Data and information elements must be tagged and labeled to facilitate discovery, processing, sharing, and safeguarding
    - Data must be transformed into quality information (timely, accurate, current, actionable, complete, concise, accessible, relevance, consumable/understandable, reliable, etc., trusted
    - Information must be structured and formatted in accordance with individual information sharing agreements
- The ability to share information in a responsible and trusted manner is the cornerstone of a digital strategy
    - Responsible sharing
        - Maximizing the sharing and availability of information, while simultaneously protecting sensitive (private, confidential, legally-significant, and classified) information from unauthorized access, use, release, or manipulation
    - Quality information
        - Provision of information that is timely, complete, concise, actionable, relevant, consumable/understandable, reliable, trusted, etc.
- Must get information out to groups that have varying levels of trust

Modern Information Management on a Side

Mike Abramson

- Identified that many people want to start controlling information that gets into our environments in a better manner
- IEF is at both ends of the spectrum as shown on slide graph

What is OMG IEF?

Mike Abramson

- IEF is an integration pattern of security services that enable the selective sharing of information with partners based on privacy, security, different levels of trust, and different levels of need
    - IEF is simply a set of services
    - The reference architecture defines how these services come together

- Information Exchange Framework (IEF) is a collection of open standards developed under the Office Management Group (OMG), a global standards organization of status Beta 2 (completing finalization)
- Information exchange packing policy vocabulary (IEPPV):
    - The policy vocabulary is a Unified Modeling Language (UML) profile that specifies how to create a policy model based on the rules documented in a policy artifact, and associate these rules with datasets at a granular level
    - Rules can be associated with data element, meta data, tags for PII or security classification, or even data values depending on mission requirements
        - This approach allows for rapid creation/update of policies as policy instruments change, and retention of institutional knowledge in a model versus buried in code
    - UML based policy models are serialized for deployment into a runtime environment
        - Currently support XML or binary code (can be mapped to SAML/XACML assertion if needed
- IEF Reference Architecture (IEF-RA):
    - The IEF reference architecture specifies how policy models are interpreted and implemented in a runtime environment
    - Decisions for redaction/enrichment or sub-setting data are made based on the policies defined, and their association with attributes of the runtime environment (content, metadata, date tags, fabric, target audience, and their need-to-know, back-end attribute, or specific data values)
    - The outcome is a mechanism to disseminate different subsets of the same information in different formats to support partner conformance needs
    - The runtime environment is mission agnostic and can be used for one or multiple missions in a shared capability configuration
- Two additional initiatives underway with OMG
    - Information Exchange Packaging and Processing Service (IEPPS)
        - New specification for [packaging and processing messages
        - Data tagging: Standardized taxonomy for data tagging

Metadata, Tags, and Labels

Mike Abramson

- Structured data that describes, explains, locates, or otherwise makes it possible to retrieve, use, handle, protect or manage an information resource
    - Provide better information on the messages
- There are many types of structured data:
    - Descriptive (title, author, owner, abstract)
    - Structural (Schema)
    - Administrative (version number, creation date, archiving date, and other technical information for purposes of file management, rights management, and preservation
    - Provenance and pedigree
    - Security/confidentiality tags (which identify what information is allowed for each person)
    - Discovery
    - Handling instructions
- AS NIEM messages are prepared by machines in real-time, there is a need to provide the ability

to automate the tagging and labeling for the exchanges
    - No capacity to put an operator in front of every message being shipped over the network and to different partners
    - Needed to automate application of the rules about the data
- As many of the underlying data sources do not contain the tags and labels, the interface needs the ability to enforce tagging and labeling policy (rules) as the messages (products) are generated

Information Exchange Framework (IEF) Use Case

Mike Abramson

- Three basic concepts
    - Enforce information sharing policy
    - Enforce information safeguarding policy
        - Cannot separate sharing of information from safeguarding of information
    - Enforce data, packaging, and processing policy
        - Structure data being built as machine speeds in real time have a different context to things such as email files or text messages

IEF Reference Architecture (IEF-RA)

Mike Abramson

- http://www.omg.org/spec/IEF-RA/
- IE-RA: Defined elements (structure data)
    - Want to put EIF environments among existing configurations without any requirement so users of the EIF do not have to change the environment
- User provisional elements
    - Trustmark interface
    - Other security services
    - Identity management services
    - Privilege management services
    - Key management services

Keys to a Solutions Success

Mike Abramson

- Separate business, IM, and IT concerns
    - The business wants to make sure that the information is effectively used
    - IM wants solid architecture for the information that is storage sharing in that environment
    - IT wants to deploy a stable infrastructure that does not change every time the information or business changes
- Augment and not replace user applications and infrastructure
- Increase flexibility, adaptability, and agility during development and operations
    - Speed up the process for defining interfaces
    - Model-driven architecture/use of MBSE

        - Traceability: Business need to operations; retention of Institutional Memory; deduction in programing requirement
    - Rule-based applications/separate business rules from the code
    - Run load of business rules
    - Runtime administration of rules
        - Increased flexibility, adaptability, and agility
- Enhanced logging and auditing
    - Able to demonstrate responsible, trusted, and auditable; Real-time monitoring; Forensic auditing
- Integration of open standards

Architecture Ecosystem for Information Sharing and Safeguarding (slide title "series")

Mike Abramson

- IEF is an architecture ecosystem to manage and address long-term support for all interfaces being designed
- The UAF is the evolution of the Unified Profile for DODAF and MODAF
- The UAF is not another Framework; it is a common ontology, UML Profile and domain model for aligning Architecture Frameworks with Standard Modeling Languages
    - It was originally developed by DoD and Ministry of Defense in the UK to apply standard modeling techniques to their architecture frameworks
- IEF seeks to exploit architecture to develop and maintain ISS policy and solutions, and sustain interoperability
- Created Information Exchange Pacing and Policy Vocabulary which allows use of UML profile to define interfaces to aggregate, integrate, transform, and redact information as necessary to selectively share information with individual partners
- Use those models to generate an executable exchange policy
- Extending is being done using the UML for NIEM profile
- Using the UML to store storage schema mapping can be done from the storage models to the exchange models, and apply the policies which determine how the data is transacted
- With everything in the architecture, what can now be done is:
    - Start thinking about analytics
    - Look at threat risks of sharing information
    - Certify exchanges of information
    - Develop automatic statements of sensitivity of privacy profiles
    - Help strategic and architecture planning for how to expand the use of the data in a secure and safeguarded manner
- By having it done and generated in architecture, institutional knowledge is retained

Full Traceability IER to Data Element

Mike Abramson

- IEF seeks to exploit architecture to develop and maintain ISS policy and solutions, and sustain interoperability
    - STANAG 5525 example
        - A detailed, multi-diagram is shared which can be viewed on the Policy Models

for Structured Messaging PDF

Information Exchange Framework (IEF)

Mike Abramson

- Standards Specifications free to access and use:
    - http://www.omg.org/spec/IEPPV/
    - http://www.omg.org/spec/IEF-RA/
    - IEPPS RFP was issued December 2017 and is currently being developed
        - (on the slide, this is shown as a link)

Closing Remarks

Cait Ryan

- Cait will follow up regarding the next Community of Interest meeting on April 16 from 2:00 PM to 3:00 PM
- Cait invites recommendations on topics for CI meeting going forward
- Brian Handspicker thanks the presenters for raising important point and technologies for security and privacy that NIEM Health needs to start pushing forward

ACTION ITEM

- Send documents to FHA Leadership team by April 12 for final FHA review and edits

Meeting Minutes - January 29, 2019

- NIEM COI Meeting
- Agenda
- Slides
    - Lesson Learned
    - NIEM Face2Face
    - Course Correction: The Big Picture
    - Proposed Activities for FY'19
    - Existing NIEM Health-related Elements
    - NIEM Health Modeling - USCDI, FHIM, NIEM and Health IT Standards
    - Cross-Technology Modeling using Authoritative Sources
    - Upcoming COI Meetings
    - Care to Share?
    - The Issue of Protecting Health Information
    - NIEM Health 102
    - HIPAA: Examples of PHI
    - Health 102: Security and Privacy Timeline
    - Laws, Regulations, and Policies Mapping to Standards (slide 1)
    - Laws, Regulations, and Policies Mapping to Standards (slide 2)
    - Laws, Regulations, and Policies Mapping to Standards (slide 3)
    - LRP Mapping to Cyber Security and Healthcare Standards Spreadsheet Content (slide 1)
    - LRP Mapping to Cyber Security and Healthcare Standards Spreadsheet Content (slide 2)
    - Use for the Cybersecurity/Healthcare Standards Cross Reference Spreadsheet
    - LRPs Total Counts
    - ONC Security Frameworks Analysis Workgroup Timeline
    - Most Referenced
    - Health and Agency Specific
    - Status
    - COI: Next Up
- ACTION ITEMS

NIEM COI Meeting

Host: Cait Ryan

Presenters:

- Brian Handspicker
- Marco Demartin
- Regina Beach

Topic: NIEM COI

Length: 1:00:05 minutes

Venue: Virtual

Agenda

- Welcome (Cait Ryan)
- General Update (Cait Ryan and Brian Handspicker)
    - Activities since August 2018
    - Lessons Learned
    - Course Correction and Proposed Activities
- The Issue of Protecting Health Information (Brian Handspicker, Regina Beach, and Marco Demartin)
    - Recap of 102 Document
    - Review LRP Spreadsheet
- General Discussion / Q&A (All)
- Closing Remarks (Cait Ryan)

Slides

Lesson Learned

Cait Ryan

- There is no advantage to modeling a NIEM Health Domain from scratch; Map NIEM Health to HIE standards
    - After more than one year, this is something the NIEM team still feels strongly about
    - The focus remains on mapping to existing HIE standards
- NIEM information exchanges should generally not contain Protected Health Information (PHI) until HIPAA/HITECH privacy, consent, and authorization requirements are satisfied by NIEM
    - This was agreed upon during a look-back prior to the NIEM face-to-face meeting in November 2018
    - There is currently no mechanism in place for consent and authorization
        - Exception: “break-the-glass” scenarios like Public Health and use cases
        - FHA has been asked by the NIEM Business and Technical Archetype Council to participate in a future NIEM Privacy and Security Tiger Team around privacy and security for NIEM
- Registry is critical for all NIEM domains to track provenance
    - This is not just for NIEM Health as it could be applied for all domains
    - Some of the same concern were raised during the face-to-face meeting as well as during monthly participation in end-back calls
        - There is currently no way to identify who is using this information and for what types of purposes, which makes the mapping exercise difficult

NIEM Face2Face

Cait Ryan

- Attended NIEM Face2Face meeting in November 2018

- Action Items:
    - Work with the NIEM Technical Architects Committee (NTAC) on possible tiger team that will examine authorization, consent, privacy, and security for NIEM
        - This includes feedback and subject matter expertise
        - This includes information regarding if and when a tiger team is produced and NIEM starts participating in that
    - Provide input to NTAC on NIEM Registry*
        - This includes feedback and subject matter expertise
        - Explore what a NIEM registry looks and feels like
- NIEM Health Follow Up:
    - In motion - Talking with the Coast Guard regarding how they handle neurotoxin reports
        - Would like someone from the Coast Guard to present to the NIEM COI
    - ACF, regarding HIPAA and Privacy Concerns,
    - Justice, regarding Opioid Use Cases

Course Correction: The Big Picture

Cait Ryan

- Steady decline in NIEM Community of Interest (COI) Engagement:
    - From 62 to 11 participants between October 2017 and August 2018
        - Currently back up to 22 participants
    - Received two agency responses/feedback on the tutorials
        - All feedback from the NIEM COI is invited, positive or negative
        - Would like the NIEM COI to be more interactive
        - Cait suggests NIEM COI invite people from different domains as well as others who are outside of the Community to discuss scenarios participants have expressed to be interesting, for instance, regarding use cases
- Difficulty with Use Case Identification
    - Extensive number of Public Health use cases
    - Working to identify suitable use cases and stakeholder
        - Having difficulty identifying high-level, top-priority use cases
        - Multiple stakeholders make it difficult to determine which cases would best be leveraged
        - Those who have solved use cases are invited “to the table” to get the discussions going and foster relationships and work-coordination

Proposed Activities for FY’19

Cait Ryan

- FHA is proceeding in a deliberate, methodic manner to support the NIEM Community
    - Refocused NIEM Health 201 tutorial with and authoritative canonical set of health elements
        - After some changes/reworking, the 101, 102, and 201 tutorials will still be provided but for now the refocus is on 201
        - NIEM Health element inventory
        - Mapping of 2018 US Core Data for Interoperability (USCDI)
        - Map to Fast Healthcare Interoperability Resources (FHIR)

    - Reformat the COI to facilitate collaboration
        - Presentations by NIEM Health COI demonstrating the ways they've leveraged NIEM to solve use case
        - Presentations by HIT Community educating the NIEM Community of existing initiatives with overlapping interests
        - Cait invites participants to share about scenarios and areas of interest, and to be part of a presentation to this forum - the schedule can be made flexible to accommodate

Existing NIEM Health-related Elements

Brian Handspicker

- ~327 existing NIEM Health-related elements defined across several domains:
    - NIEM Core (~62)
    - Biometrics (2)
    - Emergency Management (10)
    - Human Services (~90)
    - Justice (~161)
    - Screening (2)
- Few defined based on HL7 Health IT standards, or did not reflect the entirety of HL7 standards for the element being defined in NIEM
- This made it challenging to map cleanly between NIEM IEPDs and HL7 messages

NIEM Health Modeling - USCDI, FHIM, NIEM and Health IT Standards

Brian Handspicker

- US Core Date Interoperability (USCDI) requirements:
    - USCDI structured into annual incremental requirement
        - Expanding annually from 2018, 2019, 202, 2021
    - ~600 Health standards requirements in USCDI '18
    - USCDI '19 aligns with Meaningful Use Common requirements (MU Common) and Continuity of Care docs
        - MU Common elements are supported by HL7 through the Consolidated Care Document Architecture (C-CDA) and Fast Healthcare Interoperability Resources (FHIR)
    - Most existing NIEM healthcare elements are not conformant with health IT standards
        - The challenge is to map existing NIEM Healthcare elements to the healthcare standards or, when that cannot be done, define the minimum subset of healthcare elements for NIEM so they align with healthcare standards
    - FHIM provides a common authoritative source of health standards information modeling - a model Rosetta Stone
        - FHIM is being used as a tool, as opposed to a solution, to help with the mapping

Cross-Technology Modeling using Authoritative Sources

Brian Handspicker

- Mapping is done by making use of NIEM modeling and methodology processes
    - Created a spreadsheet that maps requirements a scenario has for elements
    - Identify appropriate standards-based equivalent elements using FHIM as a tool for mapping between various HL7 standards
    - Identify NIEM elements that might map to the HL7 standards and if found, using the existing NIEM element. If not, a new NIEM element is defined and type that corresponds with the standards
- Determine, identify, list, and reference source element and source definition which must be consistent with one another
- Determine, identify, list, and reference corresponding FHIM Class Qualified names, property name, property documentation, and data type
- Identify, list, and reference corresponding NIEM, element definition, and element type
- Look up corresponding FHIR and HL7 value set bindings

Upcoming COI Meetings

Cait Ryan

- February 19, 2019
    - The Prescription Monitoring Information eXchange (PMIX), a NIEM-based Standard
        - Presenter - Jean Hall, PMIX
- March 1029 (TBD)
    - Canadian Public Safety; Privacy and Security
        - Presenter - Vijay Mehra, KYM Advisors
- April 2019 (TBD)
    - The Patient Unified Lookup System for Emergencies (PULSE)
        - Presenter - Marianne Yeager, The Sequoia Project
- Cait encourages participant to invite their colleagues

Care to Share?

Cait Ryan

- Participants are encouraged to share their success stories with the group
- Participants are encouraged to mention use cases they'd like to discuss and/or explore
- Participants are asked if there are Health IT initiatives they would like to invite as presenters
- Contact information is shared
    - Cait Ryan - cryan@irishealthsolutions.com
    - Brian Handspicker - bhandspicker@irishealthsolutions.com

The Issue of Protecting Health Information

Bryan Handspicker, Regina Beach, and Marco Demartin

- Three tutorial documents were developed over the last year
    - 101 - An introduction to Health Information Exchange (HIE) providing a NIEM COI-focused explanation of the health IT ecosystem, laws, regulations, and standards
    - 102 - An introduction to Security and Privacy of Protected Healthcare Information (PHI) discussing HIPAA and Privacy and Security of PHI

    - 201 - Architecting NIEM IEPDs using Health Information Models (HIM) providing guidance for identifying health terminologies, values, and code sets
        - Use is made of the Federal Health Information Model (FHIM) as a tool as part of the NIEM IEPD lifecycle

NIEM Health 102

Bryan Handspicker, Regina Beach, and Marco Demartin

- Privacy and security content were separated out from NIEM Health 101
- By the end of this tutorial, participants will understand the privacy and security concerns with respect to HIPAA and be able to identify:
    - What is PHI
    - Privacy rule
    - Security rule
    - Omnibus
- FISMA
- HITECH
- HIT/HIE Security and Privacy Standards
- NIEM Health Security and Privacy Challenges

HIPAA: Examples of PHI

Bryan Handspicker, Regina Beach, and Marco Demartin

- Much of the focus of HIPAA is on protecting identity information
- Healthcare information includes medical information as well as personal demographic information that someone could use to correlate to determine a condition a person may have

Health 102: Security and Privacy Timeline

Bryan Handspicker, Regina Beach, and Marco Demartin

- There are multiple laws, regulations, defined policies, and rules across Federal and State government that strive to protect individuals
- The challenge is determining which policies and laws apply for the security and privacy circumstance
- To help better understand the intricacies of protecting health information, Regina Beach (FHA) and Marco Demartin (FHA) produced a laws, regulations, and policies spreadsheet for security and privacy

Laws, Regulations, and Policies Mapping to Standards (slide 1)

Bryan Handspicker, Regina Beach, and Marco Demartin

- The current spreadsheet includes multiple tabs debuting the original 23 standards as identified by the FHA Standards Workgroup
- The standards were reviewed to determine what content should be collected or filter

- The spreadsheet includes acronym definition for any acronym used in the standards
- Also created was a cross-reference load sheet that developers can use

Laws, Regulations, and Policies Mapping to Standards (slide 2)

Bryan Handspicker, Regina Beach, and Marco Demartin

- The slide 2 spreadsheet contains 1,183 rows, 83 rows, and 45 columns with more than 7,000 cross-references and various select and filtering options
- The spreadsheet was compiled using public information and includes accompanying logging details as well as cross references between all documents
- Standards and their relation to different types of security as noted

Laws, Regulations, and Policies Mapping to Standards (slide 3)

Bryan Handspicker, Regina Beach, and Marco Demartin

- This slide represents cyber security listings

LRP Mapping to Cyber Security and Healthcare Standards Spreadsheet Content (slide 1)

Bryan Handspicker, Regina Beach, and Marco Demartin

- The spreadsheet is divided in to three areas
- Standards Columns contains all information pertaining to the classification
    - Publication
    - Title
    - Description
    - Date published
    - Date withdrawn
        - Date the standard was withdrawn (or retired)
    - URL
        - Select URL in real-time to view the publication and references
    - Future
    - Source
    - Type of Standard
    - Status of the Standard
    - Original List
- Standard Cross Reference
    - Authority
        - Pertains to Executive Order, a specific law, or US code
    - Applicable
    - Standard
- Filter Categories
    - CYBER
    - FEDRAMP
    - Cloud

    - HITECH
    - HIPAA Security and Privacy
        - Represents the current Security and Privacy Standard
    - HIPAA Security
        - Represents the original Security Standard
    - HIPAA Privacy
        - Represents the original Privacy Standard
    - Common rule
        - Includes organizations that are members of the Common Rule

LRP Mapping to Cyber Security and Healthcare Standards Spreadsheet Content (slide 2)

Bryan Handspicker, Regina Beach, and Marco Demartin

- Security filters categories taken from the NIST website regarding their classification of security standards for security purposes and what they support
- Key categories of security planning purpose are covered:
    - Access Control
    - Awareness & Training
    - Configuration Management
    - Contingency Planning
    - Identification & Authentication
    - Incidence Response
    - Maintenance
    - Media Protection
    - Personnel Security
    - Physical & Environmental Protection
    - Planning
    - Program Management
    - Risk Assessment & Authorization
    - System & Communication Protection
    - System & Information Integrity
    - System & Services Acquisition

Use for the Cybersecurity/Healthcare Standards Cross Reference Spreadsheet

Bryan Handspicker, Regina Beach, and Marco Demartin

- Answers questions such as:
    - What cybersecurity laws and standards are required to Implement Identification & Authentication?
    - What agencies follow the Common Rile for Protection of Human Research Subjects?
    - What standards do agency publications reference?
    - What standards address mobile technology?
    - Show timeline of laws, regulations, and policies (LRP), public and private industry standards
    - What is the status of cybersecurity frameworks?

LRPs Total Counts

Bryan Handspicker, Regina Beach, and Marco Demartin

- This spreadsheet includes Executive Orders, bills, US Code, Federal Register, and publications
- The slide shows items according to the classification of what they are, such as a bill, guided law, regulation so the user can generate different types of reports based on the classification of the standard

ONC Security Frameworks Analysis Workgroup Timeline

Bryan Handspicker, Regina Beach, and Marco Demartin

- An original timeline of the standards identified by the FHA Standards Workgroup
- A timeline or period can be selected as can publications within that/those timeframes

Most Referenced

Bryan Handspicker, Regina Beach, and Marco Demartin

- Regina provides a listing of standards that are reference the most
- SP 800-53r4 is the most referenced standard

Health and Agency Specific

Bryan Handspicker, Regina Beach, and Marco Demartin

- This spreadsheet reflects a listing of the Health and Agency Specific standards for cybersecurity
- Indicates which one's support cybersecurity classifications identified by NIST

Status

Bryan Handspicker, Regina Beach, and Marco Demartin

- Completed initial review of the original 23 publications, reference document, and first draft spreadsheet on September 17, 2018 with the following tabs:
    - Original List
    - Definition of column headers
    - Acronyms
    - LRP Details
    - Cross Reference (load file for MS access)
- Current work:
    - Cleanup of standard cross references
    - Compare content with ISA
    - Add other Technical Frameworks
    - Decide the best way to publish and maintain the tool
    - Identification of visualization requirements
- Marco Demartin suggests showing examples of the spreadsheet other than what was reflected

during the PowerPoint presentation
    - Show one example of the policy and how it I referenced in the different columns
    - Marco suggests showing one of the most referenced policies as the example
- Regina shows an example, going from pull-down to pull-down, of standard 800-53 (Recommended Security Controls for Federal Information Systems) as well as the current 800-53r4 and various other standards information notations and relationships

COI: Next Up

Cait Ryan

- February 19 at 2:00 PM ET
    - PMIX a NIEM Based Standard
        - Presenter - Jean Hall, PMIX
- The Prescription Monitoring Information Exchange (PMIX) National Architecture enables interoperability between systems PDMPs use for interstate exchange of PDMP data
- PMIX got it's start with NIEM and has since evolved into its own program
- Topics include:
    - Prescription Drug Monitoring Program
    - NIEM Exchanges
    - Privacy and Security of Health Information
- Participants are encouraged to invite their colleagues

Thanks for Coming Cait Ryan

- Contact information is shared
    - Cait Ryan - cryan@irishealthsolutions.com
    - Brian Handspicker - bhandspicker@irishealthsolutions.com
- Participant asks if the spreadsheet can be share to which Regina Beach affirms but advises that the spreadsheet is in draft form
- Regina shares her contact information, rbeach@irishealthsolutions.com
- Cait confirms she will reply to those invited to the meeting with Regina's contact information
- Marco reminds that the spreadsheet is a work in progress
    - Once the spreadsheet is officially released it will be available on the ONC website

ACTION ITEMS

1. Work with the NIEM Technical Architects Committee (NTAC) on possible tiger team that will examine authorization, consent, privacy, and security for NIEM
2. Provide input to NTAC on NIEM Registry*
    - Explore what a NIEM registry looks and feels like
3. Find someone from the Coast Guard to present to the NIEM community

Meeting Minutes - February 13, 2018

- Agenda
- Welcome and Introductions
- Health IT and the NIEM Health Recap
- Status Update
- Question and Answers/Timeline Review

FHA Attendees: Cait Ryan, Kevin Mundy

Other Attendees: *

Agenda

- Welcome
- Health IT and NIEM Health Recap
- Status Update
- Question and Answer

Welcome and Introductions

1. Cait welcomed the group and explained that all participants had joined in listen only mode. Participants were instructed to ask all questions in the questions window located on the GoToWebinar tool bar.
2. Cait introduced Kevin Mundy to the COI and explained that he has filled Alberto's position as the tech lead on the project.
3. Cait reviewed the meeting agenda and proceeded to the first topic.

Health IT and the NIEM Health Recap

1. Cait provided a recap from the October kick-off meeting where they discussed the complexities of Health IT and talked about the federal health information model, a robust information model developed by federal agency subject matter experts.
    - FHA met with several organizations from the NIEM community who have, in the past, approached FHA about their scenarios that they needed help with.
    - FHA found that just a small subset of health elements are needed, not all of healthcare.
    - FHA also heard a strong desire for a Community of Interest (COI).
    - Note: Meetings are currently being used to socialize the tutorials, once published we will determine how to best utilize these COI meetings. A thought at this time is to hold them as "office hours" and provide Q&A support to the NIEM COI.
    - It is important to mention that the use cases that we received from the community did not have many missing "health" elements, most were already found in NIEM
2. FHA determined that rather than "re-create" the wheel and model content into a traditional NIEM domain it will instead provide educational resources and instructions for locating health elements. These guides that will be published to the NIEM Health Activities page. These tutorials will:

    - Include a Data dictionary and Mapping Spreadsheet
    - Use the Use Cases we receive
    - Provide step by step guidance
    - Leveraging Federal Health Information Model (FHIM)
3. FHA will leverage the FHIM as this is a model developed by FHA and its agency partners and their subject matter experts. FHIM harmonizes various industry and agency standards to serve as the authoritative source for federal agencies.
4. Cait urged the group to join the February 22 FHA Learning Series that will provide an hour long "level-set" of the Federal Health Information Model.
5. Reviewed the proposed timeline and milestones and the work that has been done by the time team since meeting in October. Cait plans for the COI to meet more often and explained that we did not meet between October and not because of the NBAC Face to Face and the holidays.
6. Cait will send the draft tutorials later this month for the COI to review; it is important to provide feedback so that we know if it is something that is helpful and will be used by the NIEM Community.
7. The outline of the tutorial was reviewed
    - Update: This has since been broken into 2 separate documents so that the reader does not have to read the introduction to FHIM if they do not want to.

