Master NIEM Training Document

NIEM Training
2022 January 11-13

Tom Carlson
tom@tomcarlsonconsulting.com
GTRI / NIEM Consultant

Introduction

Purpose

A technical webinar focused on developers and implementers who are interested in a deep-dive of the National Information Exchange Model.

Supporting Documents

All materials are available on the NIEM Training Github repo at https://github.com/NIEM/NIEM-Training. Specific materials used are listed on the README page and include:

- NIEM Training Syllabus
- **Master NIEM Training Document**
- Mapping Spreadsheets
- Ersatz Textual Instances

Sample IEPD

- Crash Driver IEPD

Agenda

- Logistics
- Introduction to NIEM
- Introduction to IEPD Development
  - Scenario Planning
  - Requirements Analysis
  - Mapping
    - Intro to Mapping
    - Mapping to Existing Objects
    - Creating New Objects
    - External Standards
  - Creating and Validating Schemas
  - Assembly
  - Publishing
  - Implementation
- Exercises
- Resources

Logistics and Background

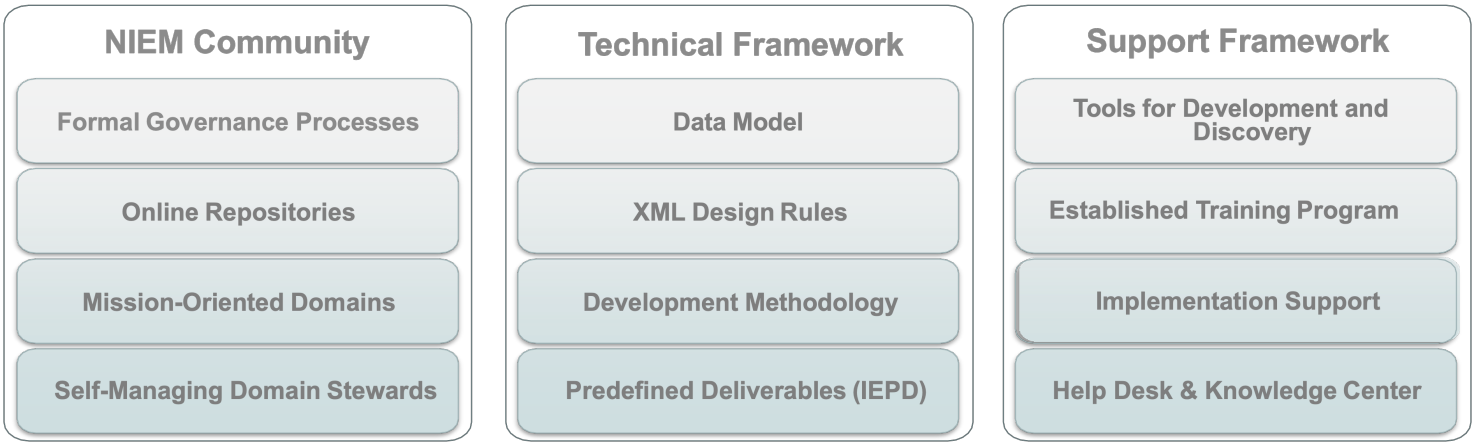
- Second live session
  - Major revamp of prior training programs
  - Materials revamped from first live session
- Three days, 1-5pm each day
- Short breaks on the hour
- Ask questions via chat *when you have them*
  - Someone is monitoring chat and will interrupt me as needed
- This document and supporting materials at:
  - https://github.com/niem/niem-training

Introduction to NIEM

- What is NIEM?
- The Scope of NIEM
- NIEM Harmonization and Organization

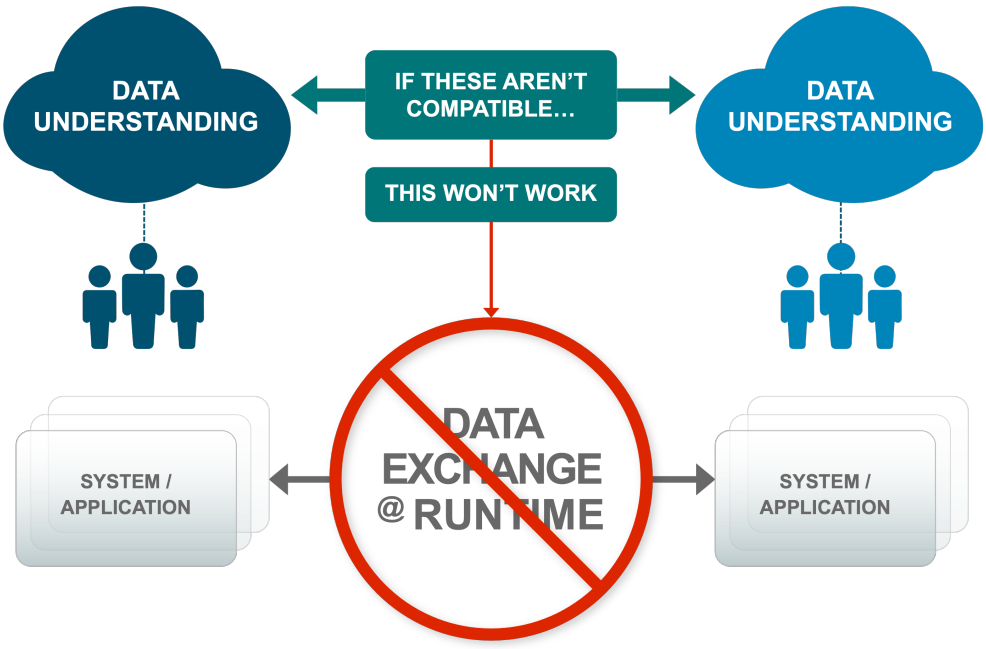
What is NIEM? - Framework

- NIEM is a community-driven, government and jurisdiction-wide, standards-based approach to exchanging information
- Diverse communities can collectively leverage NIEM to increase efficiencies and improve decision-making
- NIEM is available to everyone, including public and private organizations
- NIEM includes a data model, governance, training, tools, technical support services, and an active community to assist users in adopting a standards-based approach to exchanging data



What is NIEM? – Interop Problem

- If we don't understand what each other means, we won't be able to exchange info
- Need a common language for defining things



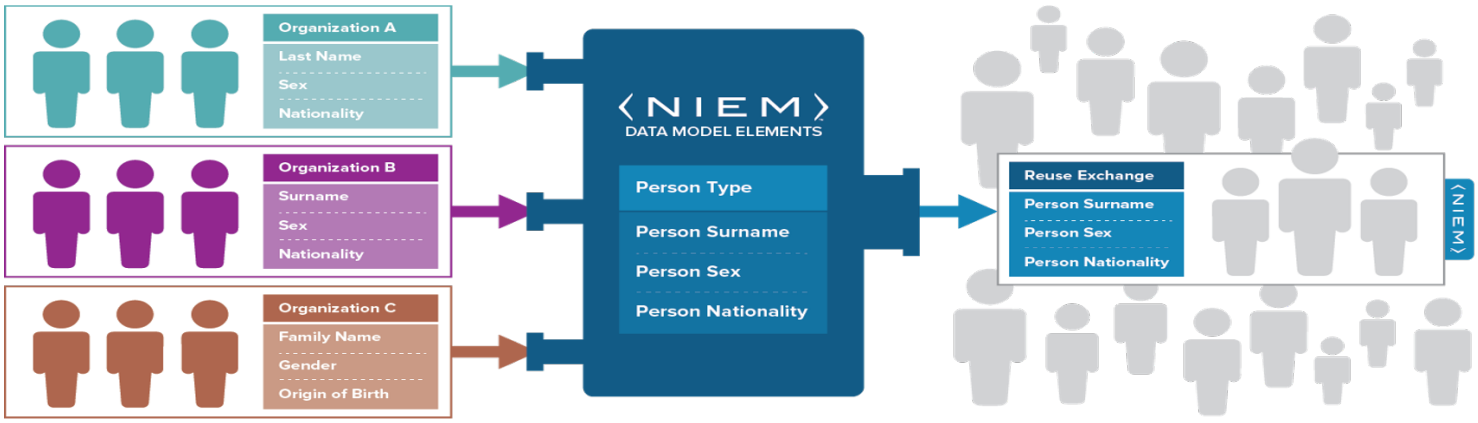
What is NIEM? And what not?

NIEM is:
- a common vocabulary
- a means of enabling efficient information exchange across diverse public and private - organizations

NIEM is not:
- a system or database
- a means of specifying how to transmit or store data

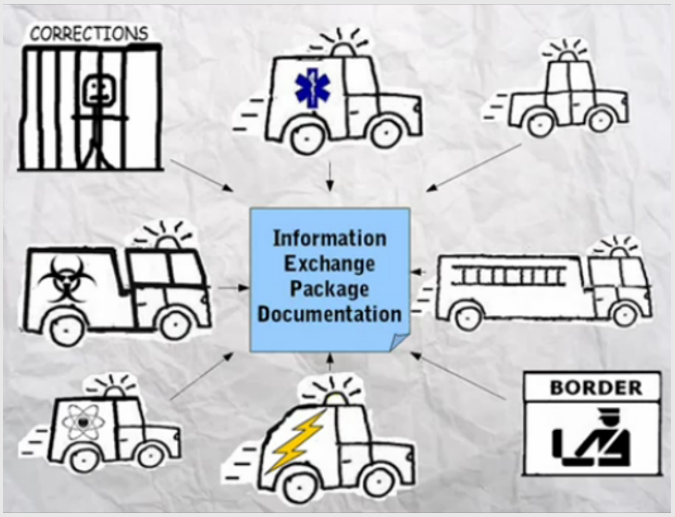
What is NIEM? – Exchanging Data and Components

- Using NIEM, organizations come together to agree on a common vocabulary
- When additional organizations are added to the information exchange, the initial NIEM exchange can be reused, saving time and money