Status Update

1. Kevin Mundy began by reviewing the illustration of where these tutorials fit into that process to help facilitate the creation of an IEPD when health elements are needed.
2. FHA is not providing a typical domain, which will hopefully become obvious as to why we have chosen this approach throughout the remainder of this presentation.
3. Kevin went through the various Health Interoperability standards, their various uses and purposes, and showed how they often overlap or conflict.
    - The tutorials are not meant to serve as a deep dive of these standards but rather provide an overview and give the reader an idea of just how complex health IT is and why we will use FHIM.
4. Kevin touched on NIEM and how it provides a common vocabulary to generate consistent, reusable, data terms and repeatable process for consistent exchanges.
    - Standardized approach to the IEPD to create those exchanges.
5. FHIM is developing a well-structured model for the exchange of health information. For the NIEM Community to exchange health elements they need a starting point, as we saw there are different ways to use health standards in certain kinds of way. FHIM acts as a reference implementation model that sits above the standards we showed earlier.
6. Domains in NIEM represent various industries. Domains in FHIM are specific to healthcare; FHIM covers approx. 80% of the entirety of healthcare. FHIM domains are represented as UML object classes.
    - Kevin walked through the Dental Domain Model as an example.
7. In these tutorials we will describe the nuts and bolts of how to get the information needed and how to ensure you are using them properly. To begin we go to www.fhims.org website. There are 2 ways to access the data elements, via a spreadsheet or a view of the model and individually view the domains and their classes.
    - The spreadsheet will have 6 columns: class qualified name, class name, class documentation, property Name, Property Documentation and data type.
    - Note: A way to search the FHIM quickly is to use 3rd party tool such as

www.fhimview.com.
    - Can search and read the same documentation from the spreadsheet or for those who are more visual, you can view in a mindmap.
8. The 4 use cases that we have selected to use and test the process that will be illustrated in the tutorials. They are as follows:
    - Custody Release
    - Supervised Person Behavioral Health
    - Supervised Person Drug Test Services
    - MHS TRANSCOM
    - The first 3 use cases were recommendations from IJIS, they have a high business prioritization, and FHA has the most detailed information for these use cases. It is important to mention that after reviewing these use cases, there was very minimal health data actually required.
    - As we were working with these use cases, another case was brought to use by DOD which contained a much larger need for health information, MHS Transcom.
9. Kevin reviewed the MHS use case mapping spreadsheet that we used and showed that several elements already exist in NIEM. In the development of the Health Domain Guide, we will determine when to use FHIM as opposed to NIEM when coming across an instance where information is modeled in both.
10. We will then show you how to use the FHIM property documentation as a reference and instruct you where to look to find the codes and values for the property. In most cases you will use the National Library of Medicine's Value Set Authority Center (VSAC) which will often have all of this information listed in one spot, for others you may have to go straight to the source of the terminology (LOINC or SNOMED) or the exchange (HL7, FHIR, C-CDA etc.).

Question and Answers/Timeline Review

1. Cait answered a question about contact information and links to various documents. She explained that links to the NIEM Health activities space and emails are in the back of the slide deck and will also be sent in a follow up email to the group. Slides and recording will be sent in the follow up email as well and can be found on the NIEM Health Activities page.
2. Cait reviewed the next steps (slide 31).
3. Cait stressed that FHA is still accepting use cases and urged the community to send them in. We need these use cases to help get a greater variety of elements.
4. Steve H. asked when and how will FHA's FHIM and ONC US Core Data for Interoperability (USCDI) be harmonized; Cait did not have that answer but said that FHA is reviewing the USCDI and FHIM and looking to schedule a meeting with the right people. She also said this would be a good question to ask at the FHIM Learning Series meeting.

Meeting Minutes - October 10, 2017

- Welcome and Introductions
- The Federal Health Information Model (FHIM)
- Deep Dive into NIEM Alignment
- Questions
- Action Items

FHA Attendees: Cait Ryan, Alberto, Llanes, Galen Mulrooney, Steve Wagner

Other Attendees: *

Welcome and Introductions

2:00 PM - 2:10 PM

1. Cait welcomed the grouped, reviewed the agenda and explained that the attendees were on listen only mode until the Open Discussion.
2. Cait provided the following overview for the Federal Health Architecture program:
    - Established as an OMB e-gov LOB in 2005
    - FHA was created to ensure the seamless and secure exchange of health information among its partners.
    - FHA funding partners include Department of Defense-Defense Health Agency (DHA), HHS-OCIO, Social Security Administration (SSA), Veteran Affairs-Veteran Health Administration (VHA) and its managing partner The Office of the National Coordinator for Health IT.
    - FHA has lead initiatives in the following areas, CONNECT Gateway, Healthcare Directory, Direct, Enterprise Architecture, Federal Health Information Modeling and NIEM. (For a complete list of FHA lead initiatives see slide)
3. It was explained that given the nature of clinical health IT, and conversations with several NIEM community participants, the health domain is unique from others in that, we struggle to recognize the business need to use NIEM given the many exchange standards (HL7v2; HL7 FHIR) and terminology standards 9SNOMED, LOINC, RxNorm, etc.) that the clinical health community is already engaged in.
4. The FHA realizes there is a need from other domains to include health elements in their exchanges and they do not have access to those elements.
5. FHA Goals for NIEM:
    - Align the NIEM with clinical health standards by leveraging FHIM
    - Formalize the relationship between the NIEM and Clinical Health Communities
6. Alberto explained that there is a lot of overlap in clinical health standards, especially as terminology standards evolve which add to the complexity and making it near impossible for someone with no prior clinical health experience to know how to properly include health elements in their exchanges.
7. FHA will provide initial guidance leveraging its FHIM and referencing authoritative sources so that these elements can be used in IEPDs.

The Federal Health Information Model (FHIM)

2:10 PM - 2:40 PM

1. Galen explained the FHIM is a logical information model of health data developed in collaboration with the federal partner agencies designed to support the federal health enterprise.
    - Standardizes and harmonizes content (information, terminologies and value sets) across federal partner and standards organizations based on their priorities (e.g., MU) and use cases
    - Integrates with open source Model Driven Health Tools (MDHT) to automate the generation of health information exchange (HIE) standards (e.g., interoperability Specifications)
    - The FHIM captures the breadth and depth of health terminology and incorporates these standards: *HL7v2, HL7 CDA (including CCD & C-CDA), HL7 FHIR, NCPDP Telecom, NCPDP Script, ASC X12*
2. Galen walked the group through the Person Package
3. The group was invited to attend the following FHIM meetings(weekly)
    - Terminology (Wednesdays)
    - Information Modeling (Fridays)
    - Please email Galen Mulrooney for more information.

Deep Dive into NIEM Alignment

2:40 PM - 2:45 PM

1. Alberto outlined the steps FHA will take to publish preliminary guidance materials within the NIEM health community: (Please see slides for full details)
    - High Level FHIM to NIEM Core Mapping/Guidance
    - Identify most commonly needed health elements
    - Leverage FHIM to access authoritative element value sets and code sets
    - Provide high level and detailed guidance with examples on how to traverse from the NIEM Core to required FHIM elements and how elements can be associated at least conceptually if not through notation

Questions

2:45 PM - 2:55 PM

1. Due to audio issues this session was ended early. Questions were generic to the presentation asking for updated slides, links to FHIM and to the NIEM Health Activities page
    - www.fhims.org
    - http://niem.github.io/health/

Action Items

1. Update NIEM.github.io/health

NIEM Health Tutorials

The Federal Health Architecture (FHA), an Office of Management and Budget (OMB) e-Gov Line of business, has published a series of National Information Exchange Model (NIEM) guidance documents for the NIEM Health Community of Interest (NH-CoI) and NIEM stakeholders at large:

- NIEM Health 101: An Introduction to Health Information Exchange

  NIEM Health 101 provides a brief overview of the current health information technology (IT) landscape, related reference exchange and terminology standards, the NIEM Health challenges, and the Federal Health Information Model (FHIM).

- NIEM Health 102: An Introduction to Security and Privacy of Protected Healthcare Information

  NIEM Health 102 provides a brief overview of the security and privacy of protected healthcare information (PHI).

- NIEM Health 201: Architecting NIEM Health IEPDs Using Health Information Models

  NIEM Health 201 demonstrates how to use health IT standard information models to map NIEM Health use case requirements to NIEM elements and health information exchange standards elements.

- NIEM Health Elements Inventory

  The NIEM Health Elements Inventory is a tentative inventory of healthcare elements and types defined across the NIEM domains. The goal of this exercise is to determine what healthcare-related elements have already been defined across the NIEM Domains, compare those elements for completeness and standards compliance with ONC USCDI and health industry standards, and identify gaps that may need to be filled by NIEM Health definitions.

NIEM Health 101: An Introduction to Health Information Exchange

Produced by the Federal Health Architecture

Download this tutorial as a document or slides.

- Introduction
- NIEM Health Information Exchange Challenge
- Health IT Landscape and Standards
- HIT Data Exchange Standards
- Health IT Vocabulary Standards
- The National Information Exchange Model (NIEM)
- Federal Health Interoperability Modeling and Standards (FHIMS)
- Federal Health Interoperability Model (FHIM)
- NIEM and FHIM: A Brief Comparison
- NIEM Health Community-of-Interest Strategy
- Conclusion

Introduction

The Federal Health Architecture (FHA), an Office of Management and Budget (OMB) e-Gov Line of business, has published a series of National Information Exchange Model (NIEM) guidance documents for the NIEM Health Community of Interest (NH-CoI) and NIEM stakeholders at large:

- NIEM Health 101: An Introduction to Health Information Exchange,
- NIEM Health 102: An Introduction to Security and Privacy of Protected Healthcare

Information,
- NIEM Health 201: Architecting NIEM Health IEPDs using Health Information Models, and
- NIEM Health Element Inventory.

NIEM Health 101, the first of these NIEM guidance documents, provides a brief overview of the current health information technology (IT) landscape, related reference exchange and terminology standards, the NIEM Health challenges, and the Federal Health Information Model (FHIM). By the end of this guidance document you should better understand:

- NIEM Health information exchange challenge,
- Health IT landscape and health IT standards,
- National Information Exchange Model (NIEM),
- Federal Health Information Model (FHIM), and
- NIEM Health Community of Interest (CoI) Strategy.

This document should be used in conjunction with the follow-up NIEM Health 201: Architecting NIEM Health information exchange packet documentation (IEPDs) using Health Information Models guidance document and https://fhims.org.

The FHA program was created to coordinate health IT activities among more than 20 federal agencies that provide public healthcare services. FHA's mission was to support federal agencies and their public-sector exchange partners navigating the various health information systems standards and policies. By ensuring health information systems are interoperable, FHA enables organizations to send, receive, interpret, and operationalize patient information.

NIEM Health Information Exchange Challenge

Many NIEM-based communities (e.g., Justice, Homeland Security, Child and Family Services, etc.) need to be able to exchange domain-specific, albeit limited, healthcare information. Their challenge is to be able to exchange this limited healthcare information within their communities using NIEM-based technologies.

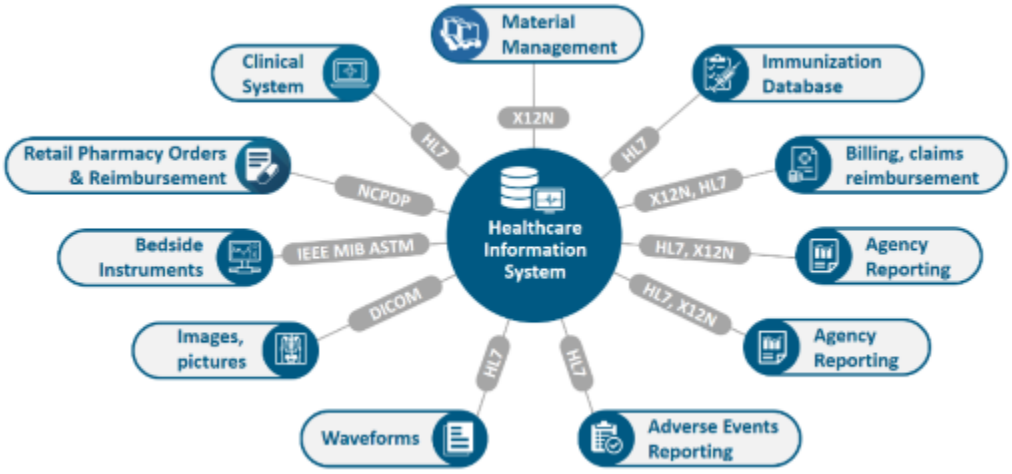
As these NIEM-based communities begin to experience a growing need to exchange health data with partner organizations, it is important that NIEM is compatible with existing health information standards (e.g., HL7, Consolidated- Clinical Document Architecture (C-CDA), Fast Healthcare Interoperability Resources (FHIR). However, it is apparent through conversations with the NIEM community that it does not require an information model in its entirety, but instead requires a small subset of general health

information. For instance, querying hospitals for available beds, dealing with emergency response, placing recovering drug users in half-way houses, etc. It is for this reason that we recommend identifying and mapping key health standard data elements that may be used instead of modeling a NIEM Health domain from scratch. This guidance is intended to help bridge the gap of health information readily available to NIEM and further the shared goal for national and regional interoperability based on a common vocabulary.

Health IT Landscape and Standards

The Healthcare Information Management Systems Society (HIMSS) summarizes today's health IT landscape as a sophisticated, heterogeneous environment comprised of a wide assortment of health care settings, stakeholders, and information systems, specifically for healthcare delivery. The sheer number and diversity of healthcare entities (e.g., providers, payers, researchers, and beneficiaries) presents a considerable information sharing challenge within the healthcare ecosystem. The challenge to interoperability [1] is primarily due to an absence of a widely adopted and integrated standardized vocabulary, or shared language. Despite the advent of several interoperability standards development organizations (SDO), the assortment of interoperability standards are not harmonized, resulting in gaps, overlaps, and duplication. Furthermore, competing standards create barriers to further development and progress of the standardized exchange and use of protected health information (PHI).

> For further information about health IT standards, please reference the white paper Evaluating Health IT Standards, published by Healthcare Information Management and Systems Society.



HIT Data Exchange Standards

HIT data exchange standards have evolved through the years to improve messaging and to drive high data quality. These structure and syntax standards include, but are not limited to:

- Digital Imaging and Communications in Medicine (DICOM): Imaging
- HL7v2.x: Messaging
- HL7v3: Clinical Document Architecture (CDA), C-CDA, Continuity of Care Document (CCD) documents
- HL7 FHIR: an emerging web services and document standard, currently in trial use

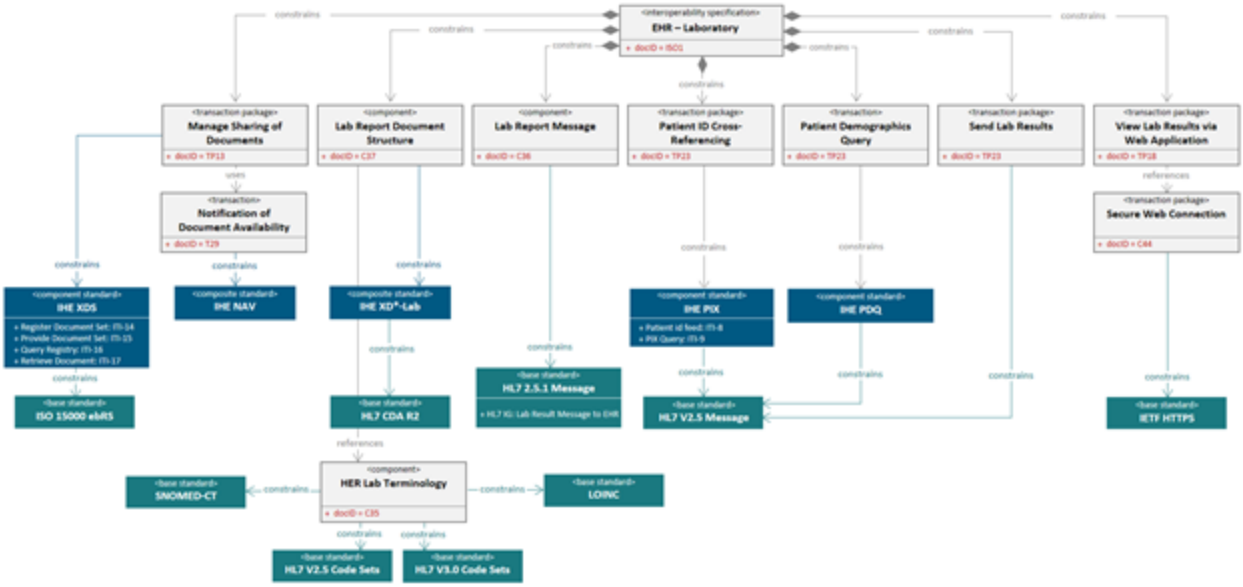
- National Council for Prescription Drug Programs (NCPDP): developed by the American National Standards Institute, an accredited standards development organization providing healthcare solutions, largely to the pharmaceutical industry

Health IT Vocabulary Standards

Health IT vocabulary standards were developed to ensure semantic and syntactical consistency with the characteristics and attributes for clinically specific patient medical record information terminologies. Examples of relevant vocabulary standards include:

- SNOMED CT (Systematized Nomenclature of Medicine-Clinical Terms): developed by the International Health Terminology Standards Development Organization (IHTSDO), it is comprehensive, multilingual clinical healthcare terminology originally intended to record observations for pathologists
- RxNorm: developed by the National Library of Medicine (NLM), it provides normalized names for clinical drugs and links its names to many of the drug vocabularies commonly used in pharmacy management and drug interaction software
- LOINC (Logical Observation Identifiers Names and Codes): developed by Regenstrief, LOINC serves as an international standard for identifying health measurements, observations, and documents, and was originally intended for reporting lab results
- CPT® (Current Procedural Terminology): developed by the American Medical Association (AMA) for the purpose of billing outpatient and office procedures
- ICD-10 (International Statistical Classification of Diseases and Related Health Problems 10th revision): a medical classification list developed by the World Health Organization (WHO), containing codes for diseases, signs and symptoms, abnormal findings, complaints, social circumstances, and external causes of injury or diseases. ICD-10 is commonly used for billing purposes
- DSM-5 (The Diagnostic and Statistical Manual of Mental Disorders): developed and administered by the American Psychiatric Association (APA), serves as a psychiatric classification and diagnostic reference tool

In addition to the standards mentioned above, health information is protected by a number of privacy and security laws that proscribe the type of terminology to utilize as well as the protections around the information being shared.



The combination of a history of multiple healthcare information exchange standards and multiple different specialized healthcare vocabulary standards results in a very sophisticated health IT environment that is challenging for non-healthcare domains to implement directly.

The National Information Exchange Model (NIEM)

NIEM is an XML information exchange standard, initially adopted by law enforcement and public safety information exchanges at the national and regional level. The systematic breakdown in communications and information sharing between emergency and first responders during the catastrophic events of 9/11 served as an inter-agency driver to develop NIEM.

NIEM specifies the foundation and building blocks for interoperable information exchange by serving as a common XML vocabulary, integrated with established information exchange standards and processes, to support cross-domain information sharing and efficient information exchange between inter-related public and private service domains (e.g., law enforcement, public safety, healthcare, etc.). As a result:

- NIEM breaks down interagency stovepipes
- NIEM enables agencies to share information that crosses system, agency, and jurisdictional borders
- NIEM improves decision-making, agility, and efficiency in satisfying business needs
- NIEM supports interoperability and reuse, reducing costs

Originally based on the Global Justice XML Data Model (GJXDM), NIEM has expanded and seen many iterative changes since its formal launch in April 2005 by the chief information officers of the U.S. Department of Homeland Security and the U.S. Department of Justice. As of June 2019, there are over 14 NIEM domains and emerging communities of interest (CoI), including Agriculture, Biometrics, Chemical Biological Radiological and Nuclear Defense (CBRN), Emergency Management, Human Services, Immigration, Infrastructure Protection, Intelligence, International Trade, Justice, Maritime, Military Operations, Screening, and Surface Transportation. Today all 50 states and over 19 federal agencies are using NIEM. The benefits of NIEM have also extended to Europe, the Americas, Australia, and Asia.

As NIEM's adoption continues to expand to new domains, non-clinical domains that utilize health data elements for information exchange will require clinical informatics subject matter expertise to successfully navigate through the complexities of the health IT/health information exchange (HIE) environment. Furthermore, operational health IT/HIE safeguards must be in place to ensure that the exchange of health information is always legal, secure, and private. The real-world challenge is to harmonize across all information exchange standards and to build a compelling business case, through the Federal Health Information Modeling and Standards (FHIMS) program and its corresponding FHIM that defines standardized terminology and the mapping of data relationships, both key to the efficient exchange of healthcare information.

Federal Health Interoperability Modeling and Standards (FHIMS)

The Federal Health Interoperability Modeling and Standards (FHIMS) program coordinates with partner agencies including:

- Department of Defense (DoD)
- Department of Veteran Affairs (VA)
- Department of Health and Human Services (HHS)
- The Office of the National Coordinator for Health IT (ONC)
- The Centers for Medicare & Medicaid Services (CMS)

These agencies collaborate in the development of standards for electronic medical/health records (EMR/EHR), health information and terminology, and the coordination of cross-agency efforts in relevant SDOs. Many partner agencies are active in SDOs; this enables the FHIMS program to unify agencies into a single voice while limiting redundant SDO participation. For those agencies that do not yet have a presence in an SDO, the FHIMS program provides a mechanism for agencies to delegate issues to another agency.

Federal Health Interoperability Model (FHIM)

Together these agency experts, including agency subject matter experts (SMEs), clinicians, and terminologists have developed and now maintain the Federal Health Information Model (FHIM). FHIM encompasses over 85% of the health domain and proscribes to the implementer exactly which terminology should be used for a given exchange, dictated by HIPAA, NCPDP, etc. This allows NIEM Health developers to identify and map key health standard data elements from the FHIM instead of modeling a NIEM Health domain from scratch.

FHIM is a logical health information model that supports semantic interoperability through development of a comprehensive and integrated set of implementation guides. The model harmonizes information and terminology content across federal partners and SDOs, and can standardize data concepts and serve as a repository of such concepts. As an implementation-neutral planning and reference tool, FHIM can be used to develop other health information models, including NIEM Health elements and NIEM Health IEPDs.

FHIM is modeled in standard Unified Modeling Language (UML) with both structural and terminology concepts, defining code systems, concept domains, and value sets. The terminology model describes the underlying terminology constructs, including URLs and version numbers. The FHIM logical model "binds to" (i.e. links to) the FHIM terminology model using the same UML profile. The combination of the two FHIM models result in rigorous specifications against which conformance can be tested. FHIM maintains traceability to underlying standards – especially HL7, NCPDP, and X12 – as well as to Standards & Interoperability (S&I) initiatives. FHIM leverages the HL7 Reference Information Model (RIM) as a reference model, but can also leverage other reference models. FHIM derivative models add "interoperability use cases," which provide additional constraints. These derivative models maintain traceability to the use cases and to the HL7 EHR-S functional model. The models can be leveraged by organizations for internal use in systems and database development.

There are multiple benefits of FHIM's model-driven approach:

- Analyze once, yet implement multiple times because clinical and business concepts remain the same regardless of HIE standards
- Ensure uniformity across HIE conceptual landscape for consistent HIE structures and clinical models
- FHIM can be used to create standards (e.g., HL7 implementation guides) because it is a product information management (PIM) system
- Generate code to minimize cost and human errors
- Generate other artifacts, including implementation guides, developer interactive assistance, etc.
- Enable seamless transition from one HIE to another HIE

FHIM continues to evolve as information modeling and terminology teams work closely together on health informatics concepts to promote the implementation of FHIM. To learn more about leveraging health standards for NIEM IEPD development, refer to the follow up tutorial to this work, NIEM 201: Architecting NIEM Health IEPDs using Health Information Models.

NIEM and FHIM: A Brief Comparison

NIEM and FHIM are logical information models. Combining the best features of FHIM and NIEM can result in NIEM Health IEPDs that are:

- Based on structures and constraints defined in NIEM
- Compatible with similar structures in consolidated CDA implementation guides required for meaningful use

FHIM's biggest value is that it is proscriptive, so that implementers know exactly which terminologies to use for a given element. Consequently, the resulting NIEM IEPDs comply with federal laws and are interoperable with existing standards-based health IT systems.

The table below provides a high-level view of the similarities and differences between NIEM and FHIM.

Comparison	NIEM	FHIM
Purpose	Provides a common vocabulary to enable efficient information exchange across diverse public and private organizations	Supports semantic interoperability and harmonizes information requirements from federal partners and multiple standards organizations
Model Type	Conceptual, Information Exchange, XML	Logical, Information, Platform-Neutral, Standard-Neutral, UML
Functionality	XML-based. "Proven" information exchange framework	Semantic-based. Information exchange, traceability, and alignment into industry information models and standards
Standards-Neutral	Yes, each domain steward determines the specific standards they will use	Yes, contents identified in the FHIM may be represented using any technology or standard

Comparison	NIEM	FHIM
		representation
Platform-Neutral	Yes	Yes
Domains	14, each is specific to a community of interest (CoI)	41, all are specific to healthcare
Exchanges	NIEM is about the semantics, not the transportation, of XML messages. Web services {e.g. Web Services Description Language (WSDL)} provide the mechanism for exchanging machine-to-machine NIEM XML messages	FHIM does not provide a method for transport of information
Developed By	Federal, State, and Local Agencies	Federal Agencies
Used By	Federal, State, and Local Agencies	Federal Agencies

Within health IT, the functional interoperability between EHR and HIE creates the bridge for inter-connecting the diverse modern health care ecosystem. NIEM can take business and information requirements to create reusable IEPDs to address the needs of heterogeneous health information producers and consumers. The IEPD is fundamental to the NIEM architecture, acting as a complete definition of an information exchange. It is composed of a data exchange schema, a business context, and usage documentation.

Like NIEM, FHIM provides a common vocabulary and a well-defined set of model elements to make the federal healthcare system more understandable and healthcare information more readily exchanged. It synthesizes recognized industry standards into a single model and is validated using a bottom up approach to its construction by agency representatives. The FHIM assists in identifying shared information needs across two or more agencies, building mutual understanding and consensus on data sharing and documents, and automating key aspects of information exchange protocols.