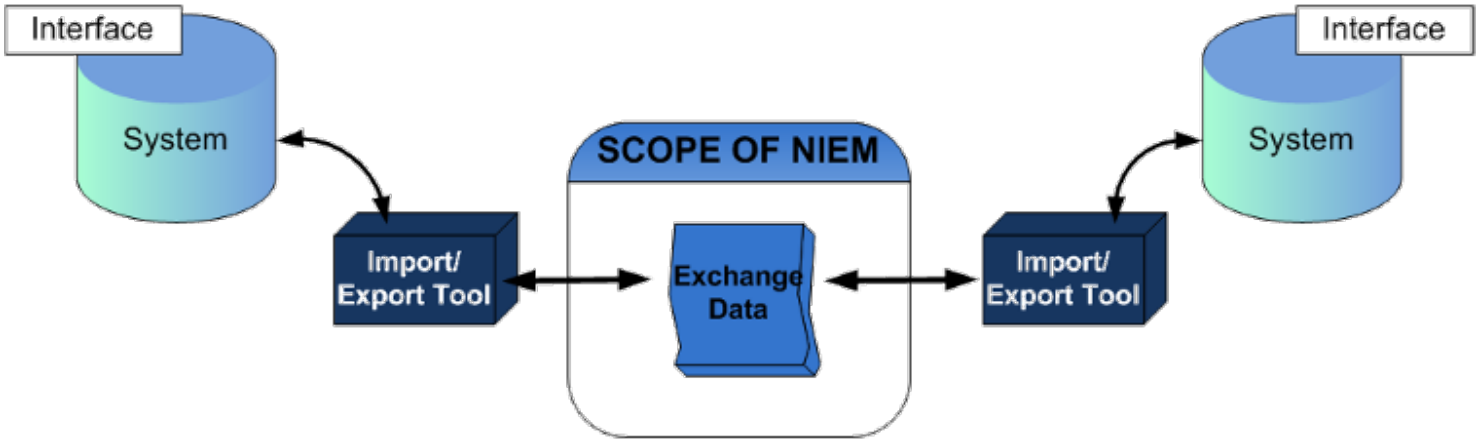
What is NIEM? – Simplified Exchanges

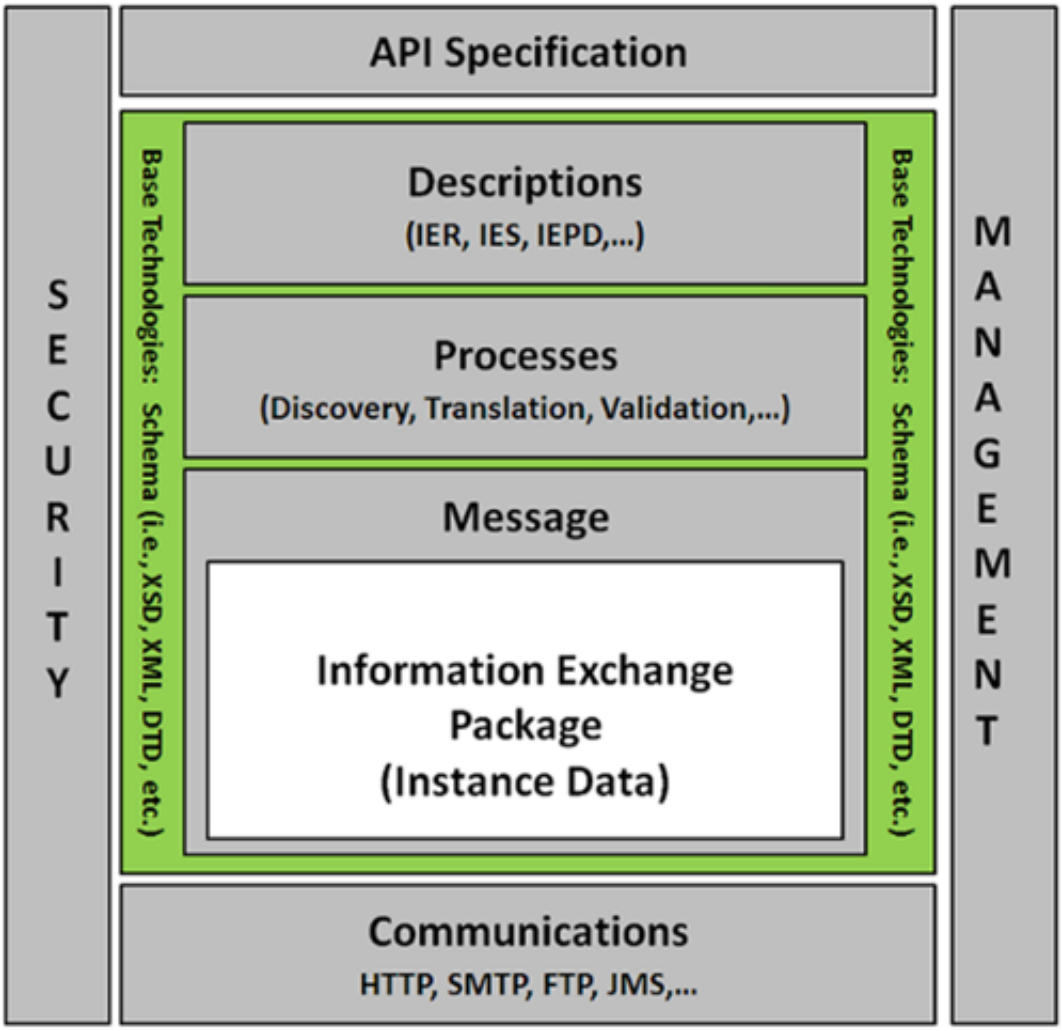
- When using NIEM, you only need to “speak” two languages: your own and NIEM

Without NIEM	With NIEM
	

Scope of NIEM

- NIEM is a data layer standard and intentionally does not address all the necessary technologies needed for information sharing
- Exchange partners decide how to store and process the NIEM-conformant data being exchanged





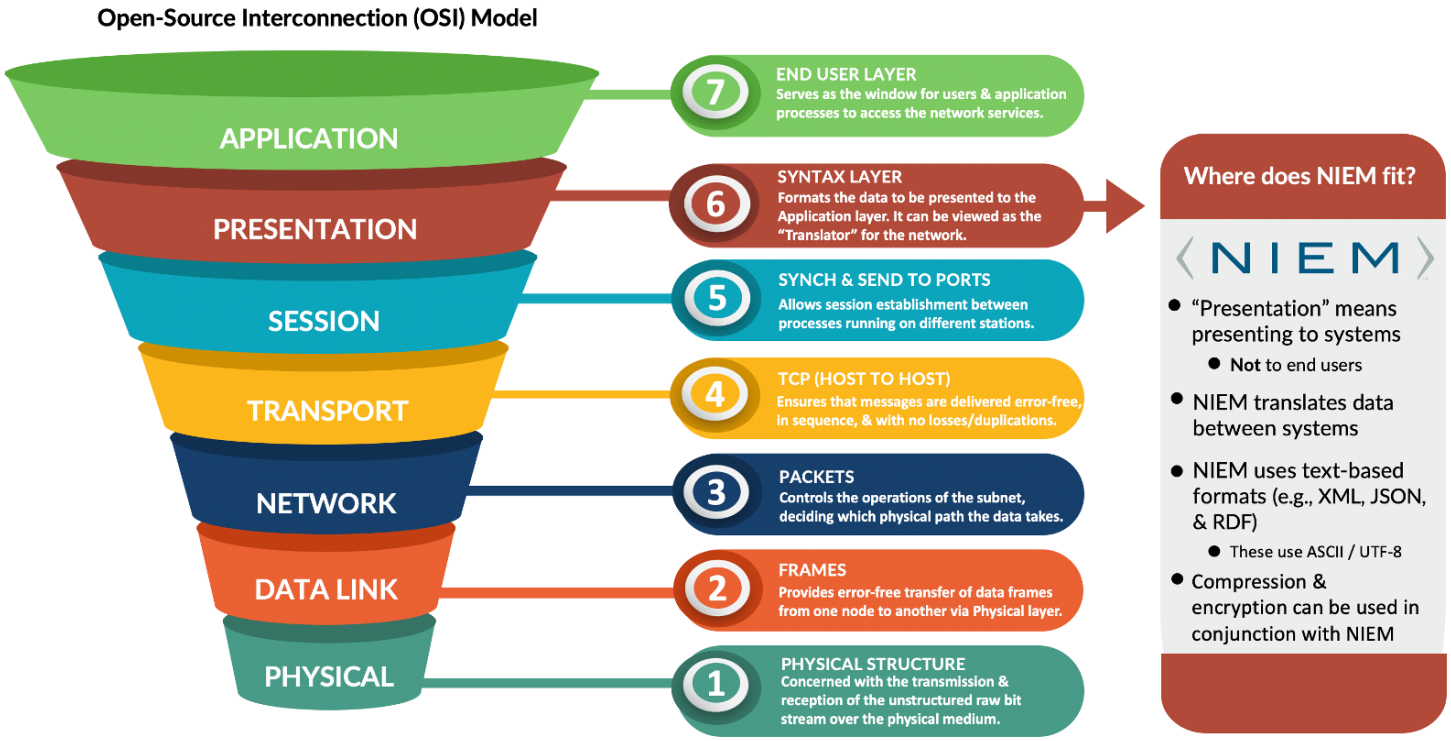
Scope of NIEM - Open Source Interconnection (OSI) model

Detailed:

OSI (Open Source Interconnection) 7 Layer Model

Layer	Application/Example		Central Device/ Protocols		DOD4 Model
Application (7) Serves as the window for users and application processes to access the network services.	**End User layer** Program that opens what was sent or creates what is to be sent Resource sharing • Remote file access • Remote printer access • Directory services • Network management		**User Applications** SMTP	GATEWAY Can be used on all layers	Process
Presentation (6) Formats the data to be presented to the Application layer. It can be viewed as the "Translator" for the network.	**Syntax layer** encrypt & decrypt (if needed) Character code translation • Data conversion • Data compression • Data encryption • **Character Set Translation**		JPEG/ASCII EBDIC/TIFF/GIF PICT		
Session (5) Allows session establishment between processes running on different stations.	**Synch & send to ports** (logical ports) Session establishment, maintenance and termination • Session support - perform security, name recognition, logging, etc.		**Logical Ports** RPC/SQL/NFS NetBIOS names		
Transport (4) Ensures that messages are delivered error-free, in sequence, and with no losses or duplications.	**TCP** Host to Host, Flow Control Message segmentation • Message acknowledgement • Message traffic control • Session multiplexing	PACKET FILTERING	TCP/SPX/UDP		Host to Host
Network (3) Controls the operations of the subnet, deciding which physical path the data takes.	**Packets** ("letter", contains IP address) Routing • Subnet traffic control • Frame fragmentation • Logical-physical address mapping • Subnet usage accounting		**Routers** IP/IPX/ICMP		Internet
Data Link (2) Provides error-free transfer of data frames from one node to another over the Physical layer.	**Frames** ("envelopes", contains MAC address) [NIC card — Switch — NIC card] (end to end) Establishes & terminates the logical link between nodes • Frame traffic control • Frame sequencing • Frame acknowledgment • Frame delimiting • Frame error checking • Media access control		**Switch** **Bridge** **WAP** PPP/SLIP	Land Based Layers	Network
Physical (1) Concerned with the transmission and reception of the unstructured raw bit stream over the physical medium.	**Physical structure** Cables, hubs, etc. Data Encoding • Physical medium attachment • Transmission technique - Baseband or Broadband • Physical medium transmission Bits & Volts		**Hub**		

Simplified:



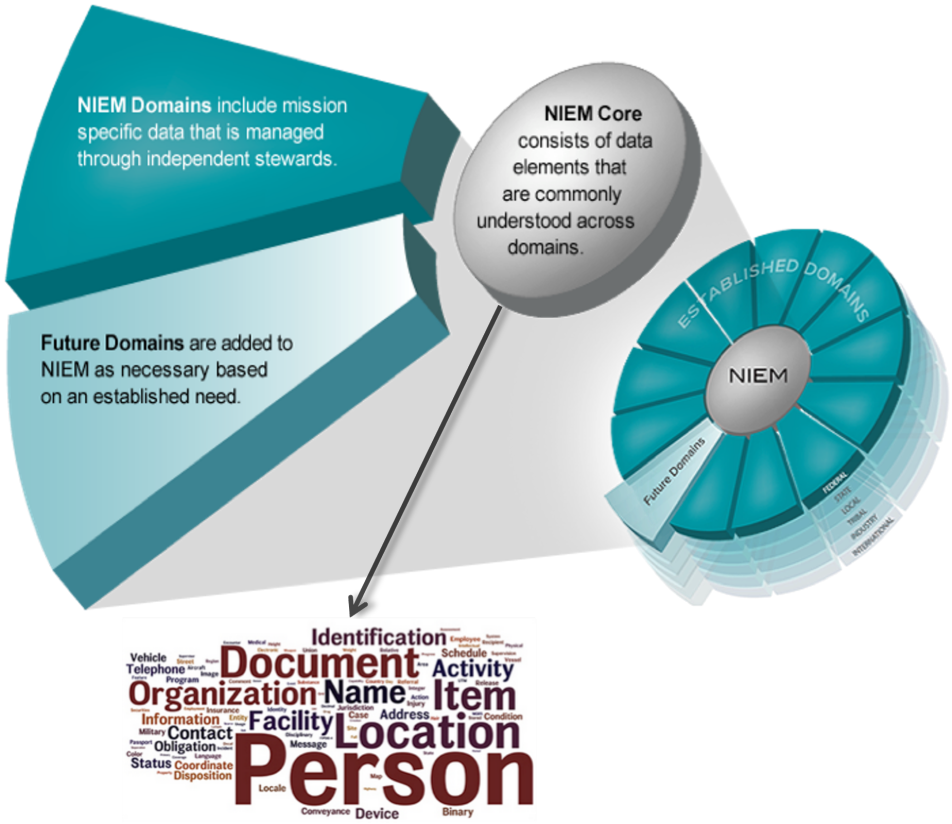
Bottle of Liquid

A bottle of liquid:

	
Has some combination of liquids inside	NIEM is the liquid inside, the payload document
Has a shape	NIEM doesn’t care about the shape
Made of certain materials	NIEM doesn’t care about what the bottle is made out of or how it’s constructed
Can be opaque or transparent	NIEM doesn’t care about whether you can see into the bottle
Is moved around by various means	NIEM doesn’t care about how you move the bottle around
Can be filled and emptied	NIEM doesn’t care about how you filled it or what you do with the liquid later

NIEM Harmonization and Organization

- Think of the NIEM data model as a mature and stable data dictionary of agreed-upon terms, definitions, relationships and formats independent of how information is stored in individual agency systems
- The data model consists of two sets of closely related vocabularies:
  - NIEM core
  - Individual NIEM domains
- NIEM core includes data elements commonly agreed upon across all NIEM domains (i.e., person, activity, location, and item, etc.)
- Individual NIEM domains contain mission-specific data components that build upon NIEM core concepts



Existing Domains	Upcoming Domains
Agriculture	Learning and Development
Biometrics	International Human Services
Chemical, Biological, Radiological, & Nuclear	
Cyber (new in 5.1)	
Emergency Management	
Human Services	
Immigration	
Infrastructure Protection	
Intelligence	
International Trade	
Justice	
Maritime	
Military Operations	
Screening	
Surface Transportation	

Domains hold objects specific to their domains:

Agriculture	Biometrics
	

Chemical, Biological, Radiological, & Nuclear

Emergency Management

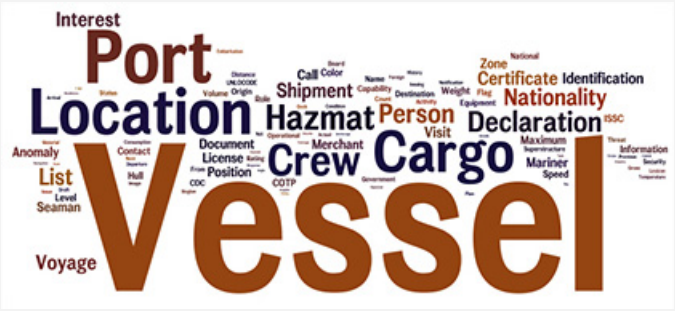
Human Services

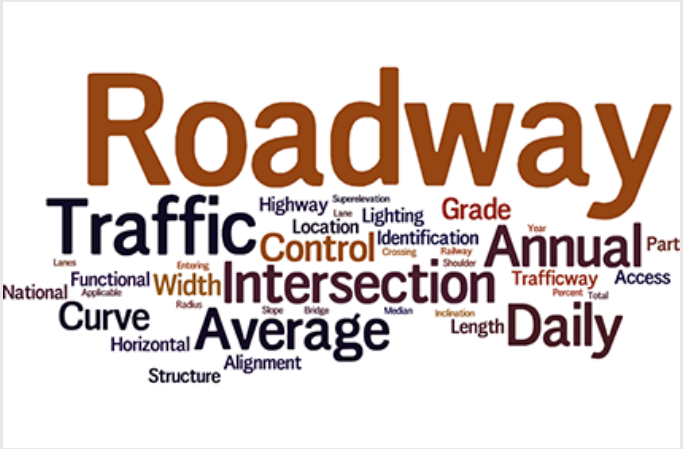
Immigration

Infrastructure Protection

Intelligence

International Trade	Justice
	

Maritime	Military Operations
	

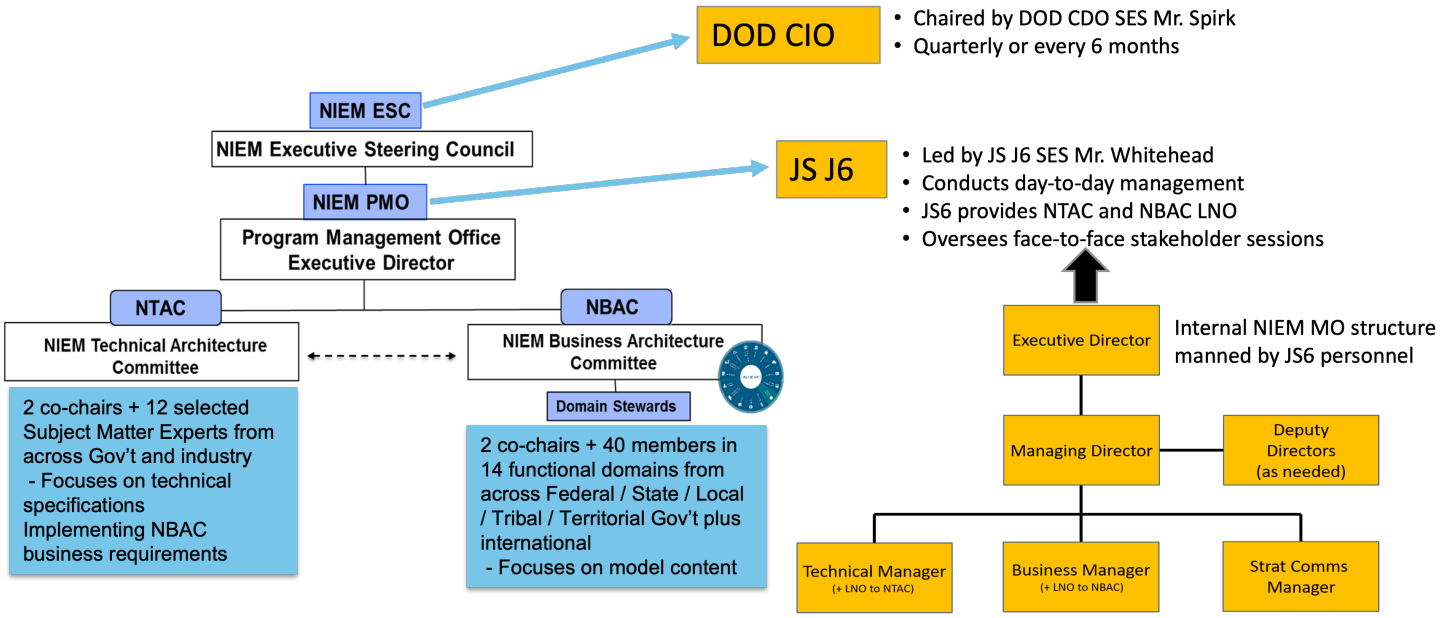
Screening	Surface Transportation
	

NIEM Versioning

- NIEM has major and minor versions, plus domain updates
- Major version releases, e.g. 4.2 to 5.0
  - Every 3 years
  - All bets are off
  - NIEM-Core can and will change

        - Underlying infrastructure can also change
    - Domains can change
    - Domains are harmonized
        - Combination of tools and human collaboration to reach consensus
        - Repeated content is collapsed
        - Misplaced content is moved, either to other domains or to core
        - New content is added
    - *Nothing* in a major version change is guaranteed to be backwards compatible with earlier major releases
- Minor version releases:
    - Annually
    - **NIEM-Core does not change!**
        - **Neither does the underlying infrastructure**
    - Domains can change
    - Domains can be harmonized
    - Domains can be added
    - Domains are not guaranteed to be backwards compatible with earlier minor releases
        - But they often are
- Domain updates are done per-domain
    - Domains can update their content in between minor releases
    - Those updates then are normally folded into the next minor release
- Older versions never go away
    - You can still use NIEM 1.0 (but shouldn't)
- Migration
    - Don't have to migrate
    - May want to migrate if a newer version gives you functionality you need (and you're already making changes)
    - NIEM provides tools for migrating the NIEM objects, ~90% effective
    - Manual work is needed for things you've added for your exchange