NIEM is comprised of industry specific domains, or CoIs, that are developed and maintained by a specific federal agency. For example, the Human Services domain is owned by HHS and the Military Operations (MilOps) domain is owned by DoD. Both domains are built from the NIEM core components and domain-specific elements.

Unlike NIEM, FHIM domains consist of a singular CoI: health. Therefore, FHIM domains are classified into an assortment of healthcare categories, as illustrated below, and can be viewed at http://fhims.org/.

AdverseEventReporting	CommonProduct	Lab	SecurityAndPrivacy
Allergies	Consultation	MedicationAdministration	SocialWork
Assessment	Dental	OncologyRegistry	SpinalCord
AudiologyAndSpecimen	Dietetics	Orders	Surgery
BehavioralHealth	Encounter	PatientEducation	VitalSigns
BloodBank	EnrollEligCOB	Person	WomensHealth
CarePlan	HealthConcern	Pharmacy	Common
ClincialDecisionSupport	HomeBasedPrim	Prosthetics	Datatypes
ClinicalDocument	Imaging	Provider	DetailedClinicalMod
ClinicalObservation	Immunization	PublicHealthReporting	

NIEM Health Community-of-Interest Strategy

The NIEM Health Community-of-Interest Strategy is to:

- Lead NIEM Health Community discussions to define use-cases, and functional and technical requirements
- Develop high-level FHIM to NIEM core mapping and guidance documents, including:
    - Identifying most commonly needed health elements
    - Leveraging FHIM to access authoritative element value sets and code sets
- Develop detailed guidance documents with examples on how to traverse from the NIEM core to required FHIM elements, and how elements can be associated at least conceptually if not through notation
- Review, revise, and socialize NIEM Health guidance documents with the NIEM Health community of interest
- Catalog and prioritize identified use cases for exchanging health related data outside the healthcare system
- Determine health IT standards/frameworks most relevant to high priority exchanges and seek alignment with the NIEM exchange development process
- Inventory existing NIEM health data elements
- Model and map required health IT standard data elements to existing or new NIEM Health elements
- Support a community of interest around the sharing of health data with non-traditional partners

Conclusion

Functional interoperability, both within and outside of HIEs, is the key to accurate, useful, and secure information exchange networks. Currently, disparate interoperability standards vary between health IT organizations and their stakeholders, and the lack of consistent standards adoption impedes HIE interoperability on both a national and regional scale.

FHIM can provide a means to harmonize competing standards and produce artifacts that will improve the quality of implementations through model-to-model transformations. Information exchanges have

numerous requirements related to standards and terminology. Their diverse data sets and cross-domain relevance require the FHIM logical model to describe the contents of the information exchanges (e.g., HL7, NIEM). With model driven health tools, the FHIM can bridge the existing semantic gap between domains (e.g., HL7, NIEM) to consistently exchange health information, regardless of context, by defining explicit information exchange requirements in a platform independent fashion.

Thus, FHIM and NIEM can be leveraged together to increase accuracy and to ease the development of functional interoperability between federal, local, state, and tribal health entities by adopting and integrating the standardized use and exchange of health data amongst various public health entities.

---

[1] The 21st Century Cures Act (Cures Act) defines interoperability in the context of health information technology (health IT) as health IT that

A. enables the secure exchange of electronic health information with, and use of electronic health information from, other health information technology without special effort on the part of the user;
B. allows for complete access, exchange, and use of all electronically accessible health information for authorized use under applicable State or Federal law; and
C. does not constitute information blocking as defined in section 3022(a).

NIEM Health 102: An Introduction to Security and Privacy of Protected Healthcare Information

Produced by the Federal Health Architecture

Download this tutorial as a document or slides.

- Preface
- Introduction
- Privacy Rule Requirements of HIPAA
- Exceptions to the HIPAA Privacy Rule
- Security Rule Requirements of HIPAA
- Evolution of Security and Privacy Protections
- The Federal Information Security Management Act (FISMA)
- Trust Frameworks
- eHealth Exchange's DURSA
- NIST's Cyber Security Framework
- ONC's Trusted Exchange Framework and Common Agreement
- Conclusion
- Appendix

Preface

The Federal Health Architecture (FHA), an Office of Management and Budget (OMB) e-gov line of business, has published a series of National Information Exchange Model (NIEM) guidance documents for the NIEM Health community of interest (NH-COI) and NIEM stakeholders at large:

- NIEM Health 101: An Introduction to Health Information Exchange,
- NIEM Health 102: An Introduction to Security and Privacy of Protected Healthcare Information,
- NIEM Health 201: Architecting NIEM Health IEPDs using Health Information Models, and
- NIEM Health Element Inventory.

NIEM Health 102, the second of these NIEM guidance documents, provides a brief overview of the security and privacy of protected healthcare information (PHI). By the end of this guidance document you should better understand:

- protected healthcare information,
- privacy requirements of HIPAA,
- exceptions to the HIPAA Privacy Rule,
- security consequences of the privacy requirements of HIPAA, and
- trusted healthcare information exchange frameworks.

Note: This document should be used in conjunction with the follow-up NIEM Health 201: Architecting NIEM Health Information Exchange Package Documentation (IEPDs) using Health Information Models guidance document and https://fhims.org.

Introduction

Privacy and security are both critical aspects of health information technology (IT). In 1996, Congress passed the Health Information Portability and Accountability Act (HIPAA), a federal law that provides data privacy and security provisions for safeguarding medical information. HIPAA was created in response to the need for healthcare reform, which aims to protect health insurance coverage, improve access to healthcare, reduce fraud and abuse, improve quality of healthcare, and decrease healthcare administrative costs (electronic transactions).

HIPAA places protections on most individually identifiable health information held or transmitted by a covered entity or its business associate in any form or medium, whether electronic, on paper, or orally. HIPAA has since been updated and additional policies and guidelines have been released surrounding patient privacy as well as the security of data (the type of information that is protected, how and where data is stored, the transport of information, etc.). The updates produced policies and procedures throughout federal, state, and local government agencies.

HIPAA resulted in established regulations to protect both the privacy and security of certain health information through two key rules:

1. HIPAA Privacy Rule: national standards for the protection of certain health information
2. HIPAA Security Rule: national set of security standards for protecting certain health information that is either held locally or transferred in electronic form

Privacy Rule Requirements of HIPAA

The HIPAA Privacy Rule defines protected health information (PHI). PHI includes demographic information, which relates to the individual's past, present, or future physical or mental health or condition, the provision of health care to the individual, or the past, present, or future payment for the

provision of health care to the individual, and that identifies the individual or for which there is a reasonable basis to believe can be used to identify the individual. Beyond health conditions, PHI includes many common identifiers when they can be associated with the individual's health information, such as:

- Name
- Birth Date
- Fax Number
- Account Number
- Web URL
- Street Address
- Admission Date Electronic Mail Address Certification/License Number
- License Plate Number
- City
- Discharge Date
- Social Security Number
- Vehicle Type and/or Serial Number
- Device Identifier and/or Serial Number
- Precinct
- Date of Death
- Medical Record Number
- IP Network Address
- Full Face Photographic Images
- Zip Code
- Telephone Number
- Health Plan Beneficiary Number
- Biometrics (e.g., Finger Prints, etc.)
- Any Other Identifying Information

The Privacy Rule assigns individuals with specific rights to their PHI. Furthermore, The Privacy Rule requires all healthcare organizations and partners to protect individuals' health records and other identifiable health information by establishing privacy protection safeguards as well as conditions and limits on the appropriate usage and disclosures of PHI. The Privacy Rule provides health information exchanges (HIEs) the foundation for balancing the privacy of individuals' PHI and assuring that their PHI is accessible to healthcare parties for providing care, payment, etc. The Privacy Rule warrants that everyone's health information is protected, especially during the ongoing flow of health information needed to provide quality health care and to protect public health.

The advent of digital healthcare systems {computerized physician order entry (CPOE), electronic health records (EHR), etc.}, has advanced healthcare delivery in so many ways: digital speed, interconnectivity, analytics, mobility, self-service, etc. The Security Rule protects individual's privacy while promoting the adoption, by diverse covered entity organizations (providers, health plans, clearinghouses, etc.), of leading and emerging digital healthcare technologies that drive high quality healthcare delivery through streamlined efficiencies, automation, new products, and services. The Security Rule enables the Privacy Rule's protections by establishing both technical and non-technical guidance and requirements which covered entities must both comply with and operationalize to secure electronic PHI (e-PHI). The Security Rule's flexibility and scalability empowers the diverse healthcare market to operationalize policies, procedures, and technologies relative to the entity's organizational size, structure, and security and privacy risks to their consumers' e-PHI.

In 2009, The Health Information Technology for Economic and Clinical Health (HITECH) Act was authorized through the American Recovery and Reinvestment Act (ARRA) to promote the adoption and meaningful use of health IT. Specifically, Subtitle D addresses the privacy and security of the electronic transmission of PHI and e-PHI, including strengthening legal enforcement of HIPAA. The following timeline illustrates the various security and privacy legislation and accompanying guidance that exist for the protections of PHI.

ARRA stipulates that the National Institute of Standards and Technology (NIST) [1] should:

- Advance health care information enterprise integration through standards and testing
- Establish grants program for health enterprise integration centers
- Consult on:
    - Updating the Federal Health IT Strategic Plan
    - Voluntary certification programs
    - Health IT implementation
- Provide pilot testing of standards and implementation specifications, as requested

NIST's security specifications enable communicating parties to ensure the confidentiality, integrity, and availability of health information and helps to safeguard health information by providing a harmonized set of security principles and guidelines for use in developing secure health information exchanges. Technical accomplishments include:

- Lead author of the Healthcare Information Technology Standards Panel (HITSP) secure communication construct
- Developed a resource guide for covered entities implementing the HIPAA Security Rule

Exceptions to the HIPAA Privacy Rule

The Department of Health and Human Services (HHS) Office of Civil Rights (OCR) released clarifying guidance on the exceptions to the HIPAA Privacy Rule that permit disclosure of PHI.

This guidance clarifies that physicians may disclose PHI to a patient's loved ones, regardless of whether they are recognized as relatives under applicable law, and that permissive disclosures are not limited by the sex or gender identity of the person. For example, a patient's unmarried same-sex partner is recognized as a relative with whom PHI can be shared.

In addition, this HIPAA and HITECH legislation permits disclosure of PHI to support public health activities including:

- Reporting communicable disease
- Conducting public health monitoring
- Public health investigations and interventions
- Supporting Food and Drug Administration monitoring
- Identifying patients exposed to a communicable disease
- Supporting medical monitoring of the workplace
- Supporting the use of certified electronic health record technology

The full list of exceptions to the HIPAA Privacy Rule includes:

- 45 C.F.R. § 164.502(a)(1)(iii) - Incidental to a permissible use or disclosure
- 45 C.F.R. § 164.506(c) - Treatment, payment, and healthcare operations purposes (e.g., quality improvement and credentialing)
- 45 C.F.R. § 164.510(a)(1) - Directory information (e.g., location, general health status) [requires opportunity for the patient to agree or object to the disclosure]
- 45 C.F.R. § 164.510(b)(1)(i) - To notify family/friends/patient-designated individuals involved in a patient's care [requires opportunity for the patient to agree or object to the disclosure]
- 45 C.F.R. § 164.510(b)(1)(ii) - To notify persons responsible for a patient's care or a patient's location and health status [requires opportunity for the patient to agree or object to the disclosure]
- 45 C.F.R. § 164.512(b)(1)(i) - To legally-designated authorities for public health activities
- 45 C.F.R. § 164.512(b)(1)(ii) - To report child abuse or neglect
- 45 C.F.R. § 164.512(b)(1)(iii) - With respect to an FDA-regulated product or activity for which that person has responsibility, for the purpose of activities related to the quality, safety or effectiveness of such FDA-regulated product or activity.
- 45 C.F.R. § 164.512(b)(1)(iv) - To an individual exposed to or at risk of contracting a communicable disease
- 45 C.F.R. § 164.512(b)(1)(v) - To an employer about an employee's work-related illness/injury or about findings from workplace medical surveillance when the patient is treated at the employer's request
- 45 C.F.R. § 164.512(c) - To report suspected abuse, neglect, or domestic violence in limited circumstances
- 45 C.F.R. § 164.512(d) - To a health oversight agency for health oversight purposes
- 45 C.F.R. § 164.512 (e) - In response to a judicial or administrative order, subpoena, etc.
- 45 C.F.R. § 164.512(f) - To law enforcement for limited purposes (e.g., reporting a crime on site)
- 45 C.F.R. § 164.512 (g)(h) - To coroners, medical examiners, funeral directors, and organ procurement agencies to enable their legally authorized duties
- 45 C.F.R. § 164.512(i) - For research purposes subject to restrictions
- 45 C.F.R. § 164.512(j) - To a person reasonably able to prevent a serious and imminent threat to the health or safety of a person or the public
- 45 C.F.R. § 164.512(k) - For specialized government functions (e.g., national security)
- 45 C.F.R. § 164.512(l) - To an organization responsible for providing worker's compensation benefits

Please reference The Nationwide Privacy and Security Framework for Electronic Exchange of Individually Identifiable Health Information (the Privacy and Security Framework), the HIPAA Omnibus Rule, and the Health Information Technology for Economic and Clinical Health Act (HITECH) for additional information regarding exceptions and exclusions for PHI and HIEs.

Security Rule Requirements of HIPAA

FHA's mission is to support the federal agencies and their public sector exchange partners and navigate the various standards and policies, like those mentioned above, so that health information systems are securely interoperable and patient information can be securely sent, received, interpreted, and operationalized. FHA leads its federal partners and other healthcare stakeholders in the development of innovative HIE architectures for the sharing of EHRs on a nationwide and regional basis. This huge effort requires careful planning, analysis, and coordination of an extensive number of agency business

requirements, healthcare delivery requirements, subject matter research, public policy, relevant technologies, and enterprise systems and devices for the sharing of healthcare information including data security, patient consent, clinical informatics, interoperability standards, and accurate healthcare provider and payer data.

Evolution of Security and Privacy Protections

Since HIPAA was first introduced, several laws, regulations, and polices have been developed to ensure that information exchange between healthcare participants (providers, HIEs, organizations, etc.) complies with HIPAA as well as new or updated standards and guidance. These include:

- HITECH (ARRA) – Health Information Technology for Economic and Clinical Health
- DoDI 8582.01 – Security of Unclassified DoD Information on Non-DoD Information Systems
- DoDI 8580.02 – Security of Individually Identifiable Health Information in DoD Health Care Programs
- VHA Directive 1605.01 – Privacy and Release of Information
- DHA AI 077 – Security Categorization (SC) and Control Selection for Information Technology (IT)
- S.2184 – Veterans Community Care and Access Act of 2017
- S.2193 – Caring for our Veterans Act of 2017
- HR 4242 – VA Care in the Community Act (Section 202)
- NPRM: CFR Title 42: Part 2 – Confidentiality of Alcohol and Drug Abuse Patient Records
- SAMSHA – Applying the Substance Abuse Confidentiality Regulation

In 2002, the federal government took matters of protecting information a step further when the Office of Management and Budget (OMB) introduced the Federal Information Security Management Act (FISMA) that applied to all federal information, including healthcare information.

The Federal Information Security Management Act (FISMA)

Updated in 2014, FISMA defines a comprehensive framework to protect government information, operations, and assets against natural or man-made threats. All agencies must be compliant with FISMA and report their compliance annually to the Office of Management and Budget. FISMA's primary framework is detailed in NIST 800-53, therefore all agencies must be complaint with NIST standards and guidelines to meet annual compliance guidelines FISMA.

FISMA assigns responsibilities to various agencies to ensure the security of data in the federal government. The act requires program officials, and the head of each agency, to conduct annual reviews of information security programs, with the intent of keeping risks at or below specified acceptable levels in a cost-effective, timely, and efficient manner. NIST outlines nine steps toward compliance with FISMA:

1. Categorize the information to be protected
2. Select minimum baseline controls
3. Refine controls using a risk assessment procedure
4. Document the controls in the system security plan
5. Implement security controls in appropriate information systems
6. Assess the effectiveness of the security controls once they have been implemented

7. Determine agency-level risk to the mission or business case
8. Authorize the information system for processing
9. Monitor the security controls on a continuous basis [2]

NIST's FISMA Implementation Project aims to produce several key security standards and guidelines required by Congressional legislation to comply with FISMA. These publications include FIPS 199, FIPS 200, and NIST Special Publications 800-53, 800-59, and 800-60. Additional security guidance documents are being developed in support of the project, including NIST Special Publications 800-37, 800-39, 800-171, 800-53A, and NIST Interagency Report 8011. It should be noted that the Computer Security Division continues to produce other security standards and guidelines in support of FISMA.

Some specific goals include:

- Implementing a risk management program
- Protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction
- Ensure the integrity, confidentiality, and availability of sensitive information

The American Recovery and Reinvestment Act (ARRA) stipulates that NIST should:

- Advance health care information enterprise integration through standards and testing
- Establish grants program for health enterprise integration centers
- Consult on:
    - Updating the Federal Health IT Strategic Plan
    - Voluntary certification programs
    - Health IT implementation
- Provide pilot testing of standards and implementation specifications, as requested

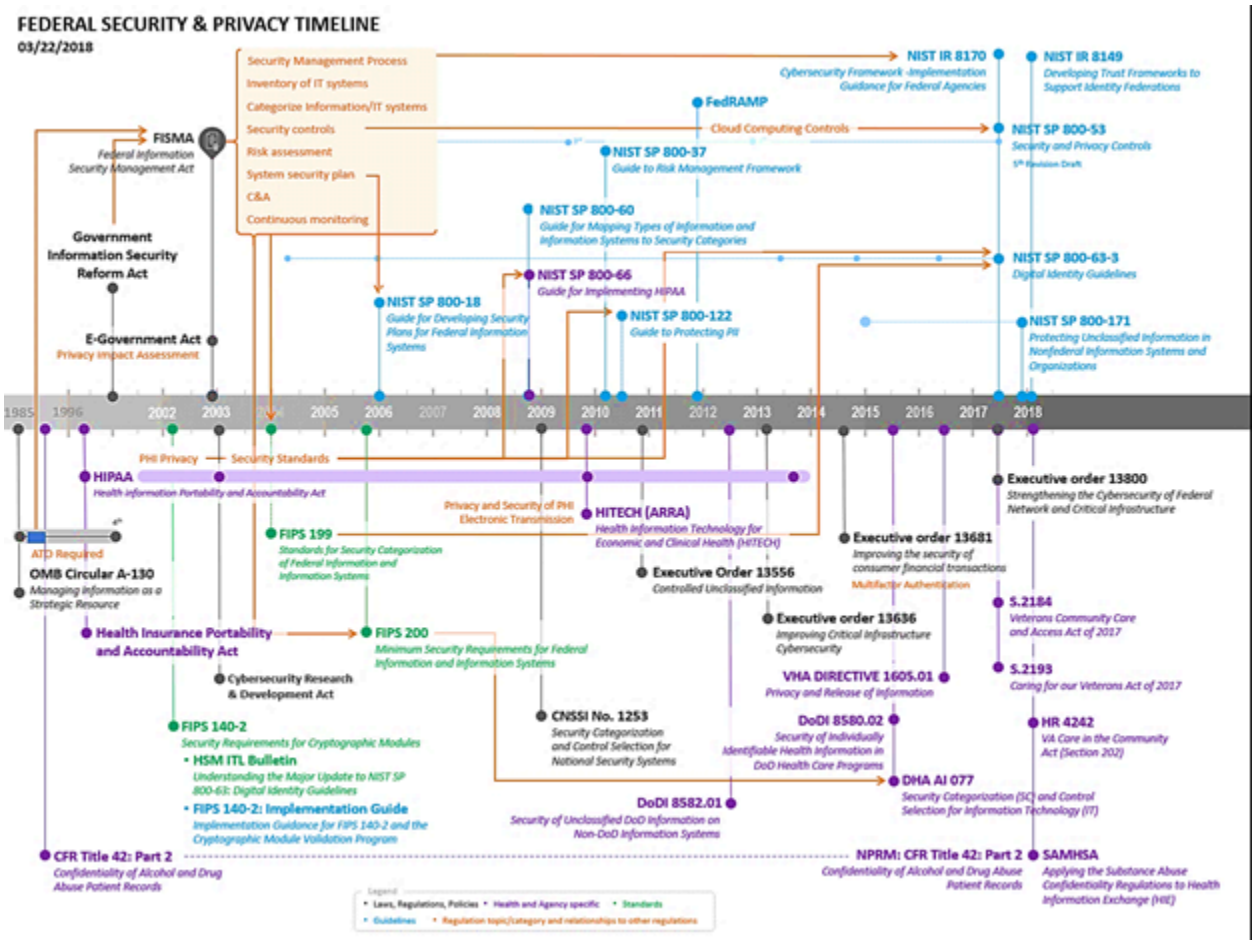
NIST's security specifications enable communicating parties to ensure the confidentiality, integrity, and availability of health information and help to safeguard health information by providing a harmonized set of security principles and guidelines for use in developing secure health information exchanges. Technical accomplishments include:

- Served as lead author of HITSP secure communication construct
- Developed a resource guide for covered entities implementing the HIPAA Security Rule

Though HIPAA and FISMA are governed by different federal agencies and serve different purposes, there is overlap between them. "When following FISMA requirements, HIPAA compliance can be achieved without any additional methodology. This philosophy does not hold true going in the opposite direction; HIPAA guidelines do not encompass all that is required for FISMA compliance. HIPAA provides guidance to address the provisions required for the security and privacy of health-related information, whereas FISMA presents instructions for the security of the information and the information systems which support these activities." [3]

Though the scope of this document is to provide a high-level overview of HIPAA requirements for protecting certain health information, it is important to point out the many laws that govern and direct privacy and security of protected health information. Below is a timeline FHA created that depicts the evolution of privacy and security governance for the federal agencies. A table with links to helpful references for additional reading is included. It is important to note that state and local governments

often have their own laws for security and privacy.



Trust Frameworks

There are several trust frameworks that healthcare organizations can leverage to help secure the confidentiality, integrity, and availability of information and data. A cyber security framework is a roadmap for securing IT systems with established policies and procedures in the form of agreements. These include eHealth Exchange's Data Use and Reciprocal Support Agreement (DURSA), NIST's Cyber Security Framework, and the Office of the National Coordinator for Health Information Technology (ONC's) Trusted Exchange Framework and Common Agreement (TEFCA).

eHealth Exchange's DURSA

The eHealth Exchange network started as the Nationwide Health Information Network (NHIN), an ONC program initiative. The network discontinued operating under the NHIN moniker and changed its name to eHealth Exchange when it moved to a private sector initiative, supported by The Sequoia Project. The eHealth Exchange network exists independently of Sequoia, the company. eHealth Exchange staff provide support to the eHealth Exchange network, the coordinating committee and participants, all under the direction of the eHealth Exchange coordinating committee. The coordinating committee has authority over the eHealth Exchange network, not Sequoia, and Sequoia is not a party to the DURSA. The DURSA is:

- A comprehensive, multi-party trust agreement that is signed by all eligible entities who wish to exchange data among eHealth Exchange participants
- A scalable alternative to multiple point-to-point agreements, which are not sustainable for widespread information exchange
- Requires signatories to abide by common set of terms and conditions that establish participants' obligations, responsibilities, and expectations
- The obligations, responsibilities, and expectations create a framework for safe and secure health information exchange. These are designed to promote trust among participants and protect the privacy, confidentiality, and security of the health data that is shared.

The DURSA was developed through an intensive effort facilitated by ONC, with consensus among a diverse group of private and state entities and federal agencies. As a living document, the agreement will be modified over time under the direction of the coordinating committee.

NIST's Cyber Security Framework

The Framework for Improving Critical Infrastructure Cybersecurity (known as the Cybersecurity Framework), is a voluntary framework. It is a risk-based approach to managing cybersecurity risk and is composed of three parts: the Framework Core, the Framework Implementation Tiers, and the Framework Profiles. Each Framework component reinforces the connection between business drivers and cybersecurity activities. It is comprised of standards, guidelines, and best practices to manage cybersecurity-related risk. The Cybersecurity Framework's prioritized, flexible, and cost-effective approach helps to promote the protection and resilience of critical infrastructure and other sectors important to the economy and national security (NIST, n.d.).

NIST developed a publication, The Cybersecurity Framework: Implementation Guidance for Federal Agencies, to assist federal agencies in strengthening their cybersecurity risk management. The publication helps agencies determine an appropriate implementation of the Framework for Improving Critical Infrastructure Cybersecurity. Federal agencies can use the Cybersecurity Framework to complement the existing suite of NIST security and privacy risk management standards, guidelines, and practices developed in response to FISMA, as amended. The relationship between the Cybersecurity Framework and the NIST Risk Management Framework are discussed in eight use cases.

ONC's Trusted Exchange Framework and Common Agreement

While the HITECH Act of 2009 stimulated significant health IT adoption and exchange of electronic health information with the goal of providing every American with access to their information, the interoperability needed to support this goal remains a work in progress. The 21st Century Cures Act's (Cures Act) 2 focus on trusted exchange is an important next step toward establishing an interoperable health system that:

- Empowers individuals to use their electronic health information to the fullest extent
- Enables providers and communities to deliver smarter, safer, and more efficient care
- Promotes innovation at all levels

In the Cures Act, Congress identified the importance of interoperability and laid out a path for establishing the interoperable exchange of electronic health information. In collaboration with NIST, federal agencies, industry stakeholders, and ONC are working diligently to further advance the

interoperability progress made to date and to address a complex-yet-core tenet of interoperability - building and maintaining trust. Among other provisions, in Section 4003, Congress directed ONC to "develop or support a trusted exchange framework, including a common agreement among health information networks nationally," which may include:

- "(I) a common method for authenticating trusted health information network participants;
- "(II) a common set of rules for trusted exchange;
- "(III) organizational and operational policies to enable the exchange of health information among networks, including minimum conditions for such exchange to occur; and
- "(IV) a process for filing and adjudicating noncompliance with the terms of the common agreement.6

The ONC's vision is to seek to achieve a system where individuals are at the center of their care and where providers have the ability to securely access and use health information from different sources; a system where an individual's health information is not limited to what is stored in EHRs, but also includes information from many different sources (including technologies that individuals use every day) and provides a longitudinal picture of their health. Additionally, this will enable a system where public health agencies and researchers can rapidly learn, develop, and deliver cutting edge treatments by having secure, appropriate access to electronic health information.