NIEM Administration Organization



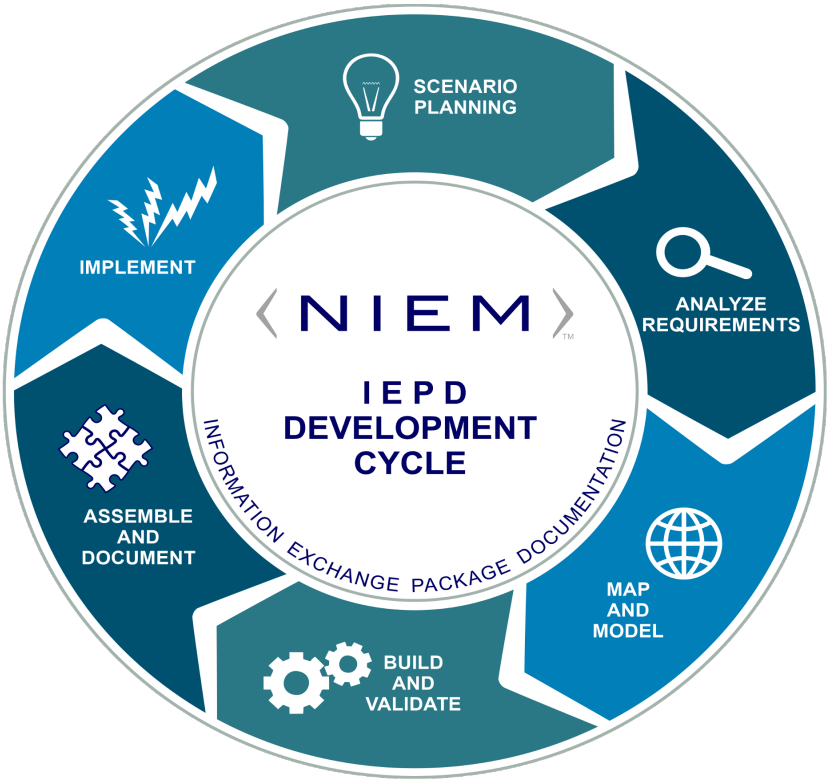
Message Spec / IEPD Overview

- "Information Exchange Package Documentation"
- Slowly changing to "Message Specification"
- Defines an exchange
- Made up of a bunch of documents, "artifacts"
    - Some meant for humans
    - Some meant for computers

Message Spec Process

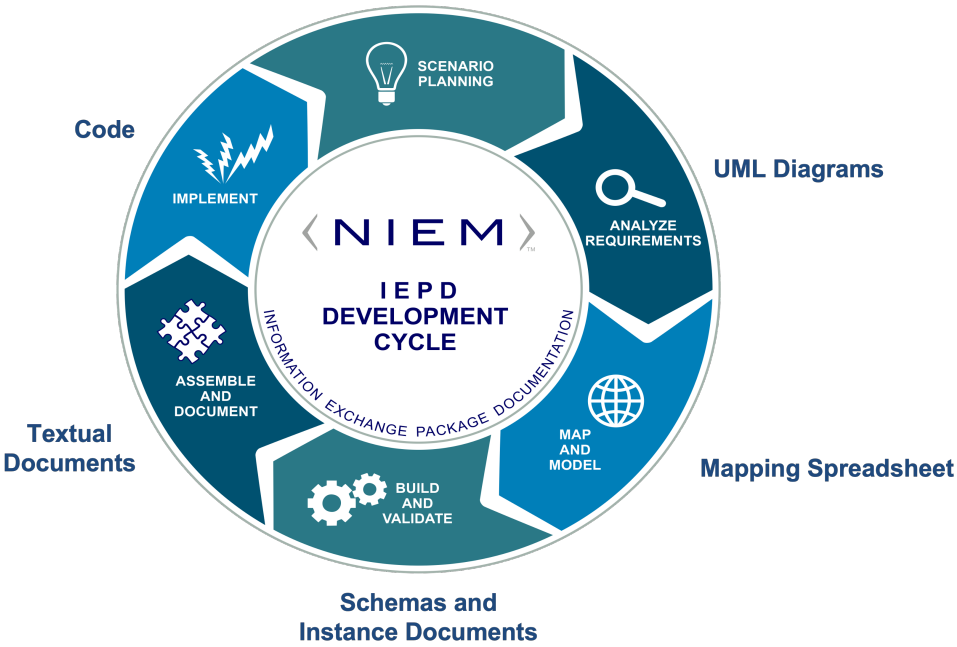
Creating a Message Spec is a multi-step process:

1. Scenario Planning
2. Requirements Analysis
3. Mapping
4. Assembly
5. Publishing
6. (Implementation)

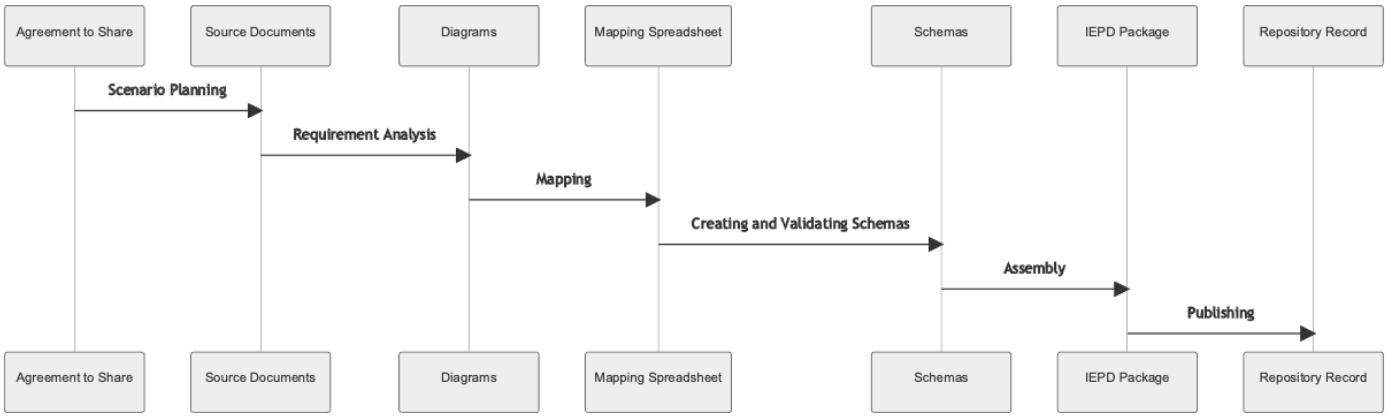


Each step produces artifacts used by subsequent steps:

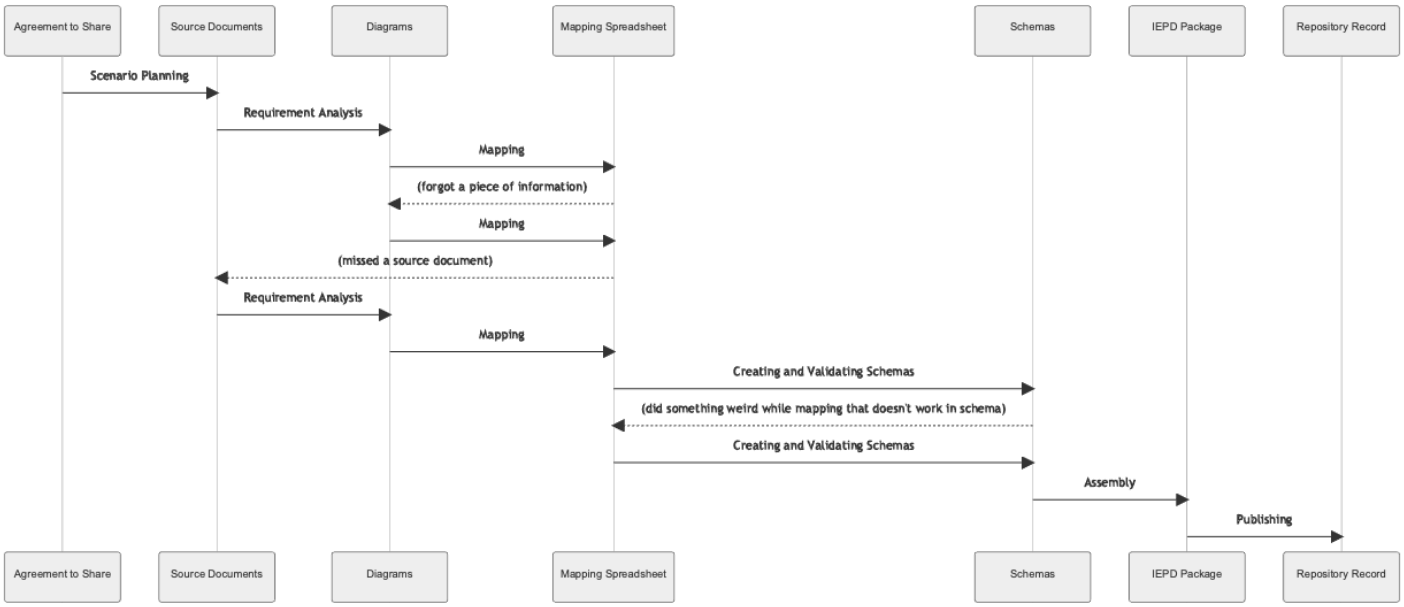
1. A clue as to what you're doing…
2. UML Diagrams
3. Mapping Spreadsheet
4. Schemas and Instance Documents
5. Textual Documents
6. Code



Message Spec / IEPD Process – Idealized



Message Spec / IEPD Process – Real Life



IEPD Artifacts - Documentation

- Master Documentation (Word)
- IEPD catalog document (`iepd-catalog.xml`)
- Change log (text)
- README (text)
- Conformance assertion (text)