Conclusion

NIEM is currently supporting the secure and private exchange of information in production today. As NIEM's adoption continues to grow, non-clinical domains that utilize health data elements for information exchange will require clinical informatics subject matter expertise to securely and privately navigate through the complexities of the health IT/HIE.

Appendix

NIST SP 800-37

Guide for Applying the Risk Management Framework to Federal Information Systems: A Security Life Cycle Approach

The purpose of SP 800-37 Rev 1 is to provide guidelines for applying the Risk Management Framework to federal information systems to include conducting the activities of security categorization, security control selection and implementation, security control assessment, information system authorization, and security control monitoring.

https://csrc.nist.gov/publications/detail/sp/800-37/rev-1/final

NIST SP 800-63r3

Digital Identity Guidelines

These guidelines provide technical requirements for federal agencies implementing digital identity services and are not intended to constrain the development or use of standards outside of this purpose. The guidelines cover identity proofing and authentication of users (such as employees, contractors, or

private individuals) interacting with government IT systems over open networks. They define technical requirements in each of the areas of identity proofing: registration, authenticators, management processes, authentication protocols, federation, and related assertions. This publication supersedes NIST Special Publication 800-63-2. https://pages.nist.gov/800-63-3/sp800-63-3.html

- Section A - Enrollment and Identity Proofing Requirements
- Section B - Authentication and Lifecycle Management
- Section C – Federation and Assertions
- Section D – Identity Assertions

https://github.com/usnistgov/800-63-3/tree/volume-d (link to NIST document repository for Part D, Identity Assertion, yet to be released)

NIST SP 800-122

Guide to Protecting the Confidentiality of Personally Identifiable Information (PII)

The purpose of this document is to assist federal agencies in protecting the confidentiality of personally identifiable information (PII) in information systems. The document explains the importance of protecting the confidentiality of PII in the context of information security and explains its relationship to privacy using the the Fair Information Practices, which are the principles underlying most privacy laws and privacy best practices. PII should be protected from inappropriate access, use, and disclosure. This document provides practical, context-based guidance for identifying PII and determining what level of protection is appropriate for each instance of PII. The document also suggests safeguards that may offer appropriate levels of protection for PII and provides recommendations for developing response plans for incidents involving PII. Organizations are encouraged to tailor the recommendations to meet their specific requirements.

http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-122.pdf

NIST SP 800-171

Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations

The protection of Controlled Unclassified Information (CUI) while residing in nonfederal information systems and organizations is of paramount importance to federal agencies and can directly impact the ability of the federal government to successfully carry out its designated missions and business operations. This publication provides federal agencies with recommended requirements for protecting the confidentiality of CUI: (i) when the CUI is resident in nonfederal information systems and organizations; (ii) when the information systems where the CUI resides are not used or operated by contractors of federal agencies or other organizations on behalf of those agencies; and (iii) where there are no specific safeguarding requirements for protecting the confidentiality of CUI prescribed by the authorizing law, regulation, or governmentwide policy for the CUI category or subcategory listed in the CUI Registry. The requirements apply to all components of nonfederal information systems and organizations that process, store, or transmit CUI, or provide security protection for such components. The CUI requirements are intended for use by federal agencies in contractual vehicles or other agreements established between those agencies and nonfederal organizations.

http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-171r1.pdf

NIST SP 800-53r5

Security and Privacy Controls for Information Systems and Organizations

This publication provides a catalog of security and privacy controls for federal information systems and organizations to protect organizational operations and assets, individuals, other organizations, and the nation from a diverse set of threats including hostile attacks, natural disasters, structural failures, human errors, and privacy risks. The controls are flexible and customizable and implemented as part of an organization-wide process to manage risk. The controls address diverse requirements derived from mission and business needs, laws, executive orders, directives, regulations, policies, standards, and guidelines. The publication describes how to develop specialized sets of controls, or overlays, tailored for specific types of missions and business functions, technologies, environments of operation, and sector-specific applications. Finally, the consolidated catalog of controls addresses security and privacy from a functionality perspective (i.e., the strength of functions and mechanisms) and an assurance perspective (i.e., the measure of confidence in the security or privacy capability). Addressing both functionality and assurance ensures that information technology products and the information systems that rely on those products are sufficiently trustworthy.

https://csrc.nist.gov/csrc/media/publications/sp/800-53/rev-5/draft/documents/sp800-53r5-draft.pdf

Article on 800-53 and the Internet of Things

- NIST retools security and privacy controls for IoT era, https://fcw.com/articles/2017/08/16/nist-retools-for-iot.aspx

NIST IR 8149

Developing Trust Frameworks to Support Identity Federations, 16 January 2018.

When supported by trust frameworks, identity federations provide a secure method for leveraging shared identity credentials across communities of similarly-focused online service providers. This document explores the concepts around trust frameworks and identity federations and provides topics to consider in their development.

https://csrc.nist.gov/publications/detail/nistir/8149/draft

NIST IR 8170 (DRAFT)

The Cybersecurity Framework: Implementation Guidance for Federal Agencies (Comment period closed June 30th 2017)

This publication assists federal agencies in strengthening their cybersecurity risk management by helping them to determine an appropriate implementation of the Framework for Improving Critical Infrastructure Cybersecurity (known as the Cybersecurity Framework). Federal agencies can use the Cybersecurity Framework to complement the existing suite of NIST security and privacy risk management standards, guidelines, and practices developed in response to the FISMA, as amended. The relationship between the Cybersecurity Framework and the NIST Risk Management Framework are discussed in eight use cases.

https://csrc.nist.gov/csrc/media/publications/nistir/8170/draft/documents/nistir8170-draft.pdf

NIST SP 800-66 Rev. 1

An Introductory Resource Guide for Implementing the Health Insurance Portability and Accountability Act (HIPAA) Security Rule

This publication discusses security considerations and resources that may provide value when implementing the requirements of the HIPAA Security Rule, was written to help educate readers about information security terms used in the HIPAA Security Rule and to improve understanding of the meaning of the security standards set out in the Security Rule, direct readers to helpful information in other NIST publications on individual topics the HIPAA Security Rule addresses, and aid readers in understanding the security concepts discussed in the HIPAA Security Rule. This publication does not supplement, replace, or supersede the HIPAA Security Rule itself.

http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-66r1.pdf

Framework for Improving Critical Infrastructure Cybersecurity v 1.1

The Framework is a risk-based approach to managing cybersecurity risk and is composed of three parts: the Framework Core, the Framework Implementation Tiers, and the Framework Profiles. Each Framework component reinforces the connection between business drivers and cybersecurity activities. These components are explained below.

https://www.nist.gov/cyberframework/draft-version-11

- Office of Civil Rights Review of NIST Cyber Security Framework https://www.hhs.gov/sites/default/files/nist-csf-to-hipaa-security-rule-crosswalk-02-22-2016-final.pdf
- Related article, Cyber Security Framework Draft, https://www.govconwire.com/2017/05/nist-reveals-draft-of-cyber-security-framework/

FIPS 140-2

Security Requirements for Cryptographic Modules

FIPS specifies the security requirements that will be satisfied by a cryptographic module, providing four increasing, qualitative levels intended to cover a wide range of potential applications and environments. The areas covered, related to the secure design and implementation of a cryptographic module, include: specification; ports and interfaces; roles, services, and authentication; finite state model; physical security; operational environment; cryptographic key management; electromagnetic interference/electromagnetic compatibility (EMI/EMC); self-tests; design assurance; and mitigation of other attacks

- https://csrc.nist.gov/publications/detail/fips/140/2/final

FIPS 140-2: HSM ITL Bulletin

Understanding the Major Update to NIST SP 800-63: Digital Identity Guidelines

This bulletin outlines the updates NIST recently made in its four-volume Special Publication (SP)

800-63, Digital Identity Guidelines, which provide agencies with technical guidelines regarding the digital authentication of users to federal networked systems.

- https://www.nist.gov/publications/understanding-major-update-nist-sp-800-63-digital-identity-guidelines

FIPS 140-2: Implementation Guide

Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program

This Implementation Guidance document is issued and maintained by NIST and the Communications Security Establishment Canada (CSE), which serve as the validation authorities of the Cryptographic Module Validation Program (CMVP) for their respective governments. The CMVP validates the test results of National Voluntary Laboratory Accreditation Program (NVLAP) accredited Cryptographic and Security Testing (CST) Laboratories which test cryptographic modules for conformance to Federal Information Processing Standard Publication (FIPS) 140-2, Security Requirements for Cryptographic Modules. The Cryptographic Algorithm Validation Program (CAVP) addresses the testing of Approved Security Functions, Approved Random Number Generators and Approved Key Establishment Techniques which are referenced in the annexes of FIPS 140-2. This document is intended to provide programmatic guidance of the CMVP, and in particular, clarifications and guidance pertaining to the Derived Test Requirements for FIPS PUB 140-2 (DTR), which is used by CST Laboratories to test for a cryptographic module's conformance to FIPS 140-2. Guidance presented in this document is based on responses issued by NIST and CSE to questions posed by the CST Labs, vendors, and other interested parties. Information in this document is subject to change by NIST and CSE.

- https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/fips140-2/fips1402ig.pdf

FIPS 199

Standards for Security Categorization of Federal Information and Information Systems

The purpose of this document is to provide a standard for categorizing federal information and information systems according to an agency's level of concern for confidentiality, integrity, and availability and the potential impact on agency assets and operations should their information and information systems be compromised through unauthorized access, use, disclosure, disruption, modification, or destruction.

- https://csrc.nist.gov/publications/detail/fips/199/final

FIPS 200

Minimum Security Requirements for Federal Information and Information Systems

FIPS 200 is the second standard that was specified by FISMA. It is an integral part of the risk management framework that NIST developed to assist federal agencies in providing levels of information security based on levels of risk. FIPS 200 specifies minimum security requirements for federal information and information systems and a risk-based process for selecting the security controls necessary to satisfy the minimum requirements.

- https://csrc.nist.gov/publications/detail/fips/200/final

Executive Order 13681, Oct 2014

Improving the Security of Consumer Financial Transactions

https://obamawhitehouse.archives.gov/the-press-office/2014/10/17/executive-order-improving-security-consumer-financial-transactions

Executive Order 13636, Feb 2013,

Improving Critical Infrastructure Cybersecurity

https://obamawhitehouse.archives.gov/the-press-office/2013/02/12/executive-order-improving-critical-infrastructure-cybersecurity

On February 13, 2013, the President signed Executive Order 13636, "Improving Critical Infrastructure Cybersecurity," which gave NIST the responsibility to develop a framework to reduce cyber risks to critical infrastructure (the Cybersecurity Framework). As directed in the Executive Order, NIST, working with industry, will develop the Cybersecurity Framework and the Department of Homeland Security (DHS) will establish performance goals. DHS, in coordination with sector-specific agencies, will then support the adoption of the Cybersecurity Framework by owners and operators of critical infrastructure and other interested entities, through a voluntary program.

- NIST Testimony, Oversight of Executive Order 13636 and Development of the Cybersecurity Framework: https://www.nist.gov/speech-testimony/oversight-executive-order-13636-and-development-cybersecurity-framework
- DHS Incentive Study: https://www.dhs.gov/sites/default/files/publications/dhs-eo13636-summary-report-cybersecurity-incentives-study_0.pdf

Executive Order 13800, May 2017

Strengthening the Cyber-Security of federal Networks and Critical Infrastructure

On May 11, 2017, President Trump issued Executive Order 13800, **Strengthening the Cybersecurity of Federal Networks and Critical Infrastructure** (EO 13800 or EO), to improve the nation's cyber posture and capabilities in the face of intensifying cybersecurity threats to its digital and physical security. EO 13800 initiates action on four fronts:

1. It secures the federal networks that operate on behalf of the American people.
2. It encourages collaboration with industry to protect critical infrastructure that maintains the American way of life.
3. It strengthens the deterrence posture of the United States and builds international coalitions.
4. It places much needed focus on building a stronger cybersecurity workforce, which is critical for the nation's long-term ability to strengthen its cyber protections and capabilities.

The EO consists of three sections: Cybersecurity of Federal Networks, Cybersecurity of Critical Infrastructure, and Cybersecurity for the Nation. A working group of representatives from across the U.S. Government has been formed to implement EO work.

https://www.whitehouse.gov/the-press-office/2017/05/11/presidential-executive-order-strengthening-cybersecurity-federal

DHA AI 077

Security Categorization (SC) and Control Selection for Information Technology (IT)

This Defense Health Agency Administrative Instruction (DHA-AI), based on the authority of References (a) and (b), and in accordance with the guidance of References (c) through (h), establishes the DHA's procedures for the SC and Control Selection for DHA IT.

This AI applies to:

- All DHA personnel, to include: assigned or attached Service members, federal civilians, contractors (when required by the terms of the applicable contract), and other personnel assigned temporary, or permanent duties at DHA to include regional and field activities (remote locations)
- DHA-owned IT or DHA-controlled IT that receives, processes, stores, displays, or transmits Department of Defense (DoD) information

DHA AI 077 Security Categorization and Control Selection for IT

S.2184

Veterans Community Care and Access Act of 2017

https://www.congress.gov/bill/115th-congress/senate-bill/2184/text

S.2193

Caring for our Veterans Act of 2017

https://www.congress.gov/115/bills/s2193/BILLS-115s2193pcs.xml

HR 4242

VA Care in the Community Act (Section 202)

https://www.congress.gov/bill/115th-congress/house-bill/4242/text

42 CFR Part 2

Substance Abuse and Confidentiality

This regulation applies to all records relating to the identity, diagnosis, prognosis, or treatment of any patient in a substance abuse program that is conducted, regulated, or directly or indirectly assisted by any department or agency of the United States.

LINK

ISO/IEC 27001:2013

International Standard ISO/IEC 27001:2013

This specifies the requirements for establishing, implementing, maintaining and continually improving an information security management system within the context of the organization. It also includes requirements for the assessment and treatment of information security risks tailored to the needs of the organization. The requirements set out in ISO/IEC 27001:2013 are generic and are intended to be applicable to all organizations, regardless of type, size or nature.

LINK

---

[1] NIST is a non-regulatory agency whose missions is to promote U.S. innovation and industrial competitiveness by advancing measurement science, standards, and technology in ways that enhance economic security and improve our quality of life. https://www.nist.gov/about-nist/our-organization/mission-vision-values

[2] https://searchsecurity.techtarget.com/definition/Federal-Information-Security-Management-Act

[3] https://www.vmracks.com/resources/hipaa-compliant-hosting-insights/how-fisma-requirements-can-impact-hipaa-compliance/

NIEM Health 201: Architecting NIEM Health IEPDs Using Health Information Models

Produced by the Federal Health Architecture

Download this tutorial as a document or slides.

- Introduction
- NIEM Health Challenge
- NIEM IEPD Lifecycle
- Analyze Scenario Information Exchange Requirements
- Mapping Scenario to Healthcare and NIEM Information Models
- Use Case: USCDI 2018 Continuity of Care Scenario
- Create Mapping Spreadsheet with Scenario Element Requirements
- Map Scenario Element Requirements to FHIM
- Map Scenario Element Requirements to NIEM
- Generate NIEM IEPD Subset
- Map NIEM to Vocabulary Code Sets
- Conclusion

Introduction

The Federal Health Architecture (FHA), an Office of Management and Budget (OMB) e-Gov line of business, has published a series of National Information Exchange Model (NIEM) guidance documents for the NIEM Health Community of Interest (NH-COI) and NIEM stakeholders at large:

- NIEM Health 101: An Introduction to Health Information Exchange,

- NIEM Health 102: An Introduction to Security and Privacy of Protected Healthcare Information,
- NIEM Health 201: Architecting NIEM Health IEPDs using Health Information Models, and
- NIEM Health Element Inventory.

NIEM Health 201, the third of the NIEM guidance documents, demonstrates how to use health IT standard information models to map NIEM Health use case requirements to NIEM elements and health information exchange standards elements.

By the end of this tutorial you should:

- understand the NIEM Health challenge,
- understand how to traverse from health IT standard information models to NIEM elements,
- experience a step-by-step example of creating a NIEM Health IEPD messages, and
- be introduced to a cross-domain sample use case.

Prerequisite: You should be familiar with the NIEM Information Exchange Package Document (IEPD) lifecycle and the IEPD tutorials.

NOTE: The precursor to this tutorial are NIEM Health101: An Introduction to Health Information Exchange and NIEM Health 102: An Introduction to Security and Privacy of Protected Healthcare Information.

NIEM Health Challenge

Many NIEM-based communities (e.g. Justice, Homeland Security, Child and Family Services, etc.) need to be able to exchange domain-specific healthcare information. Their challenge is to be able to exchange this healthcare information within their communities using NIEM-based technologies.

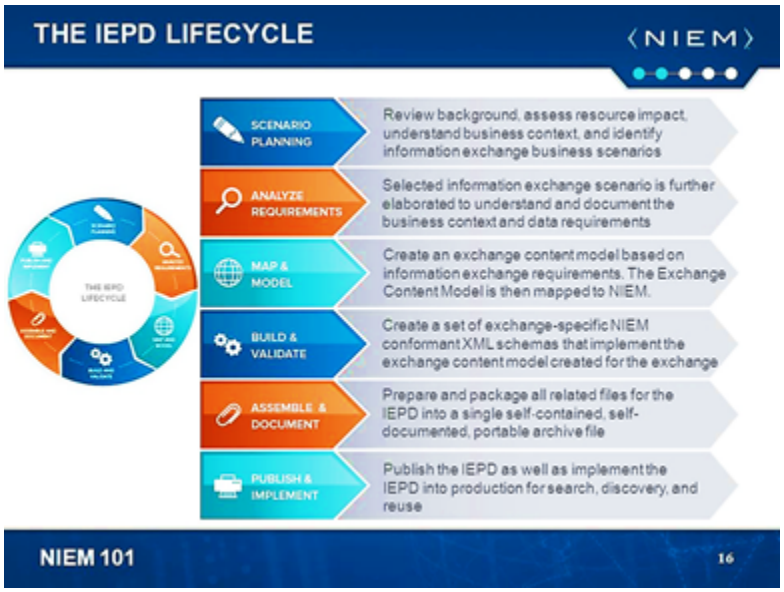
NIEM Health 101 establishes that the Federal Health Information Model (FHIM) is a useful authoritative source of health IT standard information model definitions for the NIEM Health domain. FHIM is a platform-independent logical health information model that supports semantic interoperability. FHIM is based on the HL7 reference information model (RIM). The FHIM health IT data elements have been harmonized across health information exchange standards, including HL7v2.5.1 (for meaningful use) and FHIR. It has been developed by the federal health IT community through collaboration between standards SMEs, federal agency SMEs, clinicians, and terminologists. Using FHIM as a tool to model information for the NIEM Health domain will guide the NIEM user

community, bridging the gap between the NIEM and clinical health communities.

NIEM IEPD Lifecycle

The NIEM Information Exchange Package Documentation (IEPD) lifecycle is a six-stage collaborative process involving multiple team players, including technical stakeholders (e.g., data architects, data modelers, developers), business analysts, and subject matter experts (SMEs). Since this is an interoperable exchange effort between two or more entities, it is critical for stakeholders on all sides of the exchange to participate in the planning, development, and execution of the exchange.

For the purposes of this tutorial, the focus is specifically on the second "Analyze Exchange Requirements" stage of the IEPD lifecycle. You should have your scenarios ready for the next step in this process. If you are unfamiliar with scenario planning, please visit the IEPD starter kit.



Analyze Scenario Information Exchange Requirements

The NIEM IEPD Analyze Exchange Requirements stage involves analysis of a scenario's data exchange requirements (e.g., content requirements, exchange partners, conditions that trigger an exchange, security requirements, etc.). The content portion of the requirements analysis stage may be captured in documents, spreadsheets, or UML diagrams. The kinds of information to be captured include:

- Data elements – These are the data fields for the message. For each field, the following information may or will be needed:
    - Element name – This is the name of the field / tag (e.g., first name, last name, street address, phone number, SSN, arrest date, unit name, and badge number)
    - Data type – The structural representation or format of the element. (e.g., string, date, number, Boolean, state code set, person data type, location data type)
    - Definition – Descriptive definitions will be required for all components in the exchange. At this stage, the semantics need to be captured, but the wording does not have to be finalized or perfected. If in a later stage the element is determined to map to NIEM, then the NIEM element will come with its own definition. The definition

captured here should be clear in meaning
    - Occurrence constraints – The minimum and maximum number of times an element may appear in the instance (e.g., last name may occur once only, SSN may occur 0 to 1 times, phone number may occur 0 to many times)
    - Default occurrence constraints in NIEM are 0 to unbounded (with a few exceptions); the default in XML schema is once only. Without explicitly setting these occurrence constraints, the defaults mean that components reused from NIEM will be optional and the ones added locally will be required
    - Source information – (Optional) It may be useful or necessary in some cases to record the source of the requirement. While traceability information may seem very apparent at the time, they can be difficult to reconstruct later if needed
- Objects/Classes – These are the complex, reusable data structures where related elements are grouped together. Examples include person (with elements first name, last name, SSN, and phone) and location (with elements street address, city, state, and ZIP code).
- Values/Code Sets – This is a list of allowable values, such as a state code list or an eye color list. Code sets may come from a standard or may be custom-defined.
- Conditions/Business Rules – There may be certain restrictions on the content that need to be represented. Examples include restrictions on individual values (such as a number that cannot be less than zero or a string that must have a certain number of characters) and conditional restraints (such as if the SSN is not provided, then last name and birth date are required), etc.

NOTE: Other requirements of the exchange, such as technical, security and privacy, performance, reporting, etc. should be described in this stage as well.

Mapping Scenario to Healthcare and NIEM Information Models

Mapping from scenario requirements to information models specific to healthcare information exchange standards and NIEM information exchange IEPDs is a multi-step process.

Scenario Element	Scenario Element Definition	FHIM Class Qualified Name	FHIM Property Name	FHIM Property Documentation	FHIM Data Type	NIEM Element	NIEM Element Definition	NIEM Element Type	Value-set Source	Value Set-Binding
Treatment	Medication Assisted Treatment Mental Health Treatment	FHIM::CarePlan::CareIntervention	Kind	"Detailed description of the type of planned activity. E.g. What lab test, what procedure, what kind of encounter." - HL7 FHIR, CarePlan.activity.detail.code	Code	ihx:CaseInformationDocument/nc:Treatment/nc:TreatmentText	A treatment that was provided or could be provisioned.	nc:TextType	SNOMEDCT	Care Plan Activity

Steps: A through D

A. Determine/Identify/List/Reference scenario required element and definition which must be consistent with one another
B. Determine/Identify/List/Reference corresponding health IT standard elements specified as FHIM class qualified name, property name, property documentation, and data type
    - Compare FHIM property name with scenario element name for consistency
    - Compare FHIM property documentation value with scenario definition value for consistency
C. Identify/List/Reference corresponding NIEM element, element definition, and element type
    - Generate a set of potential NIEM elements from FHIM elements
    - Compare generated NIEM elements with potential pre-existing NIEM core and domain elements to determine if generated elements are duplicative or necessary
    - Select NIEM element to be used, either from FHIM-generated set or pre-existing NIEM set
    - Compare chosen NIEM element with scenario element for consistency
    - Compare NIEM element definition with scenario definition for consistency
    - Compare NIEM element type with FHIM data type for consistency
D. Lookup corresponding value set bindings
    - Identify/list corresponding Vocabulary Value Set Binding
        - List value set binding name
        - List value set binding source(s)

- List value set binding notes: concepts/codes/source

Use Case: USCDI 2018 Continuity of Care Scenario

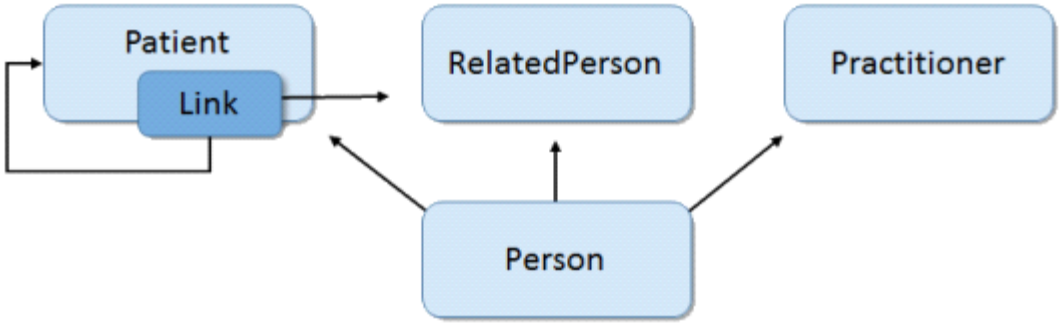
For this tutorial we will focus on mapping just the first six USCDI 2018 demographic element requirements instead of mapping all 600+ FHIM elements implied by USCDI 2018. For the complete mapping of USCDI 2018, see the forthcoming NIEM Health Inventory and Model specification.

The second stage of meaningful use requires that healthcare providers use C-CDA document exchange regularly in care transitions, and the continuity of care document (CCD) has been identified as the most appropriate document for this purpose. These documents must be capable of including data elements known as the "Common MU Data Set" that include: patient name, sex, date of birth, race, ethnicity, preferred language, smoking status, problems, medications, medication allergies, laboratory tests, laboratory values/results, vital signs, care plan fields including goals and instructions, procedures, and care team members. In addition, encounter diagnoses, immunizations, referral reason, and discharge instructions may be required based on context.