IEPD Artifacts - Definitional

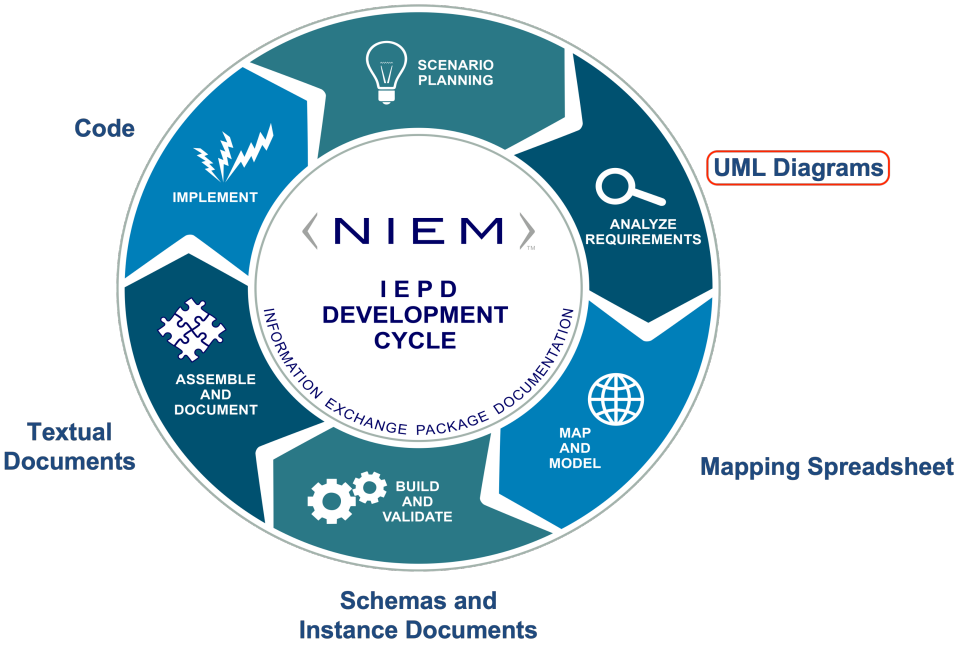
- Wantlist (`wantlist.xml`)
- Schema subset schemas
- Extension schemas
- Exchange schemas
- Sample instances
- XML catalogs

Scenario Planning

- Decide what the exchange is about
- Who are the exchange partners?
- Who are stakeholders?
- Communication is key
- Existing exchanges or other documentation can help
    - Within your organization
    - From NIEM repositories (*huge caveat*)

Existing Documentation

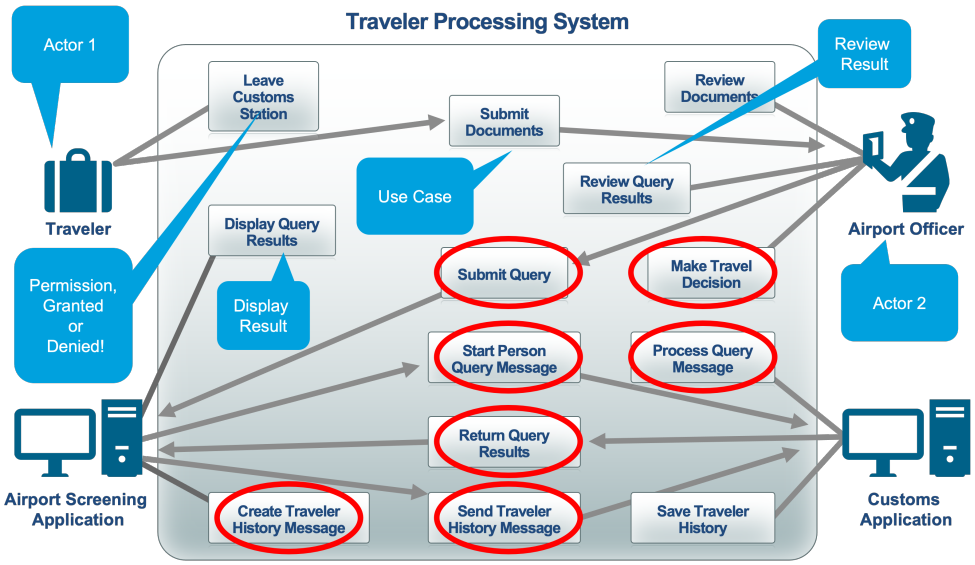
- Current technical architecture documents of all exchange partners
- Stakeholders that will be involved in the exchange
- Security, privacy, and other policy-related concerns associated with the exchange
- Technical characteristics of the exchange:
- Types of data being shared
- Number of data objects
    - Current structure of the data (logical, physical)
    - Use of external standards



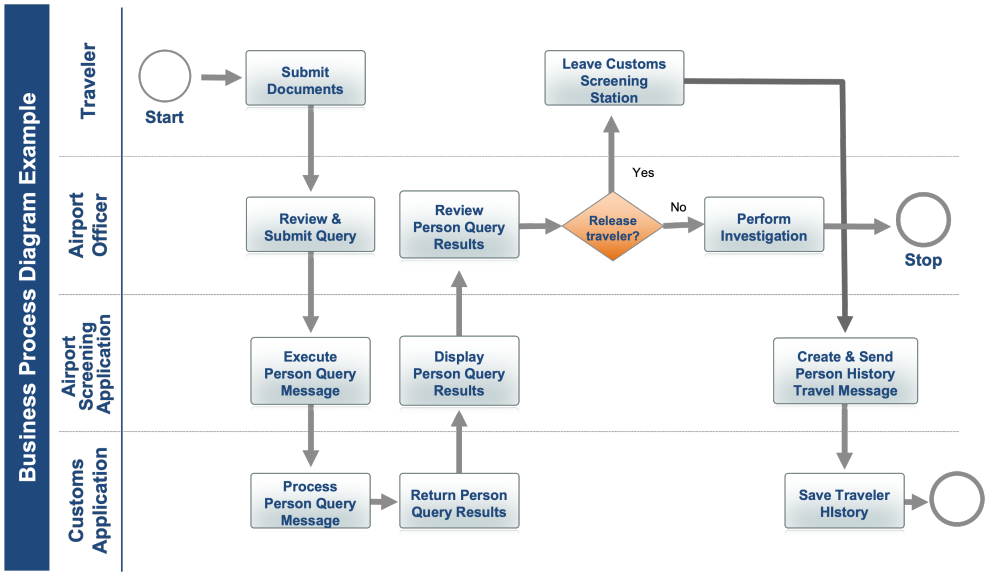
Requirements Analysis

- Diagrams
  - Use Case Diagrams
  - Business Process Diagrams
  - Sequence Diagrams
- Class Diagrams
- Spreadsheets
- Other documents