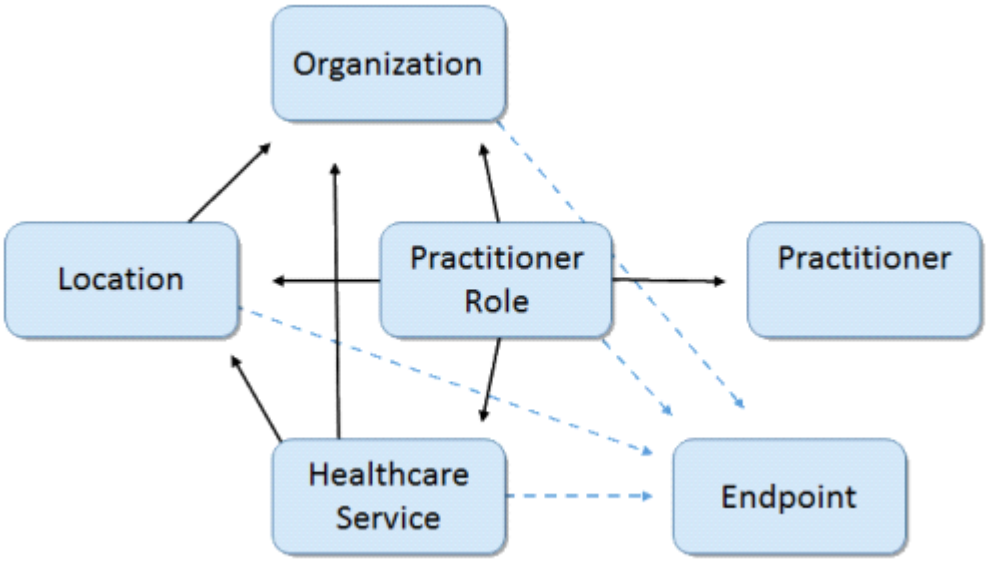
U.S. Core Data for Interoperability (USCDI) requirements for 2018 align with the Common MU Data Set, defining the required classes/elements to be:

1. Patient Name
2. Sex (Birth Sex)
3. Date of Birth
4. Preferred Language
5. Race
6. Ethnicity
7. Smoking Status
8. Laboratory Tests
9. Laboratory Values/Results
10. Vital Signs
11. Problems
12. Medications
13. Medication Allergies
14. Health Concerns
15. Care Team Members
16. Assessment and Plan of Treatment
17. Immunizations
18. Procedures
19. Unique Device Identifier(s) for a Patient's Implantable Device(s)
20. Goals
21. Provenance
22. Clinical Notes

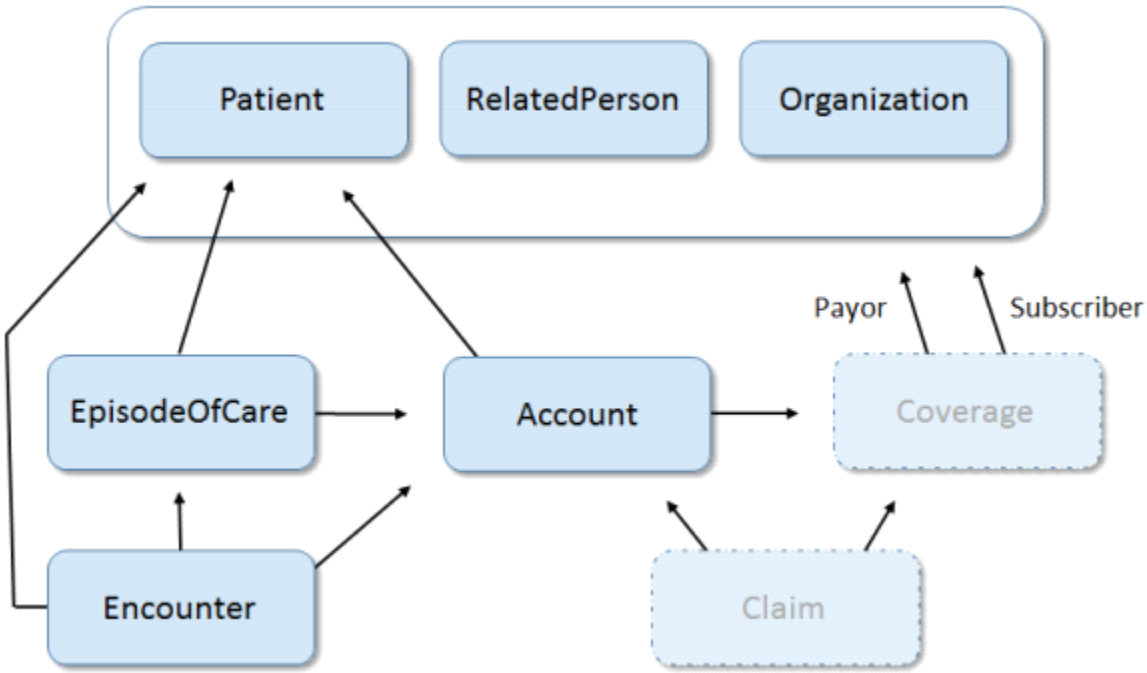
Of these requirements, this tutorial will use the first six "demographic" elements (name, sex, birthdate, language, race, and ethnicity) for our modeling and mapping examples.



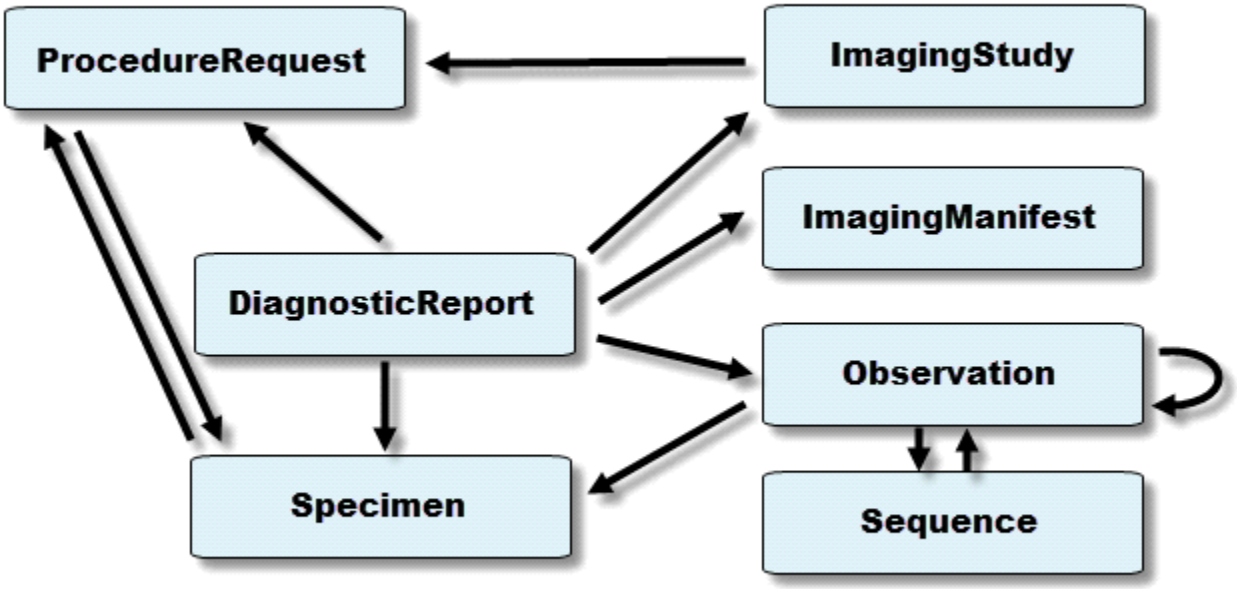
Person Data Model Derived from USCDI 2018 Scenario Requirements



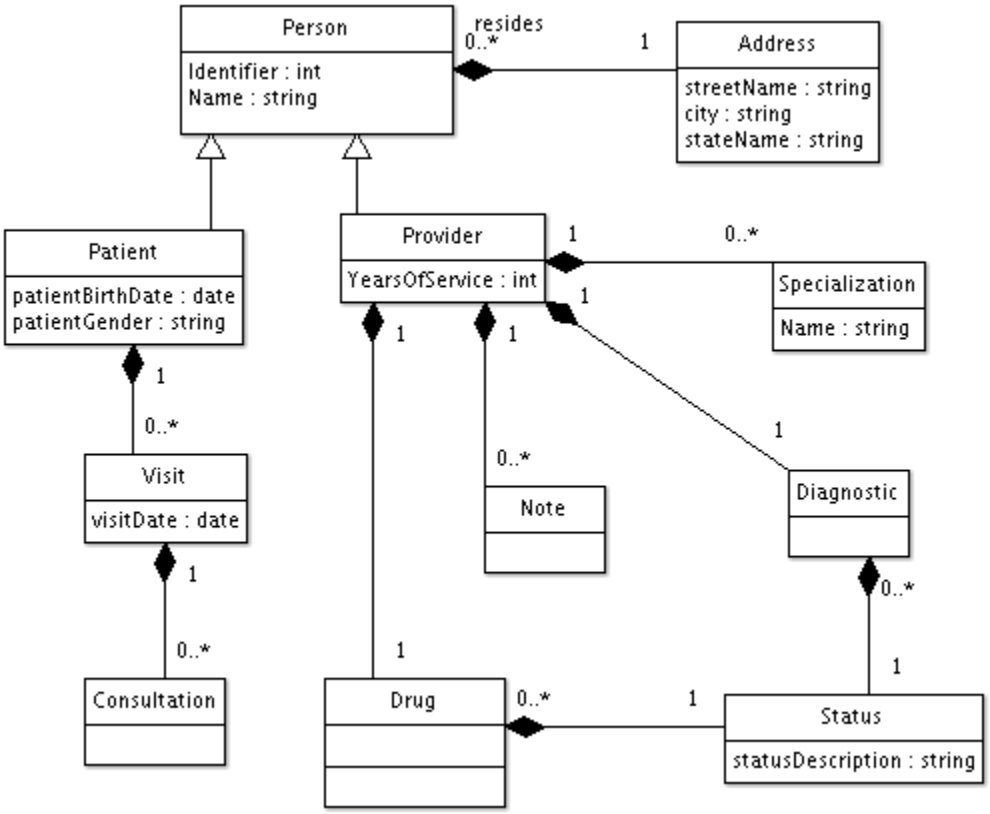
Location/Healthcare Service/Provider Model Derived from USCDI 2018 Scenario Requirements



Encounter/Episode of Care Data Model derived from USCDI 2018 Scenario Requirements



Diagnostics & Procedures report Data Model Derived from Scenario Requirements



Create Mapping Spreadsheet with Scenario Element

Requirements

The first step is to create a spreadsheet that documents the requirements derived from the scenario model. In subsequent steps, we will map these requirements to canonical healthcare standards resources, NIEM elements, and FHIR elements:

Scenario Required Object Class	Scenario Required Element	Scenario Required Definition	...	...	...	....
The name of the data object class you need for exchange	The name of the data element you need for exchange	A clear and cogent definition of your data element				

Next populate the scenario required object class and elements with their names and definitions:

Requirements Class/Object	Requirements Element	Requirements Definition	...	...	...
Patient					
Patient		A person receiving medical care.	...		...
Patient	First Name	First name of the patient	...	...	...
Patient	Last Name	Last name of the patient	...	...	...
Patient	patientBirthDate	Patient date of birth	...	...	...
Patient	patientGender	Patient gender	...	...	...

Scenario requirements in mapping spreadsheet

Map Scenario Element Requirements to FHIM

The next step is to map the scenario element requirements derived from the scenario model to conical healthcare standards resources using FHIM. First add and label columns to the mapping spreadsheet for FHIM values to aid in mapping the source elements to the FHIM semantic equivalents:

Scenario Required Object Class	Scenario Required Element	Scenario Required Definition	FHIM Qualified Class Name	FHIM Property Name	FHIM Data Type	...	....	...

Scenario Required Object Class	The name of the data element you need for exchange
Scenario Required Element	The name of the data element you need for exchange
Scenario Required Definition	A clear and cogent definition of your data element
FHIM Qualified Class Name	The fully qualified class name from FHIM
FHIM Property Name	The name of the property from FHIM that most closely matches your source elem
FHIM Data Type	The data type of the FHIM property

Proceed to fill in values for each FHIM column by locating the scenario requirements elements online in FHIMView (http://fhimview.com) or download the FHIM spreadsheet (https://fhims.org/docs/FHIM_DataElements_AllClasses.xlsx):

N.B. There are two ways to view the FHIM:

Option 1: View the UML model of each healthcare domain at http://fhimview.com

Option 2: Download the spreadsheet of all data elements from http://fhims.org

Option 1: Upon clicking on the link, you will navigate to the list of FHIM domains. Double click domain to explore the UML model for that domain, or click on **+ Diagrams** or **+ Models** at the to see the drop-down items. FHIM data elements are expressed in 6 columns, as indicated bel

Class Qualified Name	Class Name	Class Documentation	Property Name	Property Documentation
Fully qualified name of the class	Name of the class	Description of the class	Property or element within a class	Description of the property

Requirements Class/Object	Requirements Element	Requirements Definition	FHIM Class/Object	FHIM Class Qualified Name	FHIM Property Name	FHIM Property Documentation
Patient		A person receiving medical care.				
Patient	First Name	First name of the patient		FHIM::Datatypes::PersonName	Given	"A set of names given to a person at birth, but not including the family name. In western cultures, this property would contain the 'first' and 'middle' name Note that in some cultures, the given name is placed after the family name. Note also that this property contains multiple elements, so it can handle those situations where a person has more than one 'middl name." - HL7 V3
Patient	Last Name	Last name of the patient		FHIM::Datatypes::PersonName	family	"The portion of a person's name that reflects the genealogy of the person. In western cultures, this is 'last' name. In eastern cultures, the family name appears before the person's given name(s). In some cultures (e.g. Eritrea) the family name of a son is the first name of his father." - HL7 V3
Patient	patientBirthDate	Patient date of birth		FHIM::AdverseEventReporting::AbstractReportSubjectObservation	dateOfBirth	The date of birth of the person in question, which w be used to calculate their age at time that the advers event or reaction occurred. Note that this property be derived from the person's demographics information.
Patient	patientGender	Patient gender		FHIM::Person::Person::administrativeGender	sex	The sex of the person in question. Note that this property can be derived from the person's demographics information.

FHIM Data Elements in mapping spreadsheet

Within the FHIM class documentation and the property documentation columns, there are references to which of the standards to which the specific element should be bound.

Map Scenario Element Requirements to NIEM

To map scenario element requirements to NIEM, add columns to your mapping spreadsheet for NIEM values to aid in mapping the scenario elements from FHIM semantic equivalents to the NIEM semantic equivalents:

Scenario Required Object Class	Scenario Required Object Class	Scenario Required Object Class	FHIM Qualified Class Name	FHIM Property Name	FHIM Data Type	NIEM Object Class	NIEM Element	NI Ele Defi
Scenario Required Object Class			The name of the data object class you need for exchange					
Scenario Required Element			The name of the data element you need for exchange					
Scenario Required Definition			A clear and cogent definition of your data element					
FHIM Qualified Class Name			The fully qualified class name from FHIM					
FHIM Property Name			The name of the property from FHIM that most closely matches your sce					
FHIM Data Type			The data type of the FHIM property					
NIEM Object Class			The NIEM object class name					
NIEM Element			The NIEM element name					
NIEM Element Definition			The NIEM element definition					
NIEM Element Type			The data type of the NIEM element					

Based on the FHIM class and element definitions, identify potential NIEM elements. Then validate these NIEM elements by attempting to find any pre-existing elements in NIEM Core or other relevant NIEM domains. There are four ways to view NIEM domain elements:

- Option 1: View the UML model of each domain at http://FHIMview.com/NIEMview
- Option 2: View the UML model of each domain using the Movement tool at https://beta.movement.niem.gov/
- Option 3: Download the zip file of the schema for all data elements from http://niem.github.io/niem-releases/
- Option 4: Search for matching NIEM elements using the NIEM Subset Schema Generator Tool at https://tools.niem.gov/niemtools/ssgt/index.iepd

If you find a similar pre-existing NIEM element, you will need to decide if the pre-existing element meets the original scenario requirements and health IT standard, with the preference for using the pre-existing NIEM domain element rather than creating a new element.

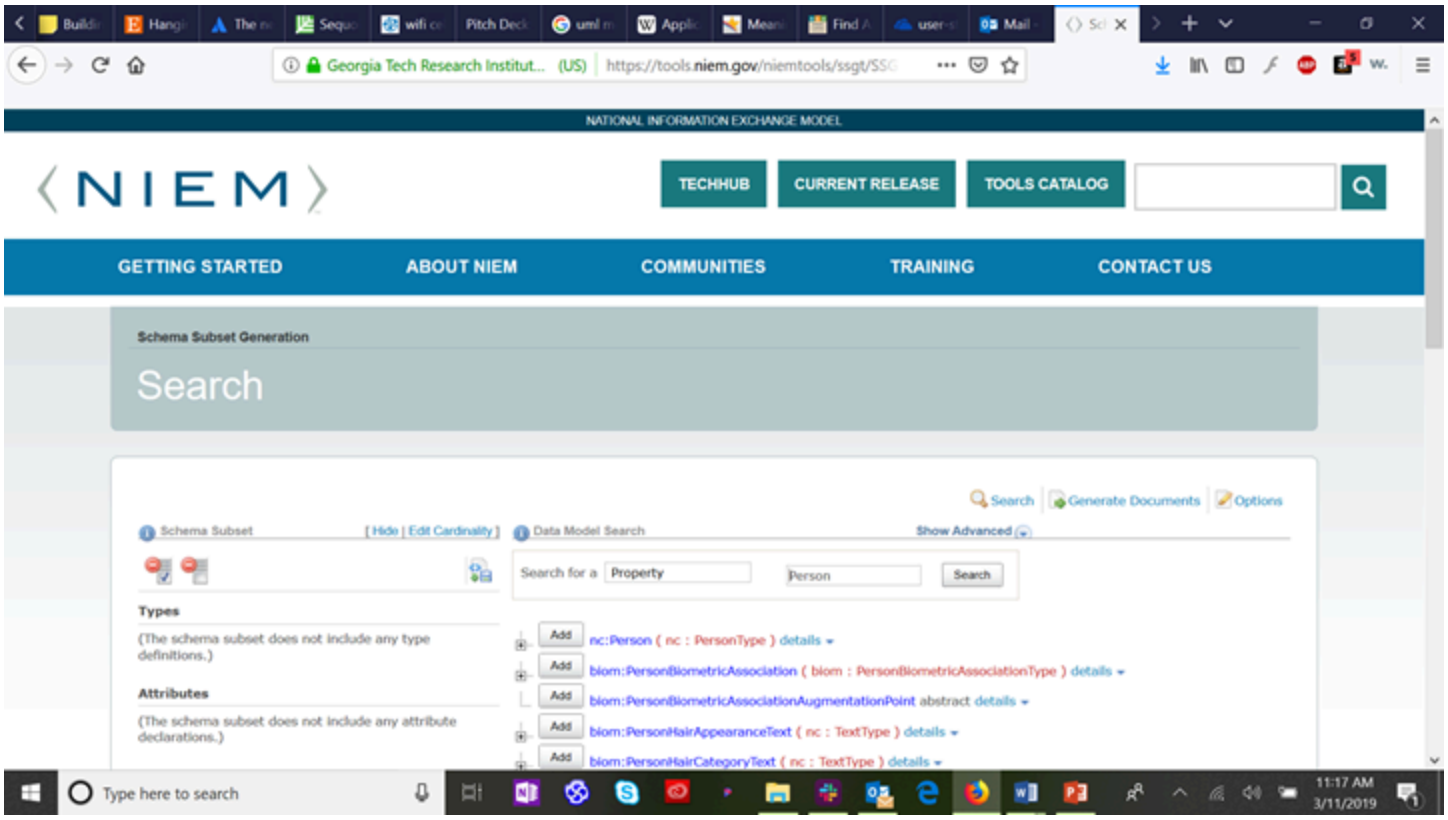
Scenario Required Object Class	Scenario Required Object Class	Scenario Required Object Class	FHIM Qualified Class Name	FHIM Property Name	FHIM Data Type	NIEM Object Class	NIEM Element	NI Ele Defi
Scenario Required Object Class			The name of the data object class you need for exchange					
Scenario Required Element			The name of the data element you need for exchange					
Scenario Required Definition			A clear and cogent definition of your data element					
FHIM Qualified Class Name			The fully qualified class name from FHIM					
FHIM Property Name			The name of the property from FHIM that most closely matches your sce					
FHIM Data Type			The data type of the FHIM property					
NIEM Object Class			The NIEM object class name					
NIEM Element			The NIEM element name					
NIEM Element Definition			The NIEM element definition					
NIEM Element Type			The data type of the NIEM element					

NIEM Data Elements in mapping spreadsheet

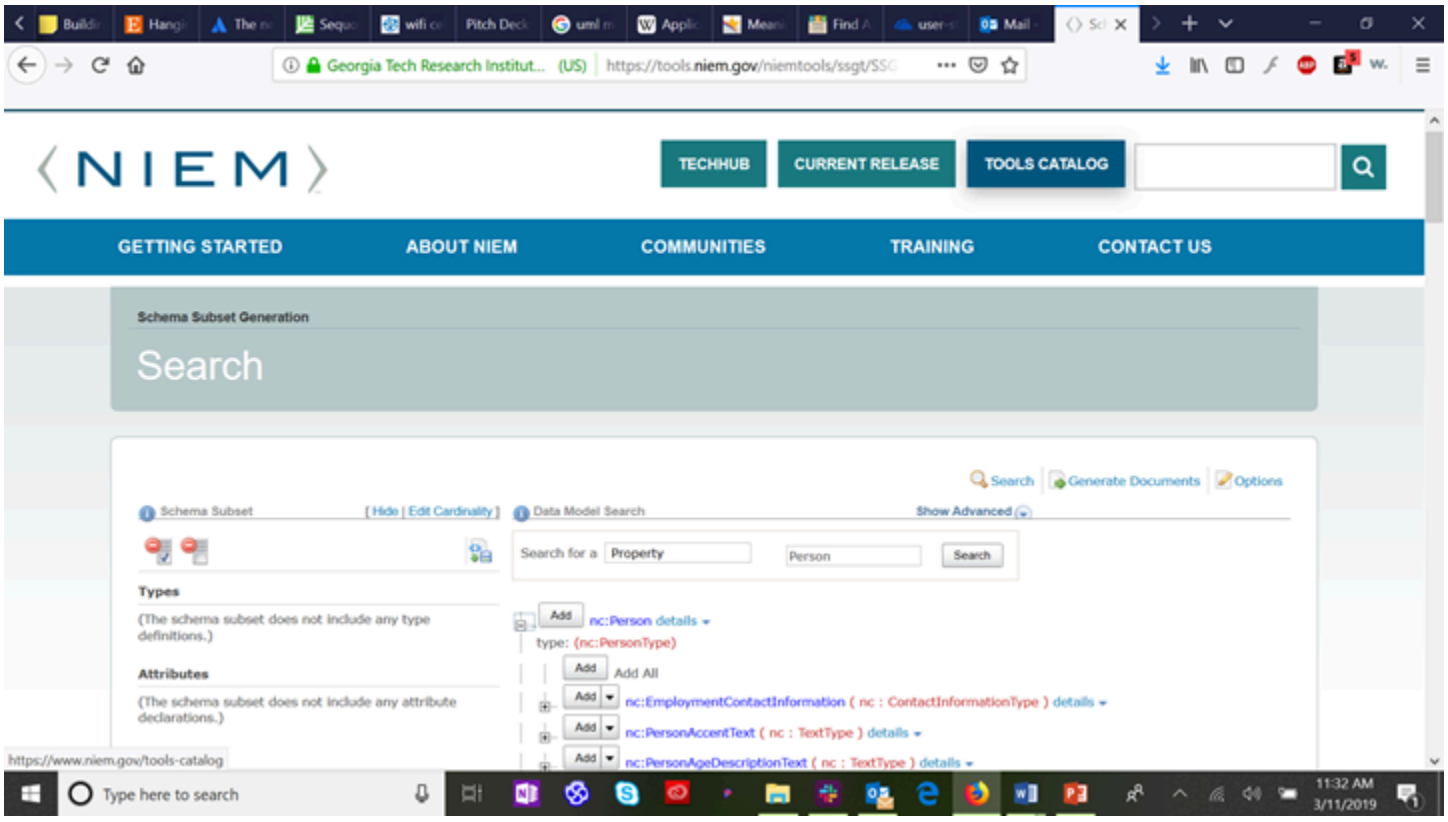
Generate NIEM IEPD Subset

Using NIEM element mappings usually requires a number of supporting classes, types and elements defined across the NIEM domains. However, most IEPD implementations don't require the full breadth and depth of all the NIEM domain schema definitions. To ease assembling the necessary subset of the NIEM XML schema definitions (XSD) for a compliant IEPD, you may use the NIEM Schema Subset Generator tool (SSGT) available at https://tools.niem.gov/niemtools/ssgt/index.iepd.

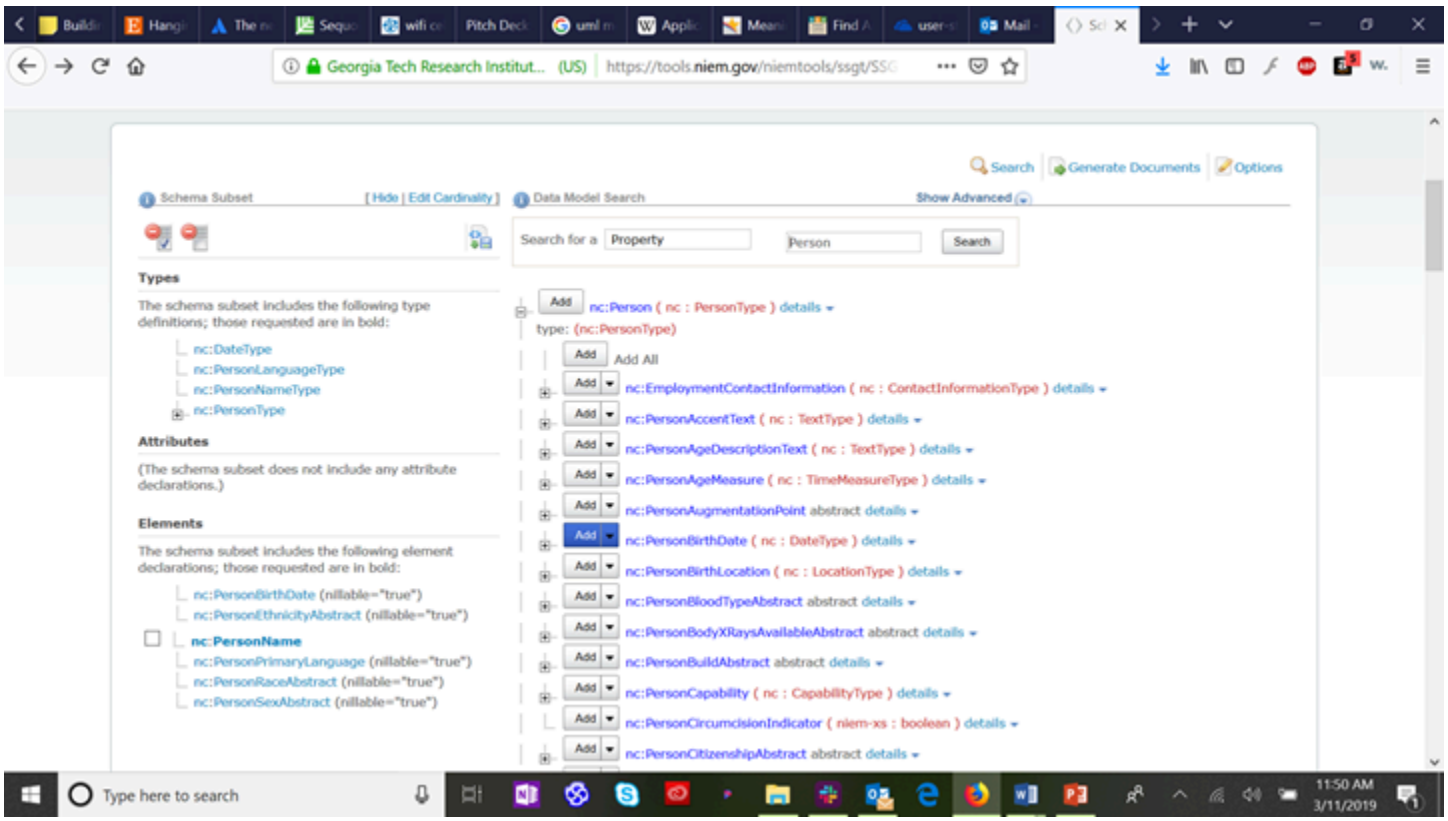
The following figure shows the results for searching in the SSGT for the person class:



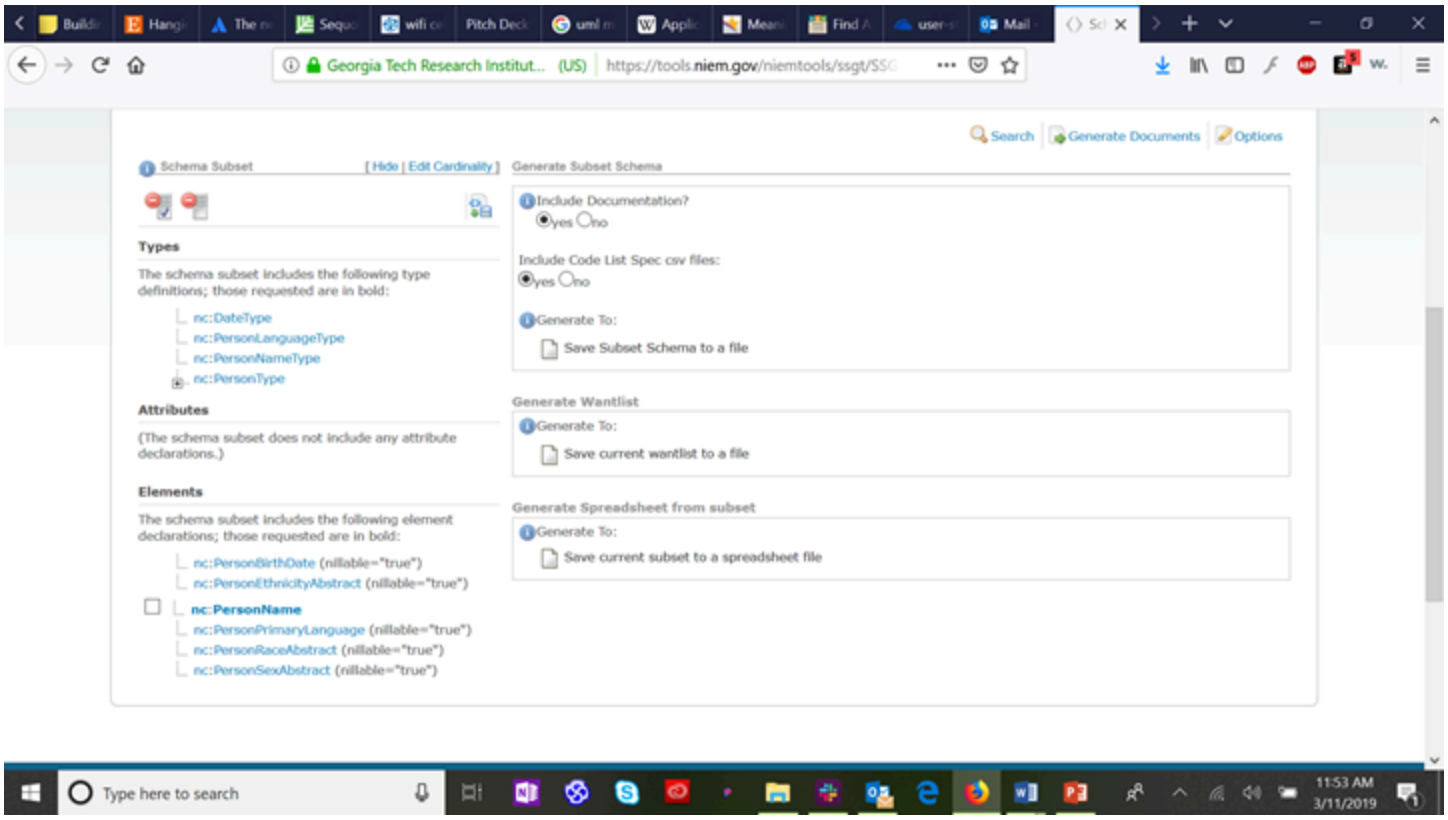
Since we know that the six patient demographic elements we are mapping are elements of the class "Person" in the NIEM Core domain, we click on the "plus" icon to expand the class definition and see the sub-elements of nc:Person.



Now we find each of the six patient demographics that are sub-elements of nc:Person, and click on the "add" button next to each. By default, this will generate XML schema with a nullable cardinality (which allows zero to infinite instances of the selected element). If you wish to restrict the schema to require at least one element (e.g., Person:Name) or no more than one element (e.g., Person:BirthDate), select the "down arrow" next to the "add" button to select the appropriate cardinality. The selected types and elements will appear in the left column:



Finally, select "generate documents" from the upper right corner of the SSGT app to generate your base IEPD documents.



Click on “save subset schema to a file,” which will bring up a standard file save dialog screen. The resulting files will include a “wantlist” (manifest) of the schema types and elements required to completely define your subset schema and a spreadsheet view of your subset schema. The generate wantlist and general spreadsheet options are for advanced users that are generating a large subset schema incrementally over time.

You now have the basic IEPD files generated and saved on your system. You may now further fill out documentation files in your IEPD folders and use the IEPD schema files to develop NIEM-compliant information exchange code. For more information on working with NIEM see https://www.niem.gov/training.

Map NIEM to Vocabulary Code Sets

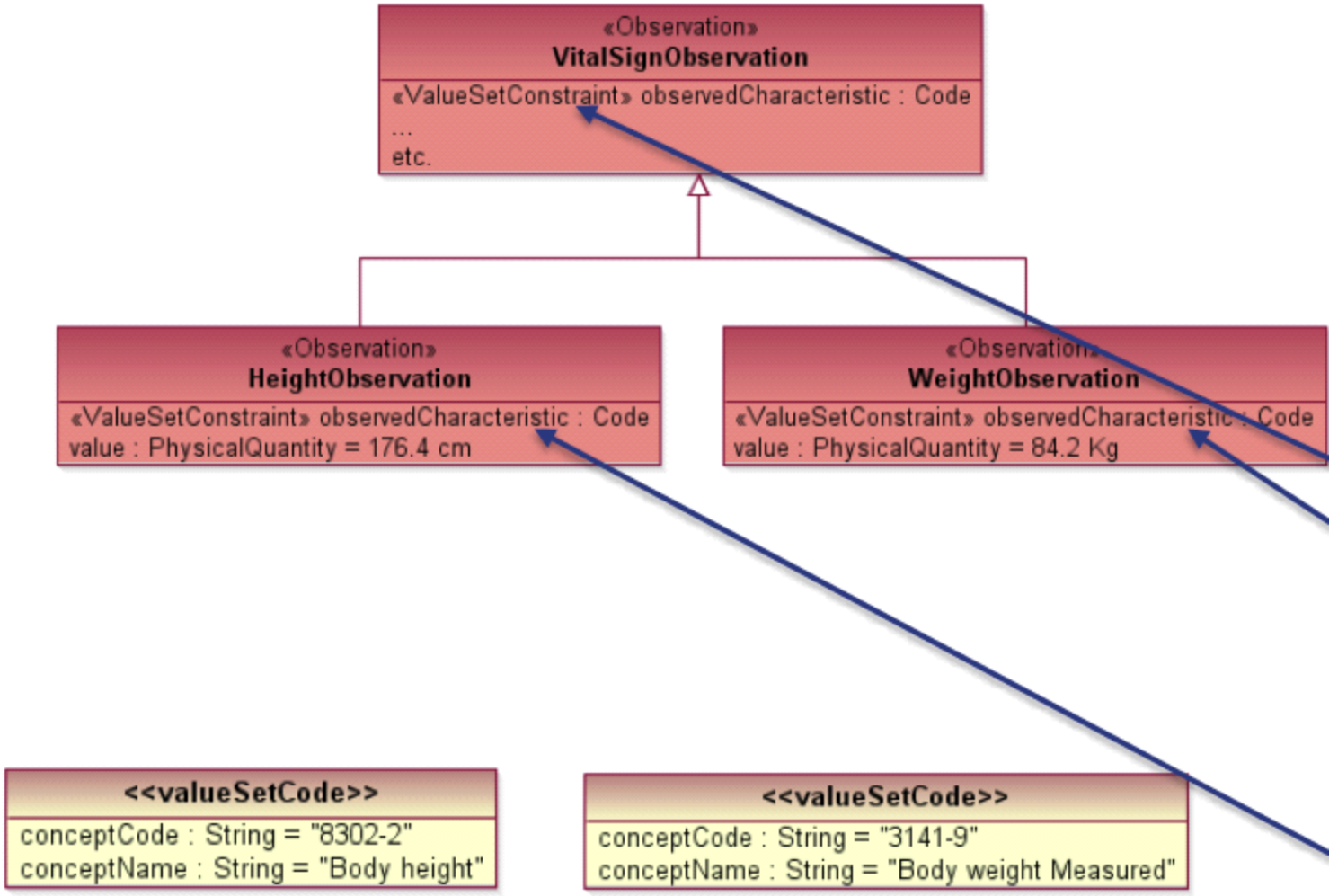
The final step is to add columns to the spreadsheet to map between individual NIEM elements and specific healthcare information vocabulary standard(s).

Most of the applicable standards have been discussed in NIEM Health 101. The table below serves as a refresher:

Acronym	Title	Description
SNOMED	Systematized Nomenclature of Medicine-Clinical Terms	Nomenclature copyrighted by the College of American Pathologists; includes diseases, clinical findings, etiologies, procedures, and outcomes
ICD-10	International Classification of Diseases, 10th Edition	Overlaps with SNOMED in diseases, events, and findings

Acronym	Title	Description
LOINC	Logical observation Identifiers Names and Codes	Overlaps with SNOMED in findings and measures
CPT	Current Procedural Terminology	Overlaps with SNOMED in procedures/interventions concepts. Not as granular as LOINC; also used for insurance data exchange
X12	Claim Status Codes	Similar to CPT but focused for insurance claims, not specific enough for clinical reporting
UMLS	National Library of Medicine's Unified Medical Language System ®	Meta vocabulary collection that includes ICD-10, LOINC, CPT, and SNOMED

For example, the value-set bindings for VitalSignObservation to LOINC vocabulary:



The FHIM includes "detailed clinical models" for certain common concepts, explicitly showing first-class classes for each concept. It is important to note that after reachin a certain point in the model (VitalSignObservation, in this case), the detailed clinical model classes could be *generated* from an ontology or a knowledge base

Add and label columns for vocabulary values to aid in mapping from scenario requirements:

Scenario Required Object Class	Scenario Required Object Class	Scenario Required Object Class	FHIM Qualified Class Name	FHIM Property Name	FHIM Data Type	NIEM Object Class	NIEM Element	NIEM Element Definition	NIEM Element Type	Voca So Acr
Scenario Required Object Class			The name of the data object class you need for exchange							
Scenario Required Element			The name of the data element you need for exchange							
Scenario Required Definition			A clear and cogent definition of your data element							
FHIM Qualified Class Name			The fully qualified class name from FHIM							
FHIM Property Name			The name of the property from FHIM that most closely matches your scenario require							
FHIM Data Type			The data type of the FHIM property							
NIEM Object Class			The NIEM object class name							
NIEM Element			The NIEM element name							
NIEM Element Definition			The NIEM element definition							
NIEM Element Type			The data type of the NIEM element							
Vocabulary Source Acronym			The acronym for the Vocabulary in which values are defined							
Vocabulary Name			The name of the vocabulary that defines values for this element							
Vocabulary Values Range/Set			A range or set that is allowed for this scenario required element							

Conclusion

This tutorial demonstrated how to use health IT standard information models such as the Federal Health Information Model (FHIM) to map NIEM Health use case requirements to NIEM elements and health information exchange standards elements, and how to create Fast Healthcare Interoperability Resources (FHIR) elements reflecting the use case requirements. In addition, NIEM Health201 strives to identify key public health IT data elements that are needed for exemplar NIEM Health community scenarios.

NIEM Health Elements Inventory

- Introduction
- NIEM Core (38)
- Agriculture
- Biometrics (2)
- Chemical, Biological, Radiological, and Nuclear
- Cyber (new)
- Emergency Management (6)
- Grants
- Housing (new)
- Human Services (75)
- Immigration
- Infrastructure Protection
- Intelligence
- International Trade
- Justice (157+61)
- Maritime
- Military Operations
- Public Safety Canada (new)
- Screening (1)
- Statistics (new)
- Surface Transportation
- USAID

Cait Ryan, Project Manager

Brian Handspicker, Sr Consultant Health IT

As of June 17, 2019

Download the inventory.

Introduction

This is an tentative inventory of healthcare elements and types defined across the NIEM domains. The goal of this exercise is to determine what healthcare-related elements have already been defined across the NIEM Domains, compare those elements for completeness and standards compliance with ONC USCDI and health industry standards, and identify gaps that may need to be filled by NIEM Health definitions.

The Office of the National Coordinator for Health Information Technology (ONC) has created a roadmap, the United States Core Definitions for Interoperability (USCDI) of healthcare information. This roadmap proposes adoption and definition of specific groups of common health elements over three years. The elements defined for 2019 are:

- Assessment and plan of treatment

- Care Team members
- Clinical Notes
- Goals
- Health concerns
- Immunizations
- Laboratory
- Medications
- Patient Demographics
- Problems
- Procedures
- Provenance
- Smoking Status
- Unique device identifier
- Vital signs

This inventory was created by first searching on USCDI defined class and element names, then searching on cognates of those names, then searching on likely synonyms, and finally visually skimming the NIEM domain XML files for additional candidates. Currently we have identified healthcare related elements in the following domains:

Core - 38

Biometrics - 2

Emergency Management - 6

Human Services - 75

Justice - 157 + 61 Mental Health

Screening - 1

This inventory is not certain to be complete until sufficient review has shaken out the better disguised medical or healthcare-oriented elements.

NIEM Core (38)

nc:MedicalConditionDoctor (nc:PersonType) - A doctor who treats a person for a medical condition.

nc:PatientMedicalFacility (nc:OrganizationType) - A medical facility receiving patient.</p>

nc:Person (nc:PersonType) - A human being.

> Although Person is an object/class defined in NIEM Core, this has been included in this specification because it is the basis of role of patient, doctor, etc plus many class elements are associated with clinical records, e.g. height, weight, etc.
>
> Likely candidate health elements highlighted.

nc: EmploymentContactInformation (nc:ContactInformationType)
nc: PersonAccentText (TextType)
nc: PersonAgeDescriptionText (TextType)
nc: PersonAgeMeasure (nc:TimeMeasureType)
nc: PersonBirthDate (nc:DateType)
nc: PersonBirthLocation (nc:LocationType)
nc: PersonBloodTypeAbstract (abstract) This term has synonym entries
nc: PersonBodyXRaysAvailableAbstract (abstract) This term has synonym entries
nc: PersonBuildAbstract (abstract) This term has synonym entries
nc: PersonCapability (nc:CapabilityType)
nc: PersonCircumcisionIndicator (boolean)
nc: PersonCitizenshipAbstract (abstract) This term has synonym entries
nc: PersonComplexionText (TextType)
nc: PersonDeathDate (nc:DateType)
nc: PersonDependentQuantity (QuantityType)
nc: PersonDescriptionText (TextType)
nc: PersonDigitalImage (nc:ImageType)
nc: PersonDigitizedSignatureImage (nc:ImageType)
nc: PersonDisguiseDescriptionText (TextType)
nc: PersonEducationLevelText (TextType)
nc: PersonEmergencyContactInformation (nc:ContactInformationType)
nc: PersonEthnicityAbstract (abstract) This term has synonym entries
nc: PersonEyeColorAbstract (abstract) This term has synonym entries
nc: PersonEyewearAbstract (abstract) This term has synonym entries
nc: PersonFacialHairAbstract (abstract) This term has synonym entries
nc: PersonGeneralAppearanceDescriptionText (TextType)
nc: PersonHairAppearanceText (TextType)
nc: PersonHairCategoryText (TextType)
nc: PersonHairColorAbstract (abstract) This term has synonym entries
nc: PersonHairLengthAbstract (abstract) This term has synonym entries
nc: PersonHairStyleAbstract (abstract) This term has synonym entries
nc: PersonHandednessText (TextType)
nc: PersonHeightDescriptionText (TextType)
nc: PersonHeightMeasure (nc:LengthMeasureType)
nc: PersonHomeContactInformation (nc:ContactInformationType)
nc: PersonHumanResourceIdentification (nc:IdentificationType)
nc: PersonInjury (nc:InjuryType)
nc: PersonJewelryDescriptionText (TextType)
nc: PersonLanguageEnglishIndicator (boolean)
nc: PersonLearningDisabilityText (TextType)
nc: PersonLicenseIdentification (nc:IdentificationType)
nc: PersonLivingIndicator (boolean)
nc: PersonMedicalCondition (nc:MedicalConditionType)
nc: PersonMedicalDescriptionText (TextType)
nc: PersonMedicalFileIndicator (boolean)
nc: PersonMedicationRequiredText (TextType)
nc: PersonMentalStateText (TextType)
nc: PersonMilitarySummary (nc:MilitarySummaryType)
nc: PersonMoodDescriptionText (TextType)

nc: PersonName (nc:PersonNameType)
nc: PersonNationalIdentification (nc:IdentificationType)
nc: PersonNationalityAbstract (abstract) This term has synonym entries
nc: PersonOrganDonatorIndicator (boolean)
nc: PersonOrganDonorAbstract (abstract) This term has synonym entries
nc: PersonOtherIdentification (nc:IdentificationType)
nc: PersonPassportIdentification (nc:IdentificationType)
nc: PersonPhysicalDisabilityText (TextType)
nc: PersonPhysicalFeature (nc:PhysicalFeatureType)
nc: PersonPrimaryLanguage (nc:PersonLanguageType)
nc: PersonRaceAbstract (abstract) This term has synonym entries
nc: PersonReligionAbstract (abstract) This term has synonym entries
nc: PersonResidentAbstract (abstract) This term has synonym entries
nc: PersonSSNIdentification (nc:IdentificationType)
nc: PersonSecondaryLanguage (nc:PersonLanguageType)
nc:PersonSecurityClearanceAbstract (abstract) This term has synonym entries
nc: PersonSexAbstract (abstract) This term has synonym entries
nc: PersonSexualOrientationText (TextType)
nc: PersonSkinToneAbstract (abstract) This term has synonym entries
nc: PersonSpeechDescriptionText (TextType)
nc: PersonStateIdentification (nc:IdentificationType)
nc: PersonTaxIdentification (nc:IdentificationType)
nc: PersonUSCitizenIndicator (boolean)
nc: PersonUnionStatusAbstract (abstract) This term has synonym entries
nc: PersonVisionPrescriptionText (TextType)
nc: PersonWeightDescriptionText (TextType)
nc: PersonWeightMeasure (nc:WeightMeasureType)
nc: PersonXRayImage (nc:ImageType)

nc:MedicalCondition (nc:MedicalConditionType)

nc:PersonMedicalCondition (nc:MedicalConditionType) - A state of health for a person, on-going or present.

nc: MedicalConditionCauseText (TextType)
nc: MedicalConditionDateRange (nc:DateRangeType)
nc: MedicalConditionDescriptionText (TextType)
nc: MedicalConditionDoctor (nc:PersonType)
nc: MedicalConditionPresentIndicator (boolean)
nc: MedicalConditionSeverityText (TextType)
nc: MedicalConditionText (TextType)

nc:MedicalConditionText (nc:TextType) - A state of health, on-going or present.

nc:PersonMedicalDescriptionText (nc:TextType) - A description of the overall health of a

person.

nc:PersonMedicalFileIndicator (niem-xs:boolean) - True if a medical history file is known to exist for a person; false otherwise.

nc:PersonMedicationRequiredText (nc:TextType) - A medication and dosage required for a person.

nc:PersonMentalStateText (nc:TextType) - A mental condition of a person.

nc:PersonLivingIndicator (niem-xs:boolean) - True if a person is alive; false if a person is dead.

nc:PersonOrganDonorAbstract (abstract / This term has synonym entries) - A data concept for an organ a person is willing to donate upon death.

PersonOrganDonorText (TextType)

PersonOrganDonorCode (PersonOrganDonorCodeType)

nc:PersonInjury (nc:InjuryType) - A form of physical harm or damage sustained by a person.

InjuryCategoryAbstract (abstract) This term has synonym entries

InjuryCauserAbstract (abstract) This term has synonym entries

InjuryDate (nc:DateType)

InjuryDescriptionText (TextType)

InjuryLocationAbstract (abstract) This term has synonym entries

InjurySeverityAbstract (abstract) This term has synonym entries

InjurySustainer (nc:PersonType)

nc:PersonBirthDate (nc:DateType) - A date a person was born.

nc:PersonDeathDate (nc:DateType) - A date a person died or was declared legally dead.

nc:PersonBloodTypeAbstract (abstract / This term has synonym entries) - A data concept for a blood group and RH factor of a person.

nc:PersonBloodTypeText (TextType)

J:PersonBloodTypeCode (ncic:BLTCodeType)

J:PersonBloodTypeRapSheetCode (j:PersonBloodTypeCodeType)

Agriculture

[None Defined]

Biometrics (2)

bio:RadiographImageSeriesCodeText (nc:TextType) - A kind of Radiograph Image Series. Only integer and periods are in the code text. An example is 8.1.3.9.1.5 for the National Provider Identifier Number of Dentist that treated the patient.

bio:HealthCommentText (nc:TextType) - A comment noting any observable health issues impacting the data subject during the recording time segment.

Chemical, Biological, Radiological, and Nuclear

[None Defined]

Cyber (new)

[None Defined]

Emergency Management (6)

em:HospitalStatus (abstract) - *The top level container element for reporting status of any number of hospitals. The EDXL-HAVE has no independent routing mechanism, so it requires a routing mechanism that is consistent with the EDXL-DE distribution types. It must contain one or more Hospital elements.*

em:EMSCensus (em:TriagePatientCountType) - *A number detailing each triage patient kind the overall hospital currently has.*

- em:TriageBlackQuantity (nc:QuantityType) - *A count of the Number of deceased victims;*
- em:TriageGreenQuantity (nc:QuantityType) - *A count of the Number of victims with minor needs*
- em:TriageRedQuantity (nc:QuantityType) - *A count of the Number of victims with immediate needs*
- em:TriageYellowQuantity (nc:QuantityType) - *A count of the Number of victims with delayed needs*

em:EMSCapacity (em:TriagePatientCountType) - A number detailing each triage patient kind the overall hospital can accept.