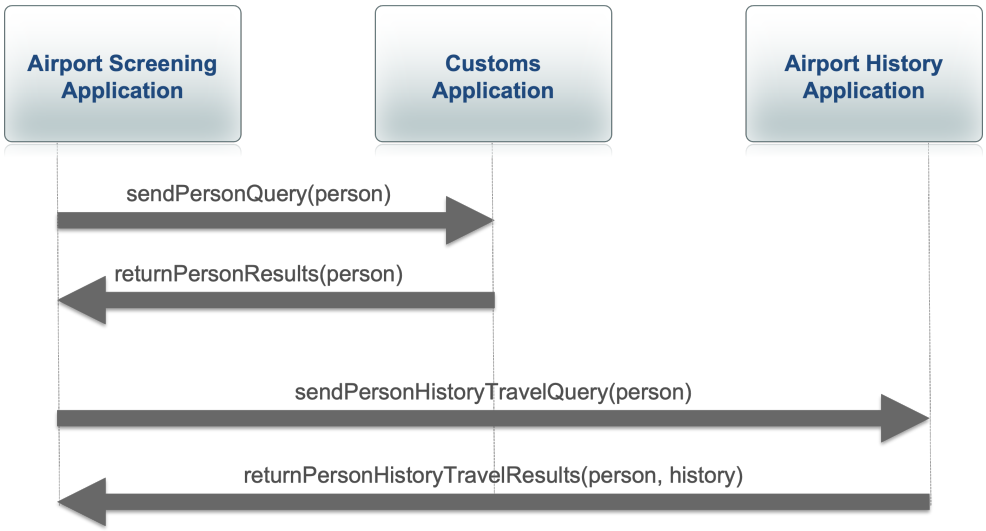
Use Case Diagrams



Business Process Diagrams



Sequence Diagrams

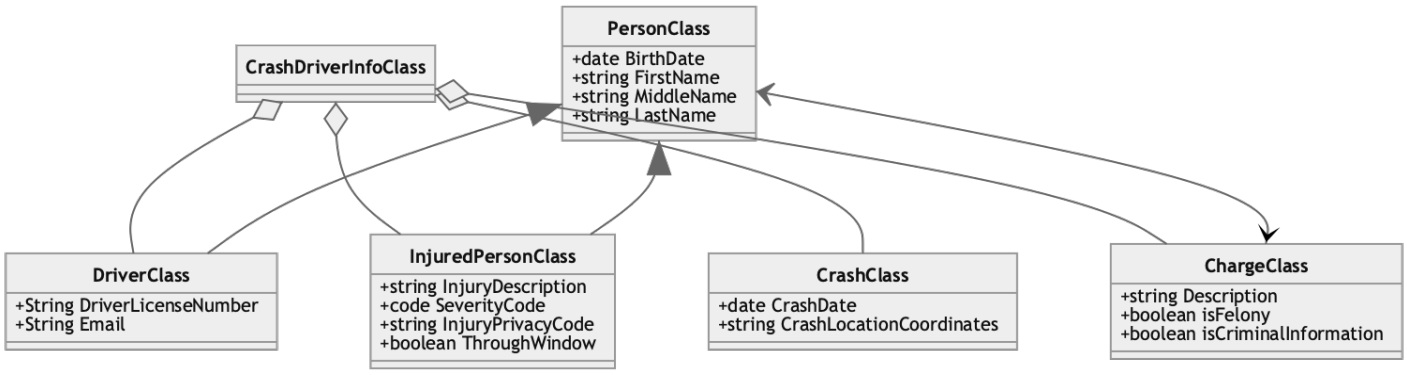


UML Class Diagrams

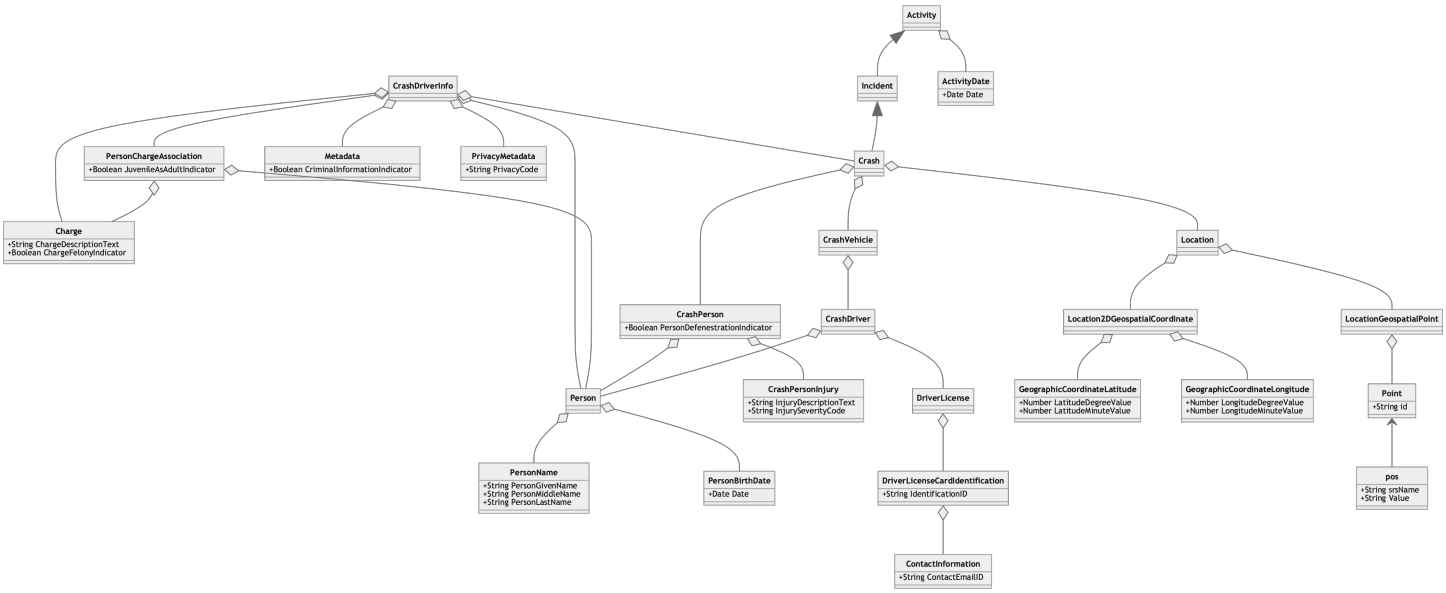
- The bread and butter of IEPD business requirements
- Can be oriented towards business terms and objects
  - Better for consensus
- Can be oriented towards NIEM terms and objects
  - Pre-loads the mapping step

Business Oriented Class Diagram

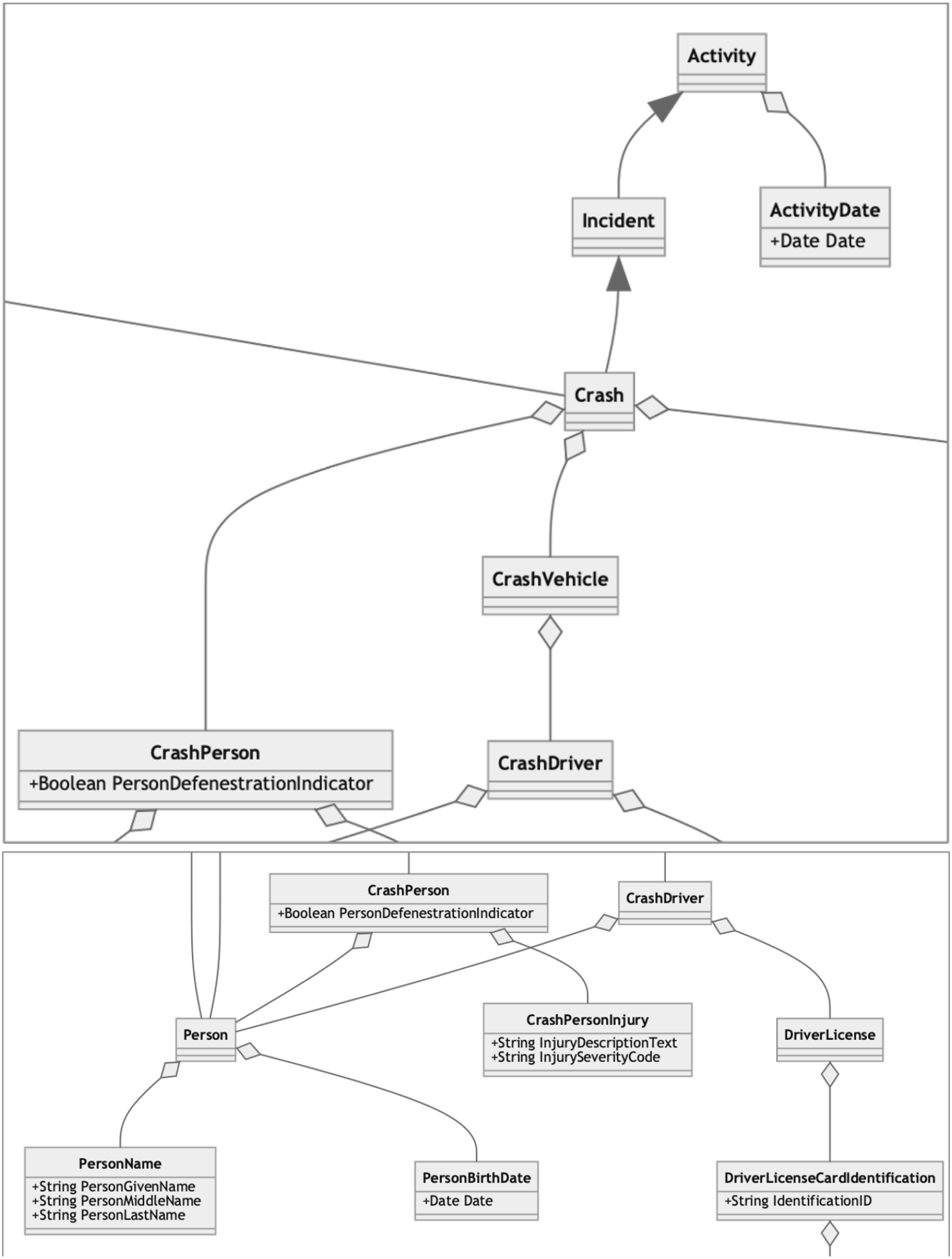
Representing objects in UML by their business names and relationships.



NIEM Oriented Class Diagram



Close-ups:


Activity
Incident
ActivityDate
+Date Date
Crash
CrashVehicle
CrashPerson
+Boolean PersonDefenestrationIndicator
CrashDriver
CrashPerson
+Boolean PersonDefenestrationIndicator
CrashDriver
Person
CrashPersonInjury
+String InjuryDescriptionText
+String InjurySeverityCode
DriverLicense
PersonName
+String PersonGivenName
+String PersonMiddleName
+String PersonLastName
PersonBirthDate
+Date Date
DriverLicenseCardIdentification
+String IdentificationID

UML Tools

- ArgoUML
- BOUML
- MagicDraw / Rational Rose ($$$)
- Visio / OmniGraffle
- Graphviz / Mermaid
- Many more…

Business Rules

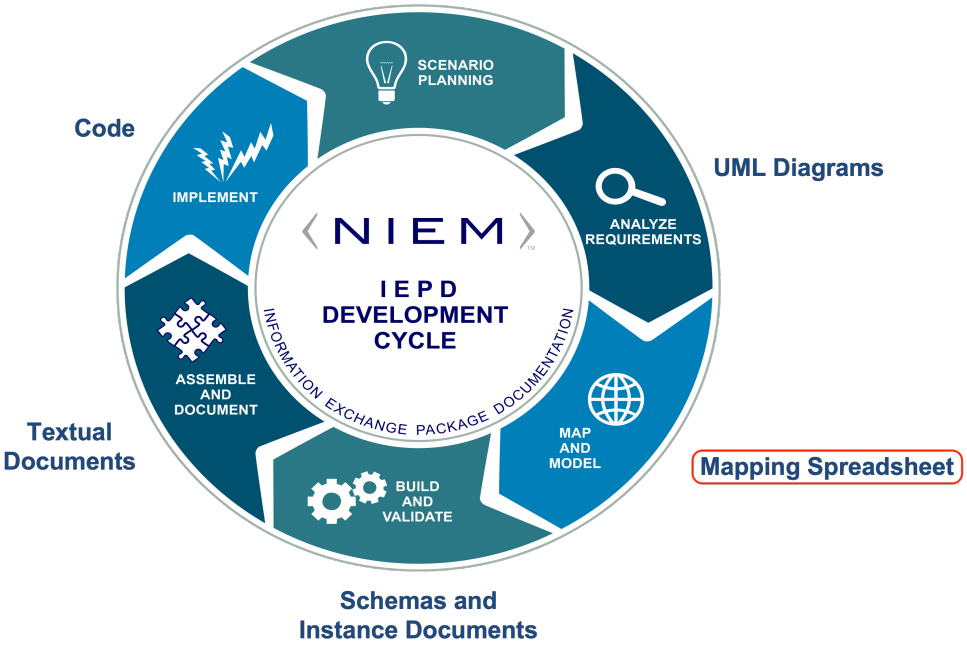
- Some rules are *not* easily represented in UML
- You can use whatever works for your needs
- Schematron for XML
  - e.g. Birth dates must be in the past, salutations matching gender
- Plain old textual descriptions are great!

The Process from Here On…

We have choices on how to proceed:

- Step-by-step
  - Finish mapping entirely before starting schemas
- Concurrent
  - Building schemas and instances as you go
- We'll use something in-between
  - Build an ersatz matching instance document as we map
  - Sorta like YAML without data values
  - Save schemas for the end

(Will use a super secret tool to help with some of this!)



Mapping

For this entire section, we'll look at various things in the mapping spreadsheet and show how to map them to NIEM or to new elements that we'll create later. As we move through, we'll cover all the major aspects of how NIEM works.