- em:TriageBlackQuantity (nc:QuantityType) - *A count of the Number of deceased victims;*
- em:TriageGreenQuantity (nc:QuantityType) - *A count of the Number of victims with minor needs*
- em:TriageRedQuantity (nc:QuantityType) - *A count of the Number of victims with immediate needs*

- em:TriageYellowQuantity (nc:QuantityType) - *A count of the Number of victims with delayed needs*

em:AvailableQuantity (nc:QuantityType) - *A count of the vacant/available beds to which patients can be immediately transported.*

em:StaffingCode (have-codes:AvailabilityStatusCodeType) - *A status indicating the staffing levels at the hospital*

em:EOCPlanCode (have-codes:EOCPlanCodeType) - *A code indicating whether the hospital has activated its Emergency Operations Plan (EOP)*

Grants

[None Defined]

Housing (new)

[None Defined]

Human Services (75)

hs:Patient (hs:PatientType)

Definition: A person receiving medical care.

nc:RoleOfPerson (nc:PersonType)

hs:MedicalSummary (hs:MedicalSummaryType)

hs:PatientLocationAbstract (abstract) This term has synonym entries

hs:PatientNameOfAnimalText (TextType)

hs:ProviderEntity (nc:EntityType)

hs:ReportingEntity (nc:EntityType)

hs:SpeciesAbstract (abstract) This term has synonym entries

CustodialEntity (nc:EntityType)

hs:**MedicalSummary** (hs:MedicalSummaryType)

Definition: A summary of a person's medical information

nc:MedicalCondition (nc:MedicalConditionType)

nc:MedicalConditionCauseText (TextType)

nc:MedicalConditionDateRange (nc:DateRangeType)

nc:MedicalConditionDescriptionText (TextType)

nc:MedicalConditionDoctor (nc:PersonType)

nc:MedicalConditionPresentIndicator (boolean)

nc:MedicalConditionSeverityText (TextType)

nc:MedicalConditionText (TextType)

hs:ClinicalObservation (hs:ClinicalObservationType)

hs:ObservationCategoryText (TextType)

hs:ObservationPrimaryIndicator (boolean)

hs:MedicalEncounter (nc:ActivityType)

hs:ServiceEvent (nc:ActivityType)

hs:**PatientLocationAbstract** (abstract) This term has synonym entries

Definition: A data concept for a location of a patient.

hs:PatientLocationCode (PatientLocationCodeType)

hs:PatientLocationText (TextType)

hs:**PatientLocationCode** (hs:PatientLocationCodeType)

Definition: A location of a patient.

hs:**PatientLocationText** (nc:TextType)

Definition: A location of a patient.

hs:**MedicalEncounter** (nc:ActivityType)

Definition: An encounter between a patient and applicable providers.

hs:**Prescription** (hs:PrescriptionType)

Definition:

nc:Drug (nc:DrugType)

hs:Dispenser (nc:EntityType)

hs:Prescriber (nc:EntityType)

hs:PrescriptionLastTakenDate (nc:DateType)

hs:PrescriptionTreatmentReasonText (TextType)

hs:PrescriptionUseInstructionsText (TextType)

nc:**Drug** (nc:DrugType)

Definition: A legal or illegal substance that can be used as a medicine or a narcotic.

hs:DEAClassScheduleAbstract (abstract) This term has synonym entries

hs:DrugCoverageExceptionReasonText (TextType)

hs:DrugDaysSupplyQuantity (QuantityType)

hs:DrugStrengthText (TextType)

nc:**DrugDispensedQuantity** (nc:QuantityType)

Definition: A total count of a drug dispensed to a patient.

hs:**PrescriptionSoldDate** (nc:DateType)

Definition: A date on which a prescription received by a patient or a patient's agent.

hs:**StateLicenseIdentification** (nc:IdentificationType)

Definition: An identification of a license giving an organization the right to prescribe drugs for patients.

hs:**MentalHealthHospitalizationHistoryDescriptionText** (nc:TextType)

Definition: A description of past in-patient hospitalizations due to mental health issues.

hs:**PatientNameOfAnimalText** (nc:TextType)

Definition: A name of a non-human patient.

hs:**SpeciesAbstract** (abstract) This term has synonym entries

Definition: A data concept for a kind of species, to differentiate humans from other potential patients.

hs:SpeciesCode (SpeciesCodeType)

hs:SpeciesText (TextType)

hs:**SpeciesText** (nc:TextType)

Definition: A kind of species, to differentiate humans from other potential patients.

hs:**SpeciesCode** (hs:SpeciesCodeType)

Definition: A kind of species, to differentiate humans from other potential patients.

hs:**HealthDetails** (hs:HealthDetailsType)

Definition: A person's mental health, substance abuse, medical health and medications.

hs:MedicalHealthDetails (hs:MedicalHealthDetailsType)
hs:MedicationUse (hs:MedicationUseType)
hs:MentalHealthDetails (hs:MentalHealthDetailsType)
hs:Prescription (hs:PrescriptionType)
hs:SubstanceAbuseDetails (hs:SubstanceAbuseDetailsType)

hs:**MedicalHealthDetails** (hs:MedicalHealthDetailsType)

Definition: A person's medical health details.

hs:DentalHistoryDescriptionText (TextType)
hs:HearingDisabilityIndicator (boolean)
hs:MedicalHistoryDescriptionText (TextType)
hs:OtherPhysicalDisabilityIndicator (boolean)
hs:VisualDisabilityIndicator (boolean)

hs:**MedicationUse** (hs:MedicationUseType)

Definition: A set of details about medication use for an individual.

hs:Dispenser (nc:EntityType)
hs:Medication (nc:DrugType)
hs:MedicationDispensingDate (nc:DateType)
hs:MedicationDoseMeasure (nc:MeasureType)
hs:MedicationLastTakenDate (nc:DateType)
hs:MedicationTreatmentReasonText (TextType)
hs:MedicationUseFrequencyAbstract (abstract) This term has synonym entries
hs:Prescriber (nc:EntityType)
hs:PrescriptionUseInstructionsText (TextType)

hs:**MentalHealthDetails** (hs:MentalHealthDetailsType)

Definition: An element that contains information about a person's mental health including social history and psychosocial information.

hs:DevelopmentalDisabilityCategoryAbstract (abstract) This term has synonym entries
hs:MentalHealthHistoryDescriptionText (TextType)
hs:MentalHealthHospitalizationHistoryDescriptionText (TextType)

hs:**Prescription** (hs:PrescriptionType)

Definition: A set of written instructions from a physician or dentist to a druggist concerning the form and dosage of a drug to be issued to a given patient

hs:Drug (nc:DrugType)
hs:Dispenser (nc:EntityType)
hs:Prescriber (nc:EntityType)
hs:PrescriptionLastTakenDate (nc:DateType)
hs:PrescriptionTreatmentReasonText (TextType)
hs:PrescriptionUseInstructionsText (TextType)

hs:**SubstanceAbuseDetails** (hs:SubstanceAbuseDetailsType)

Definition: A set of details about a person's substance abuse.

hs:Substance (nc:SubstanceType)
hs:DrugUseAgeMeasure (nc:TimeMeasureType)
hs:DrugUseFrequencyAbstract (abstract) This term has synonym entries
hs:DrugUseIndicator (boolean)
hs:PreferredDrugName (TextType)
hs:SubstanceAbuseCurrentDescriptionText (TextType)
hs:SubstanceAbusePastDescriptionText (TextType)

hs:**MentalHealthHistoryDescriptionText** (TextType)

Definition: A description of known mental health history, including behavioral health and psychological or psychiatric assessments, diagnoses, or treatments.

hs:**MentalHealthHospitalizationHistoryDescriptionText** (TextType)

Definition: A description of past in-patient hospitalizations due to mental health issues.

hs:**ClinicalObservation**(hs:ClinicalObservationType)

Definition: A clinical observation of a person.

ObservationCategoryText (TextType)
ObservationPrimaryIndicator (boolean)

hs:**HealthInsurance** (hs:HealthInsuranceType)

Definition: A health insurance policy.

hs:**HealthInsurancePolicyPrimaryInsuredFullName** (PersonNameTextType)

hs:**ProtectedHealthInformationIndicator** (niem-xs:ban>

Definition: True if a piece of information is Protected Health Information (PHI); false otherwise.

hs:**NPIIdentification** (nc:IdentificationType)

Definition: An identification called the National Provider Identifier, which includes a unique 10-digit identification number issued to health care providers in the United States by the Centers for Medicare and Medicaid Services (CMS).

j:**SubjectIntakeConstraintDayQuantity** (nc:QuantityType)

Definition: A count of days that will calculate how rapid the subject needs to go through intake with a provider following referral date.

hs:**ObservationCategoryText** (nc:TextType)

Definition: A kind of an observation (e.g. a diagnosis, lab results, allergy indications, etc.).

Immigration

[N.B. The following "provider" element is *NOT* a healthcare element. I've included it as a warning.]

i:**Provider** (nc:PersonType)

Definition: A person who has another person as a dependent.

Infrastructure Protection

[None Defined]

Intelligence

[None Defined]

International Trade

[None Defined]

Justice (157+61)

[N.B. Justice has 61 MentalHealth elements defined that are not documented here, yet.]

j:EMSAgency (nc:OrganizationType)

Definition: An EMS agency providing transport to medical facility that receives patient.

j:CrashPersonEMSTransportation (j:CrashPersonEMSTransportationType)

Definition: A transportation of the patient by the EMS unit to a medical facility.

EMSAgency (nc:OrganizationType)

j:InjuredTransportationSourceAbstract (abstract) This term has synonym entries

Definition: A data concept for a type of transport to the medical facility receiving the patient.

InjuredTransportationSourceCode (InjuredTransportationSourceCodeType)

j:InjuredTransportationSourceCode (mmucc:InjuredTransportationSourceCodeType)

Definition: A type of transport to the medical facility receiving the patient.

j:ProgramTitleText (nc:TextType)

Definition: A title or name of a class or treatment program a student or patient is participating in.

j:ProgramBehaviorText (nc:TextType)

Definition: A description of the behavior of a student or patient for a class or program.

j:ProgramAttendanceText (nc:TextType)

Definition: A description of the attendance record of a student or patient for a class or program.

j:ProgramResultantCertificationText (nc:TextType)

Definition: A certification received as a result of a student or patient completing a class or treatment program.

j:ProgramResultantCertificationExpirationDate (nc:DateType)

Definition: A date when a certification received as a result of a student or patient completing a class or treatment program expires.

j:ProgramSatisfactoryCompletionIndicator (niem-xs:boolean)

Definition: True if a student or patient satisfactorily completed the class or program; false if they did not complete the program or did not complete it satisfactorily.

j:MentalHealthFunctionLevelScoreNumeric (nc:NumericType)

Definition: A number measuring the ability of the subject to function and adapt to various problems in living. The Global Assessment of Functioning (GAF) uses a 1-100 measurement scale with 10 and below representing a persistent danger of self injury or hurting others.

j:TreatmentDiagnosisText (nc:TextType)

Definition: A determining or analysis of the manner in which to treat the cause or nature of a problem

j: PersonDentalCharacteristicText (nc:TextType)

Definition: A visible problem or characteristic of a person's teeth.

j:PersonHearingProblemIndicator (niem-xs:boolean)

Definition: True if a person has difficulty hearing; false otherwise.

j:DrugUseAssessmentYearsOfProblemUseQuantity (nc:QuantityType)

Definition: A number indicating the years the subject has had drug-related problems.

j: **SubjectDrugUseAssessment** (j:DrugUseAssessmentType)

j:DrugUseAssessment (j:DrugUseAssessmentType)

Definition: An assessment process to determine a subject's past drug use and/or dependency.

j:DrugUseAssessmentDrugDependencyIndicator (boolean)

j:DrugUseAssessmentDrugFreeDuration (duration)

j:DrugUseAssessmentDrugTest (j:DrugTestType)

j:DrugTestCategoryAbstract (abstract) This term has synonym entries

j:DrugTestCategoryCode (DrugTestCategoryCodeType)

j:DrugTestDrugSampleSizeQuantity (QuantityType)

j:DrugTestDrugTestedCategoryAbstract (abstract) This term has synonym entries

j:DrugTestDrugTestedCategoryText (TextType)

j:DrugTestResultAbstract (abstract) This term has synonym entries

j:DrugTestResultCode (DrugTestResultCodeType)

j:DrugUseAssessmentDrugTreatmentProgramsAttendedQuantity (QuantityType)

j:DrugUseAssessmentFirstUsedAgeQuantity (QuantityType)

j:DrugUseAssessmentInPrisonDrugTreatmentProgramsAttendedQuantity (QuantityType)

j:DrugUseAssessmentLastDrugTestDate (nc:DateType)

j:DrugUseAssessmentPrimaryDrugOfChoiceAbstract (abstract) This term has synonym entries

j:DrugUseAssessmentPrimaryDrugOfChoiceText (TextType)

j:DrugUseAssessmentRelapseTriggerText (TextType)

j:DrugUseAssessmentSecondaryDrugOfChoiceAbstract (abstract) This term has synonym entries

j:DrugUseAssessmentSecondaryDrOfChoiceText (TextType)

j:DrugUseAssessmentUsedNeedleIndicator (boolean)

j:DrugUseAssessmentYearsOfProblemUseQuantity (QuantityType)

j:DrugUseDrug (nc:DrugType)

j:DEAClassScheduleAbstract (abstract) This term has synonym entries

j:DrugCoverageExceptionReasonText (TextType)

j:DrugDaysSupplyQuantity (QuantityType)

j:DrugStrengthText (TextType)

j:**AlcoholUseAssessment** (j:AlcoholUseAssessmentType)

Definition: An assessment specific to a subject's alcohol use.

j:AlcoholUseAssessmentAlcoholAbuseFirstAgeUsedNumeric (NumericType)

j:AlcoholUseAssessmentAlcoholAbuseLengthOfTreatmentNumeric (NumericType)

j:AlcoholUseAssessmentAlcoholAbuseMaximumDurationAbstainedNumeric (NumericType)

j:AlcoholUseAssessmentAlcoholConsumptionFrequencyText (TextType)

j:AlcoholUseAssessmentAlcoholUseConsumptionQuantity (QuantityType)

j:AlcoholUseAssessmentComment (nc:CommentType)

j:AlcoholUseAssessmentSummaryText (TextType)

j:**SubjectAssessmentMentalHealthAssessment** (j:MentalHealthAssessmentType)

j:**SubjectMentalHealthAssessment** (j:MentalHealthAssessmentType)

j:**MentalHealthAssessment** (j:MentalHealthAssessmentType)

Definition: An activity of reviewing and assessing a subject's mental health condition.

j:MentalHealthAssessmentAbstractOfJudgmentDate (nc:DateType)

j:MentalHealthAssessmentAbstractOfJudgmentReviewedIndicator (boolean)

j:MentalHealthAssessmentCorrectionalCounselorIReviewDate (nc:DateType)

j:MentalHealthAssessmentDMHPsychiatricReportAuthorText (TextType)

j:MentalHealthAssessmentDMHPsychiatricReportReviewDate (nc:DateType)

j:MentalHealthAssessmentDMHPsychiatricReportReviewedIndicator (Boolean)

j:MentalHealthAssessmentDate (nc:DateType)

j:MentalHealthAssessmentFactor (j:MentalHealthAssessmentFactorType)

- j:MentalHealthAssessmentFactorCategoryAbstract (abstract) This term has synonym entries
  - j:MentalHealthAssessmentFactorCategoryText (TextType)
- j:MentalHealthAssessmentFactorDetailsText (TextType)

j:MentalHealthAssessmentLegalStatusSummaryReviewedIndicator (boolean)

j:MentalHealthAssessmentMentallyDisorderedOffenderIndicator (boolean)

j:MentalHealthAssessmentPsychiatricReportAuthorText (TextType)

j:MentalHealthAssessmentPsychiatricReportDate (nc:DateType)

j:MentalHealthAssessmentPsychiatricReportReviewedIndicator (boolean)

j:MentalHealthAssessmentReferral (nc:ReferralType)

- j:ReferralActivity (nc:ActivityType)
- j:ReferralEntity (nc:EntityType)
- j:ReferralIssuer (nc:EntityType)
- j:ReferralItem (nc:ItemType)
- j:ReferralMandatoryIndicator (boolean)
- j:ReferralPerson (nc:PersonType)
- j:ReferralSupervisor (nc:EntityType)

j:MentalHealthAssessmentReferralSexualViolentPredatorIndicator (boolean)

j:MentalHealthAssessmentScreeningStaffMember (j:StaffMemberType)

- j:RoleOfPerson (nc:PersonType)
- j:StaffMemberAffiliatedPrimaryProgram (nc:ProgramType)
- j:StaffMemberAppliedForce (j:ForceType)
- j:StaffMemberAssignedWeapon (nc:WeaponType)
- j:StaffMemberAssociatedSubject (j:SubjectType)
- j:StaffMemberBadgeNumberID (string)
- j:StaffMemberDisciplineText (TextType)
- j:StaffMemberIdentification (nc:IdentificationType)
- j:StaffMemberIncident (nc:IncidentType)
- j:StaffMemberJusticeSupervisionIndicator (boolean)
- j:StaffMemberKeyPositionIndicator (boolean)
- j:StaffMemberLocation (nc:LocationType)
- j:StaffMemberQualification (j:StaffMemberQualificationType)
- j:StaffMemberStatus (nc:StatusType)

j:MentalHealthAssessmentServicesReceivedAbstract (abstract) This term has synonym entries

- j:MentalHealthAssessmentServicesReceivedText (TextType)

j:MentalHealthAssessmentSubjectParticipatedIndicator (boolean)

j:MentalHealthAssessmentTestingRequiredIndicator (boolean)

j:MentalHealthCareLevelText (TextType)

j:MentalHealthCertification (j:CertificationType)

- j:CertificationExpirationDate (nc:DateType)
- j:CertificationExpirationReasonText (TextType)
- j:CertificationIdentification (nc:IdentificationType)
- j:CertificationIssuingAuthority (nc:EntityType)

j:CertificationName (ProperNameTextType)

j:CertificationReviewDate (nc:DateType)

j:MentalHealthOtherIndicator (boolean)

j:MentalHealthStatusAbstract (abstract) This term has synonym entries

j:MentalHealthStatusText (TextType)

j:MentalHealthSubjectClearedIndicator (boolean)

MentalHeathAssessmentIndependentEvaluationRequestedIndicator (boolean)

j:SubjectAssessmentMedicalAssessment (j:MedicalAssessmentType)

j:SubjectMedicalAssessment (j:MedicalAssessmentType)

j:**MedicalAssessment** (j:MedicalAssessmentType)

Definition: An activity of reviewing and assessing a subject's medical condition.

j:MedicalAssessmentClearanceAbstract (abstract) This term has synonym entries

j:MedicalAssessmentClearanceText (TextType)

j:MedicalAssessmentClearanceDentalText (TextType)

j:MedicalAssessmentClearanceEligibilityAbstract (abstract) This term has synonym entries

j:MedicalAssessmentClearanceEligibilityText (TextType)

j:MedicalAssessmentClearancePregnantSubjectTrimesterNumeric (NumericType)

j:MedicalAssessmentClearanceRestrictionsUrgentIndicator (boolean)

j:MedicalAssessmentClearanceSeizureLastDate (nc:DateType)

j:MedicalAssessmentComment (nc:CommentType)

j:MedicalAssessmentConsultationText (TextType)

j:MedicalAssessmentFunctionalText (TextType)

j:MedicalAssessmentLevelOfCareAbstract (abstract) This term has synonym entries

j:MedicalAssessmentLevelOfCareText (TextType)

j:MedicalAssessmentMedicalRiskAbstract (abstract) This term has synonym entries

j:MedicalAssessmentMedicalRiskText (TextType)

j:MedicalAssessmentNursingCareLevelAbstract (abstract) This term has synonym entries

j:MedicalAssessmentNursingCareLevelText (TextType)

j:MedicalAssessmentPermanentIndicator (boolean)

j:MedicalAssessmentRequiredMedication (nc:DrugType)

j:DEAClassScheduleAbstract (abstract) This term has synonym entries

j:DEAClassScheduleCode (DEAClassScheduleCodeCodeType)

j:DEAClassScheduleText (TextType)

j:DrugCoverageExceptionReasonText (TextType)

j:DrugDaysSupplyQuantity (QuantityType)

j:DrugStrengthText (TextType)

j:MedicalAssessmentSpecializedServiceAbstract (abstract) This term has synonym entries

j:MedicalAssessmentSpecializedServiceText (TextType)

j:MedicalAssessmentTemporaryAssessmentExpirationDate (nc:DateType)

j:MedicalAssessmentTemporaryAssessmentReviewIndicator (boolean)

j:MedicalAssessmentTemporaryIndicator (boolean)

j:MedicalAssessmentTuberculosisAlertText (TextType)

j:MedicalReport (nc:DocumentType)

j:MedicalTreatment (nc:TreatmentType)

j:Person (nc:PersonType)

j:TreatmentProvider (nc:EntityType)

j:TreatmentText (TextType)

j:MedicationUseComment (nc:CommentType)

j:MedicationUseFrequencyNumeric (NumericType)

j:CommunityLeaveDoctor (nc:PersonType)

Definition: A doctor specified as a part of a community leave.

j:EvaluationProvider (nc:EntityType)

Definition: An entity who provides a service to a Person to determine physical or mental state

j:TreatmentProvider (nc:EntityType)

Definition: An entity that provides a treatment.

j:ProviderEntity (nc:EntityType)

Definition: A person or organization providing medical services to a person.

j:PersonHospitalBirthIndicator (niem-xs:boolean)

Definition: True if a person was born in a hospital; false otherwise.

Maritime

[None Defined]

Military Operations

[None Defined]

Public Safety Canada (new)

[None Defined]

Screening (1)

scr:PersonMedicalConditionStatus (scr:PersonMedicalConditionStatusType)

Definition: A status or name of the PERSONs health problem and physical limitation that an officer should be cautious of.

scr:PersonMedicalConditionStatusCodeAbstract (abstract) This term has synonym entries

Statistics (new)

[None Defined]

Surface Transportation

[None Defined]

USAID

[None Defined]

Final Report for FHA NIEM Health

Produced by the Federal Health Architecture

Cait Ryan Cryan@irishealthsolutions.com

Brian Handspicker BHandspicker@irishealthsolutions.com

Downloads:

- Final Report for FHA NIEM Health
- NIEM Health Terminology
- NIEM Health Scenarios

- Introduction
- THE NATIONAL INFORMATION EXCHANGE MODEL
- Accomplishments
- Final Recommendations
- CLOSING REMARKS

Introduction

The following is a brief final report of the NIEM Health Community of Interest (COI) under the Federal Health Architecture (FHA) Program of the Office of the National Coordinator for Health IT (ONC) from 2012 to September 2019. The FHA will sunset September 9th, 2019 leaving an opportunity for a new steward to enhance this work supporting future of health interoperability.

The value of NIEM lies in its ability to exchange information across multiple domains and platforms.

The ability to exchange health information using NIEM enables non-clinical NIEM Domains to save lives.

THE NATIONAL INFORMATION EXCHANGE MODEL

The National Information Exchange Model (NIEM) is a framework for creating a common vocabulary by standardizing "data in motion" between disparate systems and domains. NIEM consists of 16 domains with 2 new domains – Housing Domain, Stewarded by the United States Department of Housing and Urban Development (HUD), and the Statistics Domain, stewarded by the US Census Bureau.

Although as a nation we have celebrated many accomplishments in Health information exchanges, these are focused on clinical exchanges where HL7 the predominant standard protocol. Overlooked are those in the NIEM community who require health information when responding to natural disasters, querying for beds in an ally's country for a wounded warrior, preparing an inmate for re-entry into the community, or matching patients to a prescription at a pharmacy. It is safe to assume that the majority of the world outside of a health provider setting will not have access to a FHIR server or be able to exchange C-CCDAs.

NIEM Health attempts to solve this dilemma by mapping the health information exchange requirements among 2 or more exchange partners so that the data exchanged will be understood by both the clinical health IT participants and the NIEM IT participants. This type of interoperability has the power to save lives by:

1. Integrating Social Determinants of Health with Healthcare Records;
2. Opioid Epidemic Early Intervention and Prevention;
3. Providing First Responders with the information they need to treat and where to transport a patient
4. Etc.

Accomplishments

In our time leading NIEM Health we have:

Domain	# of Health Elements
Core	38
Biometrics	2
Emergency Management	6
Human Services	75
Justice	157 + 61 Mental Health
Screening	1

1. **Surveyed NIEM Health stakeholders to understand their unique requirements.** During this time, we discovered that at a minimum, the NIEM Community needed a COI that fostered collaboration among organizations.
2. **Stood up a Community of Interest.** The COI was comprised of 30+ members from local, state and federal organizations which met on a quarterly, and then monthly basis. The COI highlight success stories, lessons learned, and milestones.

3. **Authored two documents meant to educate the NIEM user about the complexities of the health IT ecosystem** highlighting the various messaging and terminology standards as well as the importance of protecting privacy and security of protected health information.
4. **Conducted an analysis of previously modeled health related elements.** This resulted in an inventory of over 350 previously modeled and publicly defined health related elements were uncovered.
5. **Began mapping US Core Health Data for Interoperability (USCDI).** Of the 350+ health related elements uncovered; the majority have not been harmonized across NIEM domains nor do they align with HIE standards. Once mapped to USCDI.
6. **Developed a repeatable process for leveraging Health IT standards in NIEM Information Exchange Package Documents (IEPD)s.** This was done deliberately so that if NIEM Health were to be archived the NIEM community would be able to exchange health information with confidence.

Final Recommendations

1. **Existing NIEM Health elements must be aligned across domains and enhanced based on USCDI to be leveraged with existing HIE standards;**

   Health is a nationwide public matter that carries over into nearly every NIEM domain. Yet healthcare is sophisticated, involving multiple specialties and multiple health information exchange protocols. NIEM's goal is to be simple. In the absence of a mapping between these sophisticated and simple realms, NIEM Domains have, over the years, defined many simple NIEM healthcare elements on their own - over 300 NIEM Health elements defined by over six domains, These have been defined to collect critical, yet tightly focused health information to meet the requirements of non-clinical scenarios (e.g., public health, emergency management, foster care records, etc.) But these existing elements often do not align with clinical health information standards and many are not even consistently coordinated across the multiple NIEM domains that define health-related elements.

   This has happened because, while there are effective standards for clinician-to-clinician health information exchange, there is still a need for healthcare information exchange for scenarios that are just beyond the reach of the sophisticated world of health IT standards.

Historically, when NIEM Domains need to collect or exchange healthcare information, generally that information is not central to the mission of the domain and is collected from the healthcare world at the "edge" of the domain – often by reentering data scraped from printed sources. And similarly, from the healthcare IT world's perspective, information that needs to transit from the clinical health information standards world out to a NIEM Domain, is transiting at the edge of the clinical world's domain – often by printing out a record. So, from the perspective of both NIEM Domains and the Clinical Healthcare Domain enabling NIEM Health digital Information Exchange has been an edge-case that is not central to any of their missions. These edge cases are often cases of dire emergency when seconds count to save lives.

2. **NIEM Health should focus on the exchange of healthcare information in non-traditional settings or "Edge Case Scenarios"**

National Disaster Response
- Hurricane/Flood
- Earthquake/Tsunami
- Drought/Wildfire
- Dirty Bomb on the High Seas

Local Emergency Response
- Multi-Vehicle Accident with Trauma and Burns
- Building Collapse Requiring Search and Rescue

Public Health Reporting
- Ebola Infected Flight Arriving
- Vaccine Causing Widespread Allergic Reactions
- Pharmaceutical Recall

Medical Record Exchange
- Foster/Juvenile Medical Records Collection
- Wounded Warrior on Battlefield
- Reentry into the Community after Incarceration

Substance Abuse Monitoring
- Substance Abuse Rehab with Effective Supervision
- Tracking Opioid Abuse and Illegal Distribution Networks
- Law Enforcement Poisoned by Drugs During Mission

The nation has made great progress in EHR implementations. But what happens outside of these clinical to clinical transactions (hospitals, doctors offices, walk-in clinics, etc.). What happens when we involved police, firefighters and ambulances? What about triaging patients after a natural disaster? Querying for available beds when air lifting a wounded warrior from the battlefield? We coined these "Edge Case" scenarios and there is still a systematic breakdown in the ability for patient health information to be exchanged to the end user, …that is where the value of NIEM for Health is most likely.

We have identified many examples of NIEM health-related scenarios that require the ability to digitally interoperate between one or more purely NIEM Domains and the purely Clinical Healthcare Domain. Given the requirements of these many scenarios, how do we enable this interoperability between NIEM Domains and the Clinical Healthcare Domain?

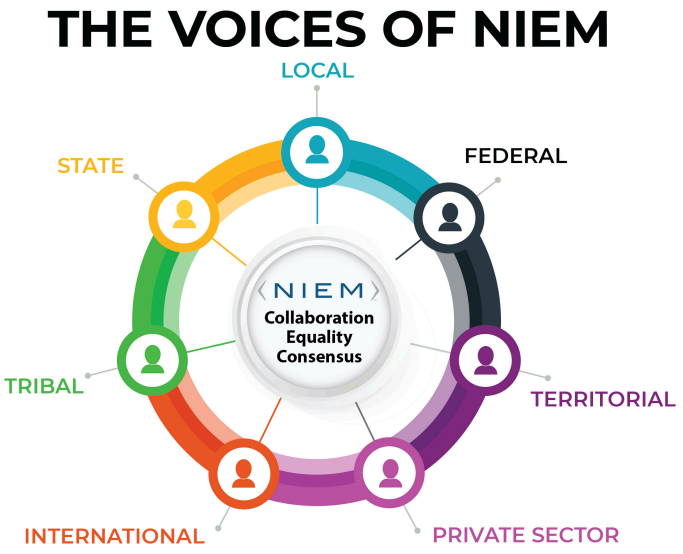
3. **Leveraging NIEM for Social Determinants of Health.**

Determinants of health are the broad range of personal, social, economic and environmental factors that determine individual and population health. NIEM is unique in that information is collected and shared for Justice, Statistics, Housing and Human Services. It is our belief that many of the social, economic and physical environment metrics for an area are likely to be drawn from databases created and supported by agencies which are required to use NIEM for interoperability. The vision here is to work with others organizations to enable automating the integration of SDOH data from these NIEM-based sources into HL7-based clinical health records.

CLOSING REMARKS

We recognize that in order to achieve nationwide health interoperability, we must look beyond the borders of provider-to-provider, clinical health information exchanges. NIEM Health, if executed correctly, has the power to save lives. It has been an honor and a privilege to have worked on this program and contribute in some small way to the improvement of our nation's health and well-being.

State, Local, Tribal, and Territorial Tiger Team



- State, Local, Tribal, and Territorial Plans
  Learn more
- State, Local, Tribal, and Territorial Educational Material
  Learn more
- State, Local, Tribal, and Territorial Success Stories
  Learn more
- State, Local, Tribal, and Territorial Action Tracking
  Learn more
- State, Local, Tribal, and Territorial Membership
  Learn more
- State, Local, Tribal, and Territorial Announcements
  Learn more
- State, Local, Tribal, and Territorial Events Calendar
  Learn more
- State, Local, Tribal, and Territorial Meeting Notes
  Learn more
- State, Local, Tribal, and Territorial Opportunities
  Opportunities
  Learn more

State, Local, Tribal, and Territorial Plans

Plans

Title	Description	Date Posted
Kickoff Plan	NIEM SLTT Tiger Team	04/01/2021

State, Local, Tribal, and Territorial Educational Material

Educational Material

This area will contain educational material.

Title	Description	Date Posted
Introduction to NIEM	Executive Briefing Presentation	03/2021
SLTT Basic Deck	Basic Deck Template for reuse by SLTT members	04/2021

State, Local, Tribal, and Territorial Success Stories

Success Stories

The following are Success Stories associated with State, Local, Tribal, and Territorial agencies/ organizations:

Title	Agency/Organization	Year
Arizona and Washington Medicaid Information	Health Policy and Justice Policy Center	2017
NIEM Drives Major Police Department's Data Integration Projects	City of Charlotte, North Carolina	2020
NIEM Ensures Accurate Statewide Crime Reporting and Analysis	State of Florida Department of Law Enforcement	2020
Hawaii Integrated Information Sharing and the Open Justice Broker Consortium	Hawaii Integrated Justice Information Sharing (HIJIS)	2012
Indiana Data Exchange - Return on Investment	Indiana Data Exchange (IDEX)	2011
Kansas Criminal Justice Information System (KCJIS): Electronic Disposition Reporting for DUIs and Traffic Safety	Kansas Bureau of Investigation (KBI)	2015
Commonwealth of Virginia's Enterprise Information Architecture & NIEM Integration Strategy	Virginia Information Technologies Agency (VITA)	2013
NIEM's Impact On Pharmaceutical Drug Monitoring	Prescription Drug Monitoring Programs (PDMPs)	2013
Governance Guidelines for Juvenile Information Sharing	National Juvenile Information Sharing Initiative (NJISI)	2012
New York State eJusticeNY Integrated Justice Portal (IJP)	New York State Division of State Police and Division of Criminal Justice Services	2011
The AMBER Alerts Information Exchange Packet Document	New York State Division of State Police and Division of Criminal Justice Services	2007
Electronic Pre-Assessment Screening Service (ePASS)	NC FAST Architecture and Infrastructure Team	2011
Automated Secure Alarm Protocol (ASAP)	New York State Police (NYSP)	2011
Prescription Monitoring Program Information Exchange (PMIX) Architecture	Prescription Monitoring Programs Technical Committee	2012
Suspicious Activity Reporting	Nationwide Suspicious Activity Reporting Initiative	2007
Criminal Justice Information Sharing Project	State of Iowa Department of Public Safety	2008
Project Interoperability in Puget Sound: A Regional Mission-Centric Perspective on Information Sharing and	Center for Collaborative Systems for Security, Safety and Regional	2016

Title	Agency/Organization	Year
Safeguarding	Resilience	

State, Local, Tribal, and Territorial Action Tracking

Action Tracker

Our action tracker:

Task	Owner	Status
Placeholder	Placeholder	Placeholder

State, Local, Tribal, and Territorial Membership

Membership

Name	Organization	Email
Paul Wormeli	Wormeli Consulting	pwormeli@wormeliconsulting.com
Michael Abramson	Canadian Public Satety & Defence	abramson@asmg-ltd.com
Christiaan Burner	Quicket Solutions	cburner@quicketsolutions.com
James Cabral	MTG Management Consultants LLC	jcabral@mtgmc.com
Kurt Cagle	Semantical, LLC	kurt.cagle@gmail.com
Tom Carlson	Tom Carlson Consulting	tom@tomcarlsonconsulting.com
Cory Casanave	OMG, Model Driven Solutions	cory-c@modeldriven.com
Chintan Dixit	Enterprise e-Support, Inc	cdixit@enterpriseesupport.com
Eric Emerson		eme9eme9@gmail.com
Dave Gajadhar	Rgroup	dgajadhar@resultantgroup.com
Juan Gonzalez	DHS/CISA	Juan.m.gonzalez@cisa.dhs.gov
Asad Hakim	KYMADVISORS	asad.hakim@kymadvisors.com
Brian Handspicker	PracticalMarkets, Inc.	bd@practicalmarkets.net
Brian Harrig	Defense Forensics and Biometrics Agency	brian.w.harrig.ctr@mail.mil
Cheryl Harrison		cherylharrison.mail@gmail.com
Dwight Hines	IndyMedia	Dwight.hines@gmail.com
Lain McNeill	DHS (S&T)/DoD (PM Biometrics)	lain.mcneill@hisdi.com
Vijay Mehra	ODNI	vijaym@dni.gov
Ada Pecos Melton	American Indian Development Associates, LLC	ada@aidainc.net
Sarah S. Pearson	Representing myself– but I work @ Tribal Tech LLC	sspearson@verizon.net
Brian K Petitt	DHS/CWMD/SSD	brian.petitt@associates.hq.dhs.gov
Michael Phillips	SLG Innovation	michaelphillips@slginnovation.com
Mark Pritchard	AAMVA	mpritchard@aamva.org
Cait Ryan	None	cryan@irishealthsolutions.com
Jitendra	Sakhamuri Wiztech Inc	Jitendra@wiztechinc.com
Linda Self	JS J6 DSD	linda.l.self3.ctr@mail.mil
Kate Silhol		ksilhol@nlets.org
Andrew Sloan	DHS/OCIO	andrew.sloan@hq.dhs.gov
Steve Sullivan	JS J6 DSD	stephen.m.sullivan14.ctr@mail.mil
Christopher Tarantino	Epicenter Innovation	ctarantino@epimetra.com
Tammara Tarvin		ttarvin@idahosheriffs.org

State, Local, Tribal, and Territorial Charter

Download Charter

Charter Date

- 04/19/2021

Sponsor

- NIEM Business Architecture Committee (NBAC)

Stakeholders

- U.S. and International State, Local, Tribal, and Territorial information sharing partners

Co-Chairs

- Mike Phillips, SLG Innovation
- Paul Wormeli, Wormeli Consulting

Project Purpose

- To advance information sharing at the state, local, and tribal level to help protect, support, and respond to community needs to improve the quality of life for all

Project Description

- The NBAC State, Local and Tribal Tiger Team was created to improve understanding of the NIEM value proposition, support adoption, and develop advocacy champions

Scope Statement

- Information sharing collaboration, planning, development, and demonstration activities that are prioritized and can be accomplished by the SLTT membership

Goals and Objectives

1. Obtain Executive-level SLTT leader advocacy champions
2. Obtain Program Manager-level SLTT leader advocacy champions
3. Obtain State CIOs/CDOs as advocacy champions
4. Introduce the NIEM Value Proposition at State, Local, Tribal and Territorial association symposiums, conferences and seminars
5. Include conference participation as part of the domain capability framework

6. Add SLTT representation to the Speakers Bureau and provide materials for presenters

Deliverables

1. Support materials for Tiger Team and others to use in presenting the NIEM framework
2. Tiger Team Charter
3. Tiger Team Web Page on niem.github.io to host project artifacts
4. Mentoring Program (New to NIEM and Grant Assistance)
5. Monthly NBAC SLTT Updates

Resources

Required

1. Two Co-Chairs to lead Tiger Team activities and delegate responsibilities

Requested

1. At least one representative from every NIEM Domain 2.At least one representative from every NIEM Community of Interest

Project Approach

- Collaboration with SLTT members to develop an engagement process to increase association support
    - Host Tiger Team Meetings
    - Work with NBAC and the NIEM Management Office to coordinate outreach to SLTT organizations and communities
    - Review and provide advice on updating existing NIEM Marketing Materials
        - Videos, Slides, Fact Sheets, Success Stories
    - Develop Communications and Marketing Support
        - Develop materials for use in presenting NIEM
        - Create interactive channels to connect SLTT stakeholders to NIEM
        - Hold Educational/Informational Webinars
    - Follow up with interested parties and set up one on one meetings
        - Anyone who has emailed with questions
        - Anyone who expresses interest in learning more about NIEM
- Leverage National, State, Local, Tribal & Territorial association meetings, sessions, conferences, and seminars, to disseminate NIEM information
    - Identify appropriate events to attend/participate/speak
    - Identify Speaker (if applicable)
    - Develop Marketing Materials
        - Presentation Slides (if applicable)
        - Leave Behinds
            - Fact Sheet
            - Brochure
        - Marketing Videos

State, Local, Tribal, and Territorial Announcements

Announcements

- NIEM Announces New SLTT Tiger Team Leadership
  03-04-2021
  Learn more
- NIEM Announces New Major Release
  03-30-2021
  Learn more

NIEM Announces New SLTT Tiger Team Leadership

NIEM Announces new State, Local, Tribal, and Territorial Tiger Team Leadership

Thursday, March 4, 2021

The National Information Exchange Model (NIEM) is pleased to announce newly appointed co-chairs to the NIEM's State, Local, Tribal, and Territorial (SLTT) Tiger Team. Incoming co-chairs are Paul Wormeli and Michael Phillips. The co-chairs are focusing on how NIEM can best support the highest priorities of each SLTT segments.

"NIEM is exceptionally fortunate to have these two long-time, highly experienced NIEM advocates to chair NIEM's SLTT Tiger Team. It was established with the strong support and considerable interest of our NIEM community. We look forward to the continued growth and involvement of the team under our co-chairs' leadership," stated Katherine Escobar, NIEM's Managing Director. The team was chartered in 2020 with the leadership of co-chairs Cait Ryan and Tom Carlson. During their tenure, NIEM hosted a series of well attended virtual meetings gaining both interest and membership across NIEM's domains and the community at large.

The SLTT Tiger Team's purpose is to advance information-sharing at the state, local, tribal, and territorial jurisdictions to protect, support, and respond to community needs. Its mission is to increase NIEM's use in facilitating cross-domain information exchanges for these agencies and organizations to improve the safety and quality of life for our citizens.

Paul Wormeli is an Information Strategist with Wormeli Consulting, LLC. He is an innovator and entrepreneur who has focused on the application of information technology to public safety, law enforcement, criminal justice, and homeland security. Wormeli has been active with NIEM since its inception in 2005, including serving as the founding chair of the communication and outreach committee.

Michael Phillips is Vice President–Integrated Solutions with SLG Innovation. He has extensive experience working with state, local, tribal, federal, and international governments and agencies. Phillips was the project manager for one of the largest state-level NIEM adoptions implementing data exchanges between over 200 law enforcement agencies to ensure the sharing of warrant information, suspicious activities, arrests, and incidents in answer to the 9/11 attack. To learn more about the State, Local, Tribal, and Territorial (SLTT) Tiger Team, please contact NIEM.

NIEM Announces New Major Release

NIEM Announces New Major Release

Tuesday, March 30, 2021

The National Information Exchange Model (NIEM) has released NIEM 5.0. This major release makes it easier to use NIEM, enabling interoperability across diverse and complex systems and organizations. The model is streamlined and simplified and now contains a Universal Modelling Language (UML) capability for easier development of message specifications.

NIEM 5.0 increases NIEM's flexibility, establishing the framework for the NIEM metamodel currently in development. As planned, the metamodel will allow the development of NIEM conformant message specifications to not be limited to the XML data format. NIEM JSON's Specification v5.0 is anticipated to be released in May, establishing the technical basis for using JavaScript Object Notation (JSON) as a data format. NIEM's TechHub provides the developer community with resources and training.

Technical repositories are simplified for ease of navigation and use. The landing page for NIEM Specifications is located at https://niem.github.io/reference/specificati ons/.

The release includes expanded content from NIEM domains and communities of interest, including Cyber, NIEM's newest domain, and the planned Statistics domain. Major updates were made to the Federal Bureau of Investigation's National Crime Information Center National Data Exchange (NDEx) and Uniform Crime Reporting (and NIBRS) code tables, as well as the Public Health Emergency Operations Center's Minimum Data Set associated with the NIEM Emergency Management Domain. New and reusable message specifications have been developed, such as the Department of Transportation's model for crash reporting.

"NIEM 5.0 represents three years of extensive NIEM user-community input, development, and harmonization efforts to deliver information exchange solutions addressing our users' needs," said Katherine Escobar, Managing Director, NIEM Management Office. She continued, "With NIEM's community-powered, open- source approach, users directly contribute both content and technical expertise, propelling NIEM's growth forward with timely solutions to challenging interoperability problems."

NIEM's Core contains the data components within information exchange elements which are universally available to all NIEM users. New and updated specifications better describe what data look like and mean. NIEM's strength of extensibility allows individual domains and agencies to further define data elements specific to their unique needs and requirements, in accordance with naming and design rules that support interoperability standards.

NIEM currently supports exchange efforts among fifteen chartered domains and communities of interest working towards becoming a domain. State and local agencies are particularly served by the domains of biometrics, cyber, justice, emergency management, human services, family services, and the planned domains associated with statistics and transportation. Participants in NIEM are representatives from federal, state, local, tribal, territorial, international, and the private sector. Overall, the NIEM model defines more than 11,000 existing elements, or properties, that define the semantic content of the model.

Key changes to NIEM accomplished in version 5.0 can be viewed at https://release.niem.gov/niem/5.0/README.html To review the NIEM 5.0 release package , visit release.niem.gov. Online tools are available to search and view NIEM release content. To learn more about NIEM and how to participate in the State, Local, Tribal, and Territorial (SLTT) Tiger Team, please contact NIEM. About the National Information Exchange Model

The National Information Exchange Model (NIEM) provides a model framework with rules and guidance designed to develop consistent, well-defined, enterprise level information exchanges. For sixteen years, NIEM has successfully supported information communities of interest with a shared need to exchange data within the federal, state, local, tribal, and territorial agencies and organizations, the private sector, and internationally. To learn more, please visit www.NIEM.gov.

#####

State, Local, Tribal, and Territorial Events Calendar

Meetings & Events Calendar

Date	Venue	Topic	Domain	Speakers
7/13/21	SEARCH Symposium	Advances in NIEM	Justice	K. Escobar, P. Wormeli, M. Phillips
7/15/21	IJIS Mid-year Briefing	Advances in NIEM	Multiple	K. Escobar, P. Wormeli, M. Phillips
9/15/21	NBAC National Meeting	NIEM and Child Care	Human Services	Jeremy Howard
9/15/21	NBAC National Meeting	SLTT Update	All	P. Wormeli, M. Phillips
10/21/21	NLETS Webinar	NIEM Applications	Justice and Emergency Management	Kate Silhol, K. Escobar, P. Wormeli, M. Phillips

State, Local, Tribal, and Territorial Meeting Notes

SLTT Meeting Notes

The State, Local, Tribal, and Territorial Tiger Team has regular meetings, typically via teleconference. Notes are archived here.

Date	Meeting Description
2020-06-10	Kickoff Call

State, Local, Tribal, and Territorial Opportunities

SLTT Opportunities

The State, Local, Tribal, and Territorial Tiger Team periodically checks grants.gov for state, local, and tribal NIEM-based opportunities:

Opportunity Number	Title	Closing Date
DHS-20-CISA-128-NIEM001	State, Local, Tribal and Territorial National Information Exchange Model Cyber Pilot	07/27/2020

Geospatial Integration

There is a growing need for information exchanges to include geospatial elements and capabilities—NIEM provides this functionality.

Using GML with NIEM

The role and utility of geospatial technology continues to increase along with the amount of information containing geotags (address, latitude/longitude, location). To meet this growing demand, NIEM developed geospatial exchange capabilities that leverage Geography Markup Language (GML) standards established by the Open Geospatial Consortium (OGC). By combining NIEM with GML, mission partners benefit from shared access to the common operating data and services used within these geospatial systems. NIEM is working with members of the community who have leveraged GML in NIEM exchange development. Please contact us if you have examples of how you're using NIEM and GML.

Geo4NIEM Initiative

The Geospatial Enhancement for NIEM (Geo4NIEM) initiative enhances NIEM's geospatial exchange capabilities by demonstrating and providing enterprise NIEM-conformant content in a map visualization context that:

- Provided the appropriate security markings and access controls to ensure the right people have access to the right information at the right time in the right context.
- Can be reused across the public and private sector.
- Promoted cost-savings (through use of NIEM and OGC community open standards).

Geo4NIEM Efforts

There have been two major Geo4NIEM efforts:

- Geo4NIEM Part 1 (2013)

  Geo4NIEM Part 1 (2013) was a collaborative, hands-on rapid prototype development and testing initiative in accordance with the Open Geospatial Consortium (OGC)'s Interoperability Program. The partnership included NGA, Department of Defense (DoD), Defense Information Systems Agency (DISA), U.S. Geological Survey (USGS), the Army Geospatial Center and numerous participants and supporters from the public and private sector.

- Geo4NIEM Part 2 (2015)

  Geo4NIEM Part 2 (2015) was one of four threads in OGC's Testbed 11—a scenario-based test and demonstration of Geo4NIEM work. It focused on enhancing NIEM's geospatial exchange capabilities to include Intelligence Community (IC) data encoding specifications, along with OASIS standards to enable granular data object

level access authorization and/or denial aligned to OCG web services.

Geo4NIEM Part 1 (2013)

Geo4NIEM Part 1 (2013) was a collaborative, hands-on rapid prototype development and testing initiative in accordance with the Open Geospatial Consortium (OGC)'s Interoperability Program. The partnership included NGA, Department of Defense (DoD), Defense Information Systems Agency (DISA), U.S. Geological Survey (USGS), the Army Geospatial Center and numerous participants and supporters from the public and private sector.

Objectives

The Geo4NIEM Part 1 pilot met the following objectives:

- Provided the appropriate security markings and access controls to ensure the right people have access to the right information at the right time in the right context.
- Can be reused across the public and private sector.
- Promoted cost-savings (through use of NIEM and OGC community open standards).

Efforts

There have been two major Geo4NIEM efforts:

- Developed recommendations for the inclusion and standard use of embedded geography markup language (GML) with NIEM Information Exchange Package Documentations (IEPDs).
- Developed recommendations for the standardized use of Naming and Design Rules (NDRs) and the use of adaptors (e.g., NIEM wrapper for GML).
- Tested and demonstrated use of a standardized embedded GML and adaptors within NIEM IEPDs.
- Developed architecture documentation and "Fact Sheet" for the use of embedded GML and adaptors for use with NIEM IEPDs.

Findings

Developed architecture documentation and "Fact Sheet" for the use of embedded GML and adaptors for use with NIEM IEPDs.

The Geo4NIEM Part 1 initiative affirmed that NIEM information exchange packages (IEPs) can include GML-based data components, and that GML documents can include NIEM-based data components. It culminated in a demonstration of how enhanced geospatial capabilities enable situational awareness through the ability to identify, process, and comprehend critical information during an incident.

Recommendations

The Geo4NIEM initiative provided eight recommendations (specific to GML adapters in NIEM) for consideration in the NIEM 3.0 update. Seven recommendations were implemented; one recommendation had no impact on the model itself. Overall, no significant changes were recommended

to the NIEM technical architecture. An engineering report summarizing the findings and recommendations of the Geo4NIEM initiative was published in November of 2013. Download the engineering report to read more about the findings.

NIEM Request of Information Exchange (RFI)

The NIEM Request for Information (RFI) exchange facilitates interagency processes involving requirements, tasking, and information gathering activities that support data collection and awareness within intelligence, public safety, emergency management, and response operations. A NIEM RFI exchange document provides the mechanism through which intelligence analysts and operations managers can request and receive relevant information for processing and analysis.

MDA – Maritime Domain Awareness (Vessel Position/Track)

Maritime Domain Awareness is an inter-agency and international effort to detect and prevent threats at sea or in any navigable waterway. The NIEM Vessel Position exchange from the Maritime Domain supports the effective understanding of anything associated with global maritime that could impact the United States' security, safety, economy, or environment. NIEM facilitates this understanding through effective, timely sharing of vital, secure information among many key partners by representing vessels, people, cargo, and maritime locations and activities.

Geo4NIEM Part 2 (2015)

Geo4NIEM Part 2 (2015) was one of four threads in OGC's Testbed 11—a scenario-based test and demonstration of Geo4NIEM work. It focused on enhancing NIEM's geospatial exchange capabilities to include Intelligence Community (IC) data encoding specifications, along with OASIS standards to enable granular data object level access authorization and/or denial aligned to OCG web services.

Objectives

The Geo4NIEM Part 2 testbed met the following objectives:

- Assessed support in NIEM for IC security specifications (ISM, NTK, TDF) and recommended a security architecture.
- Demonstrated secure information exchange using architecture from Task 1.
- Assessed NIEM and GML support for geospatial data exchange from NIEM-based client to GML-based client and back (round-trip); recommended a round-trip architecture.
- Demonstrated round-trip geospatial data exchange using the architecture from Task 3.
- Demonstrated OGC Web Feature Service (WFS) on GML feature representations with embedded NIEM components.
- Reached consistent security approach across the OGC suite of service standards.

Why is it needed?

Geospatial information technologies are increasingly a foundation for supporting emergency management, public safety, and other missions. Because many of these communities are already using NIEM, efforts to enhance NIEM's geospatial exchange capabilities will help significantly improve inter-government information sharing of this critical data source.

What's next?

The following recommended focus areas are under consideration for potential future Geo4NIEM work:

- Tools and best practices for exposing NIEM Information Exchange Package Documentation (IEPD) through the OGC Web Feature Service (WFS) interface, including making IEPD schemas available via WFS.
- Tools and best practices for expressing and enforcing access control policy in terms of IC security metadata.
- Best practices for simplified IEPDs (i.e., location and time core elements in GML Simple Features profile; model similar to Cursor-on-Target (CoT)).

Findings

The Geo4NIEM Part 2 Testbed affirmed that, with reasonable effort, it is possible to combine NIEM, IC security specifications, OGC Web Service components and GML-aware clients to support information exchange with authorized users. This is a huge step forward toward enabling first responders, law

enforcement, emergency management, military support and the intelligence community to collaborate real time without compromising access controls. Access control engines can work with NIEM/IC data encoding, with or without a services framework. Findings were shared during the OGC meetings held during June 2015.

Testbed 11 Engineering Reports summarizing Geo4NIEM Part 2

Testbed 11 Geo4NIEM Architecture Design and Implementation Guidance and Fact Sheet

Reference Number: OGC 15-051r3

This document describes background considerations, as well as an overview of the services, data encodings and access control frameworks that compose the Geo4NIEM Testbed 11 architecture.

https://portal.opengeospatial.org/files/?artifact_id=65419

Testbed 11 NIEM & IC Data Encoding Specification Assessment and Recommendations ER

Reference Number: OGC 15-048r3

This document is a product of a Task 1, which assessed the potential for security tagging and access control from IC Data Encoding Specifications to be leveraged with NIEM in support of information exchange. The purpose was to determine if the current architecture of NIEM can support IC specification alignment. The IC Data Encoding Specifications include, but are not limited to, ISM, NTK and TDF metadata.

https://portal.opengeospatial.org/files/?artifact_id=65420

Testbed 11 Results of Test and Demonstration of NIEM Using IC Data Encoding Specifications ER

Reference Number: OGC 15-050r3

This document is a product of Task 2, which used preliminary findings and recommended architectures for IC Data Encoding Specification support identified in Task 1, and performed a Test and Demonstration of the recommended architecture leveraging the results of Testbed 9 and previous Geo4NIEM initiatives where appropriate. Results of this Test and Demonstration were documented in an Engineering Repot containing the Findings and Recommendations with reference to refinements to the originally proposed architecture prepared in Task 1.

https://portal.opengeospatial.org/files/?artifact_id=65421

Testbed 11 NIEM-GML-NIEM Round-Trip Assessment and Recommendations ER

Reference Number: OGC 15-030r3

This document is a product of Task 3, which assessed the NIEM and GML support for geospatial data exchange round-trip workflow processes to include: creation, transfer, receipt, modification, return and acceptance of XML content originating as NIEM IEPDs; and Task 4, which covered the testing and demonstration of the recommended architecture structure supporting NIEM-GML-NIEM round-trip assessment identification.

https://portal.opengeospatial.org/files/?artifact_id=65451

Testbed 11 NIEM-IC Feature Processing API Using OGC Web Services ER

Reference Number: OGC 15-047r3

This document is a product of Task 5, which performed Test and Demonstrations using OGC web services, such as Basic and Transactional Web Feature Service (WFS-T) and Policy Enforcement Points (PEPs), to process GML feature representations leveraging NIEM components. The Test and Demonstrations included, but are not limited to feature, retrieval, insert, update and delete.

https://portal.opengeospatial.org/files/?artifact_id=65418