Introduction to Mapping

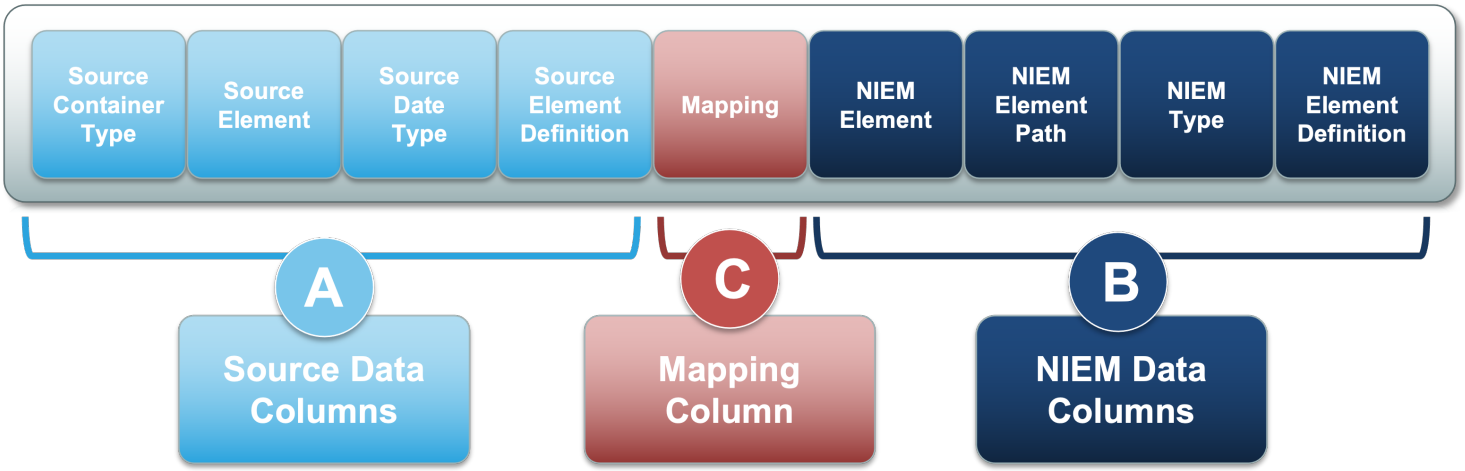
- Mapping Spreadsheets Options
  - NIEM Mapping Template
  - Simple Training Spreadsheet
  - Something In-between
- Document Business Objects
- Map them to NIEM objects, existing or new
- Maintaining an Ongoing Sample Instance Skeleton

NIEM Mapping Template

- Primarily for submitting content for inclusion in NIEM
  - Eight different tabs
- Can also be used for mapping in an IEPD
  - Just need one of the tabs, mainly
- Is a bit overkill for a Message Spec

Simple Training Spreadsheet

- Minimal
- Designed to be simple enough to fit on slides
  - No, really, that's the constraint
- Usually expanded in practice



Tom's Custom Mapping Spreadsheet

- Has evolved over time
- Contains all the info needed to make schemas
- Not as overwhelming as the NIEM Mapping Template
- Works with the NIEM Linter
- You can make your own custom one
  - The IEPD Spec doesn't specify a required format, by design

---

Basics of Searching NIEM

Tools

- SSGT
- Wayfarer
- NIEM Schemas
  - Official Releases
  - HyperNIEM
- Spreadsheet (included in the official releases)

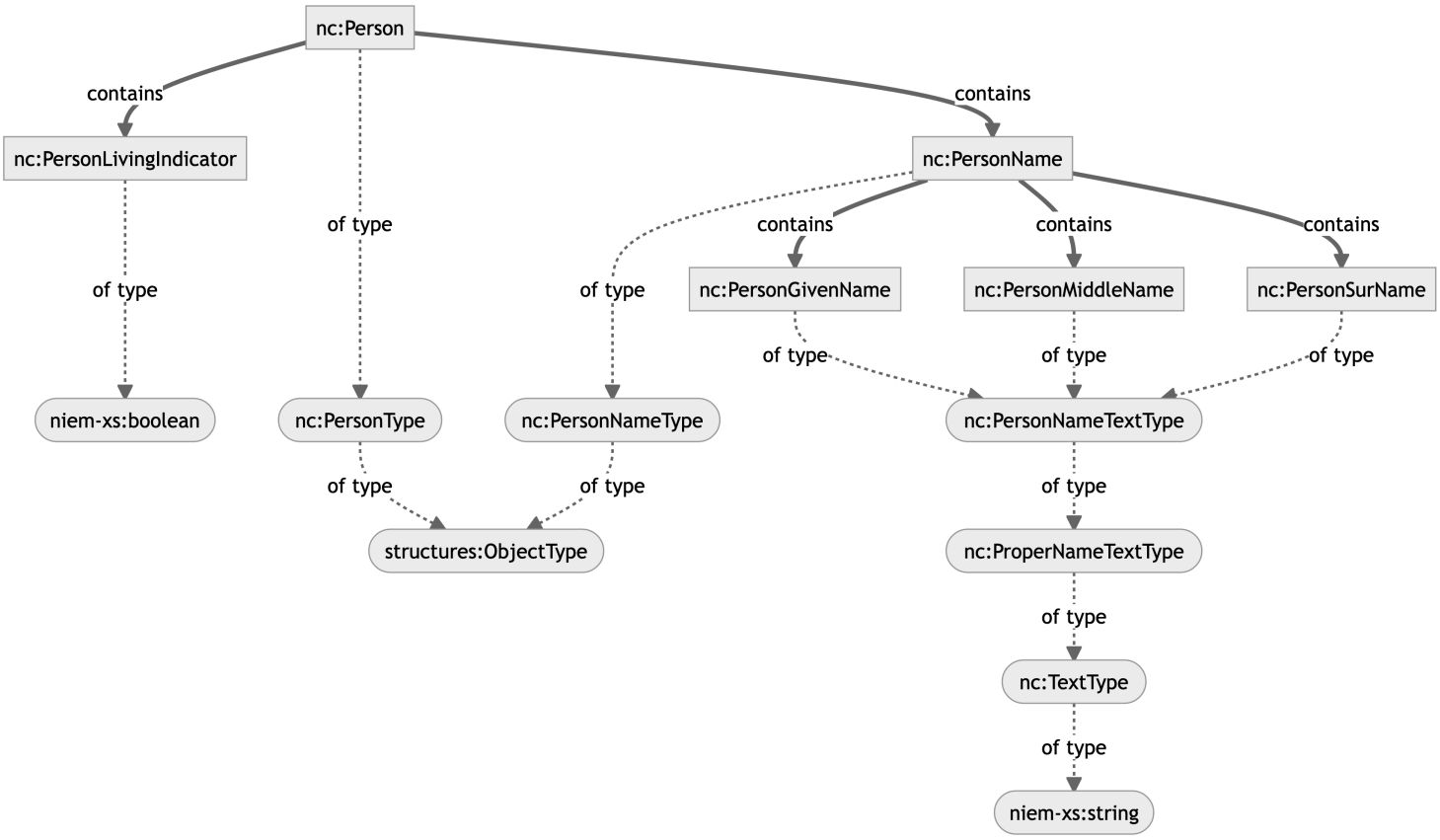
Techniques

- Search for Terms
  - Simple vs Advanced
- Search for Synonyms
- Search for Word Roots
- Search for Containers
- Search for Properties

Mapping and NIEM Technical Details

Understanding NIEM Objects

- XML Schema uses elements and types
- What an element can hold is based on its type
- What you are (your type) defines what you can hold



XML Schema

The XML Schema defining `nc:Person` includes a definition and a type. It looks like:

```
<xs:element name="Person" type="nc:PersonType" nillable="true">
    <xs:annotation>
        <xs:documentation>A human being.</xs:documentation>
    </xs:annotation>
</xs:element>
```

Its type, `nc:PersonType`, has a little more information. It also includes a definition, one similar to `nc:Person`. It also includes a `base` that tells us what sort of thing it is. In this case, the base is `structures:ObjectType`, which is just an empty object (that has a few infrastructure pieces we'll learn about later). To that `base` it adds several objects. These are objects that go *inside* an `nc:Person` object. Each one is a reference to a declaration of each of those objects. Each also has cardinality defined, which tells us how many of each can go inside of an `nc:Person`. `minOccurs` is the minimum number of times. Any non-negative integer can go here, but 0 and 1 are what you'll usually find. Zero essentially means "optional." `maxOccurs` is the maximum number of times. This can also be any non-negative number, but can also be "unbounded", which means "as many as you want." Typical values are 1 and unbounded. Here's the schema for `nc:PersonType` with some of the contained objects removed for clarity:

```
<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:PersonAccentText" minOccurs="0" maxOccurs="unbound
ed"/>
                <xs:element ref="nc:PersonAgeDescriptionText" minOccurs="0" maxOccurs=
"unbounded"/>
                <!-- A whole slew of objects removed for clarity -->
                <xs:element ref="nc:PersonName" minOccurs="0" maxOccurs="unbounded"/>
                <!-- A whole slew of objects removed for clarity -->
                <xs:element ref="nc:PersonHomeContactInformation" minOccurs="0" maxOcc
urs="unbounded"/>
                <xs:element ref="nc:PersonAugmentationPoint" minOccurs="0" maxOccurs="
unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

The XML Schema defining `nc:PersonName` and `nc:PersonNameType` looks like:

```
<xs:element name="PersonName" type="nc:PersonNameType" nillable="true">
    <xs:annotation>
        <xs:documentation>A combination of names and/or titles by which a person is kn
own.</xs:documentation>
    </xs:annotation>
</xs:element>

<xs:complexType name="PersonNameType">
    <xs:annotation>
        <xs:documentation>A data type for a combination of names and/or titles by whic
h a person is known.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:ObjectType">
            <xs:sequence>
                <xs:element ref="nc:PersonNamePrefixAbstract" minOccurs="0" maxOccurs=
"unbounded"/>
                <xs:element ref="nc:PersonGivenName" minOccurs="0" maxOccurs="unbounde
d"/>
                <xs:element ref="nc:PersonMiddleName" minOccurs="0" maxOccurs="unbound
ed"/>
                <xs:element ref="nc:PersonSurName" minOccurs="0" maxOccurs="unbounded"
/>
                <xs:element ref="nc:PersonNameSuffixText" minOccurs="0" maxOccurs="unb
```

```
ounded"/>
                <xs:element ref="nc:PersonMaidenName" minOccurs="0" maxOccurs="unbound
ed"/>
                <xs:element ref="nc:PersonFullName" minOccurs="0" maxOccurs="unbounded
"/>
                <xs:element ref="nc:PersonNameCategoryAbstract" minOccurs="0" maxOccur
s="unbounded"/>
                <xs:element ref="nc:PersonNameSalutationText" minOccurs="0" maxOccurs=
"unbounded"/>
                <xs:element ref="nc:PersonOfficialGivenName" minOccurs="0" maxOccurs="
unbounded"/>
                <xs:element ref="nc:PersonPreferredName" minOccurs="0" maxOccurs="unbo
unded"/>
                <xs:element ref="nc:PersonSurNamePrefixText" minOccurs="0" maxOccurs="
unbounded"/>
                <xs:element ref="nc:EffectiveDate" minOccurs="0" maxOccurs="unbounded"
/>
                <xs:element ref="nc:PersonNameAugmentationPoint" minOccurs="0" maxOccu
rs="unbounded"/>
            </xs:sequence>
            <xs:attribute ref="nc:personNameCommentText" use="optional"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

The XML Schema defining `nc:PersonGivenName`, `PersonNameTextType`, and supporting types looks like:

```
<xs:element name="PersonGivenName" type="nc:PersonNameTextType" nillable="true">
    <xs:annotation>
        <xs:documentation>A first name of a person.</xs:documentation>
    </xs:annotation>
</xs:element>

<xs:complexType name="PersonNameTextType">
    <xs:annotation>
        <xs:documentation>A data type for a name by which a person is known, referred,
 or addressed.</xs:documentation>
    </xs:annotation>
    <xs:simpleContent>
        <xs:extension base="nc:ProperNameTextType">
            <xs:attribute ref="nc:personNameInitialIndicator" use="optional"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="ProperNameTextType">
    <xs:annotation>
        <xs:documentation>A data type for a word or phrase by which a person or thing
```

```
is known, referred, or addressed.</xs:documentation>
    </xs:annotation>
    <xs:simpleContent>
        <xs:extension base="nc:TextType"/>
    </xs:simpleContent>
</xs:complexType>

<xs:complexType name="TextType">
    <xs:annotation>
        <xs:documentation>A data type for a character string.</xs:documentation>
    </xs:annotation>
    <xs:simpleContent>
        <xs:extension base="niem-xs:string">
            <xs:attribute ref="nc:partialIndicator" use="optional"/>
            <xs:attribute ref="nc:truncationIndicator" use="optional"/>
            <xs:attribute ref="xml:lang" use="optional"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

The resulting instance document that all this creates might look like this:

```
<nc:Person>
    <nc:PersonName nc:personNameCommentText="copied">
        <nc:PersonGivenName>Peter</nc:PersonGivenName>
        <nc:PersonMiddleName>Death</nc:PersonMiddleName>
        <nc:PersonMiddleName>Bredon</nc:PersonMiddleName>
        <nc:PersonSurName>Wimsey</nc:PersonSurName>
    </nc:PersonName>
</nc:Person>
```

What About JSON?

NIEM can be used with JSON via JSON-LD. JSON-LD extends JSON to allow for meaningfully linking data together. NIEM leverages JSON-LD in two ways:

1. NIEM uses `@context` to provide a mapping from JSON object names back to their counterparts in NIEM
2. NIEM uses `@id` to provide links between JSON objects

Linking is a topic for later in the class, but here is an example of the `@context`. This context tells us that `nc:Person` refers to the `Person` object in the NIEM-Core namespace.

```
{
    "@context": {
```

```
        "nc": "http://release.niem.gov/niem/niem-core/5.0/#"
    },
    "nc:Person": {
        "nc:PersonName": {
            "nc:personNameCommentText": "copied",
            "nc:PersonGivenName": "Peter",
            "nc:PersonSurName": "Wimsey"
        }
    }
}
```

There are other means to shorten long NIEM names into names more amenable to JSON developers via normalization, but that topic is outside the scope of the training.

Note that the structure of the JSON mirrors the structure of the XML. NIEM with JSON does require mirroring the structure.

JSON-LD and JSON Schema are separate concepts. NIEM does not *yet* support JSON Schema, although efforts are underway to enable that. Currently, JSON with NIEM is all about the JSON instance documents. NIEM itself is still defined in XML Schema.

Throughout the training, matching JSON instances will be included with XML instances.

Native Properties

- Some things will be easy to find
- Will map directly to NIEM objects
- Examples:
  - `nc:Person`, `nc:PersonName`, `nc:PersonGivenName`, etc.
  - Search for `nc:Person` in the SSGT
  - Search for `nc:Person` in Wayfarer

We've already seen the schema for these in detail.

Substitution Groups

- Some concepts can be represented multiple ways
- Text / code combinations are common
- Date can be a date, datetime, or a range
- NIEM uses substitution groups to support both:
  - The single concept, and
  - Multiple representations of that concept
- Examples:
  - `nc:PersonBirthDate` (SSGT/Wayfarer) contains a `nc:DateRepresentation` (SSGT/Wayfarer)
  - Substitution group heads follow the form of: `SomethingRepresentation` or `WhateverAbstract`

Schemas

Here we see `nc:PersonBirthDate` and its type, `nc:DateType`:

```
<xs:element name="PersonBirthDate" type="nc:DateType" nillable="true">
    <xs:annotation>
        <xs:documentation>A date a person was born.</xs:documentation>
    </xs:annotation>
</xs:element>
```

`nc:DateType` contains `nc:DateRepresentation`, along with other date properties:

```
<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
nded"/>
                <xs:element ref="nc:DateAccuracyAbstract" minOccurs="0" maxOccurs="1"/
>
                <xs:element ref="nc:DateMarginOfErrorDuration" minOccurs="0" maxOccurs
="1"/>
                <xs:element ref="nc:DateAugmentationPoint" minOccurs="0" maxOccurs="un
bounded"/>
            </xs:sequence>
        </xs:extension>
```

```
    </xs:complexContent>
</xs:complexType>
```

`nc:DateRepresentation` is *abstract*, meaning it has no type and must be substituted with another object:

```
<xs:element name="DateRepresentation" abstract="true">
    <xs:annotation>
        <xs:documentation>A data concept for a representation of a date.</xs:documenta
tion>
    </xs:annotation>
</xs:element>
```

Those other objects are identified by having a `substitutionGroup` attribute set to the name of the substitution group head. The most common one you'll see for `nc:DateRepresentation` is `nc:Date`:

```
<xs:element name="Date" type="niem-xs:date" substitutionGroup="nc:DateRepresentation"
nillable="true">
    <xs:annotation>
        <xs:documentation>A full date.</xs:documentation>
    </xs:annotation>
</xs:element>
```

Instance Documents

In the resulting instance document, you don't see `nc:DateRepresentation` at all. You just see `nc:Date`, which is taking its place:

```
<nc:Person>
    <nc:PersonBirthDate>
        <nc:Date>1890-05-04</nc:Date>
    </nc:PersonBirthDate>
    <nc:PersonName>
        <nc:PersonGivenName>Peter</nc:PersonGivenName>
        <nc:PersonMiddleName>Death</nc:PersonMiddleName>
        <nc:PersonMiddleName>Bredon</nc:PersonMiddleName>
        <nc:PersonSurName>Wimsey</nc:PersonSurName>
    </nc:PersonName>
</nc:Person>
```

The matching JSON similarly just shows `nc:Date`:

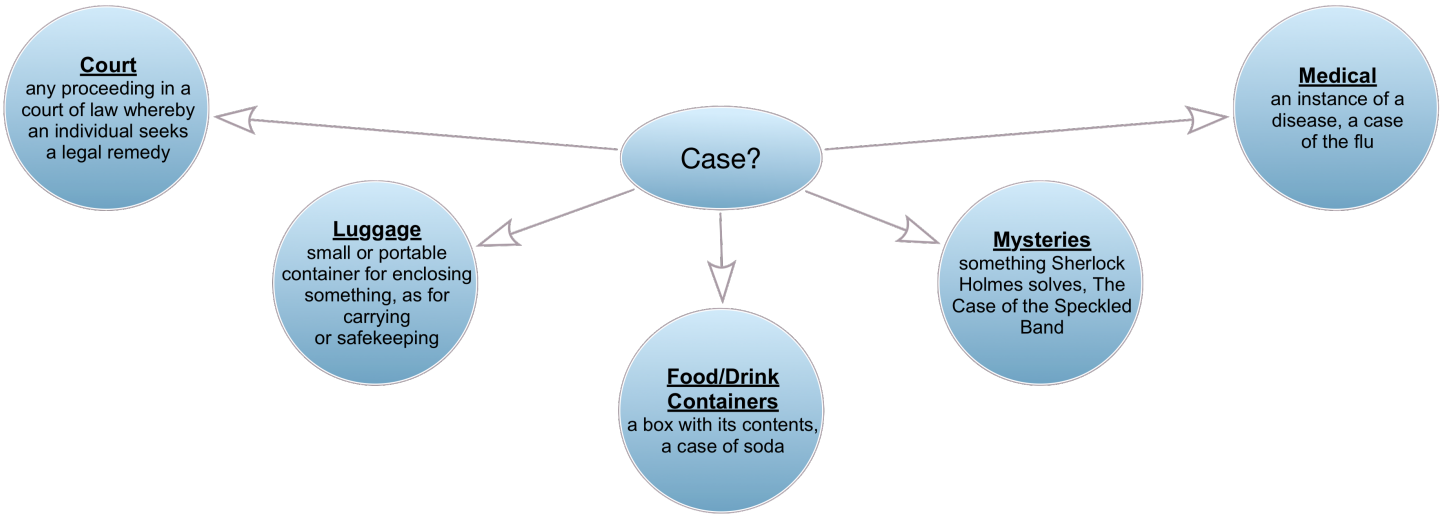
```
"nc:Person": {
```

```
    "nc:PersonBirthDate": {
        "nc:Date": "1890-05-04"
    },
    "nc:PersonName": {
        "nc:PersonGivenName": "Peter",
        "nc:PersonMiddleName": [
            "Death",
            "Bredon"
        ],
        "nc:PersonSurName": "Wimsey"
    }
}
```

Namespaces

Above you can see object with different prefixes, `j:Crash` and `nc:ActivityDate`. The `j` and `nc` refer to different namespaces in XML Schema.

- Namespaces organize elements by context
- Identified by prefix, a nickname for the namespace
- NIEM governance is organized along these lines



JSON-LD maps JSON objects to NIEM namespaces in the `@context`. Each of these entries maps a prefix to a NIEM namespace, providing a link back to the NIEM object.

```
"@context": {
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "nc": "http://release.niem.gov/niem/niem-core/5.0/#",
    "exch": "http://example.com/CrashDriver/1.0/#",
    "ext": "http://example.com/CrashDriver/1.0/extension#",
```

```
    "j": "http://release.niem.gov/niem/domains/jxdm/7.0/#"
}
```

As mentioned earlier, NIEM doesn't support JSON Schema well yet. Using NIEM with JSON is currently focused on creating matching instance documents. Upcoming NIEM developments will greatly enhance the ability to work in JSON as a similar level as with XML and XML Schema.

Inherited Properties

- NIEM is a model, not a flat data dictionary
- Some concepts don't exist as elements using the terms for the concept
- Instead, properties are inherited
- There is no "Crash Date" in NIEM (SSGT/Wayfarer)
- There *is* an `ActivityDate` which can be inside a `Crash` object (SSGT/Wayfarer)

Schemas

Here's `j:Crash` and its type, `j:CrashType`:

```
<xs:element name="Crash" type="j:CrashType" nillable="true">
    <xs:annotation>
        <xs:documentation>A traffic accident.</xs:documentation>
    </xs:annotation>
</xs:element>
```

`j:CrashType` contains several things, but the important thing here is what it's based on, `j:DrivingIncidentType`:

```
<xs:complexType name="CrashType">
    <xs:annotation>
        <xs:documentation>A data type for a traffic accident.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="j:DrivingIncidentType">
            <xs:sequence>
                <xs:element ref="j:CrashServiceCall" minOccurs="0" maxOccurs="unbounde
d"/>
                <xs:element ref="j:CrashInformationSource" minOccurs="0" maxOccurs="un
bounded"/>
                <!-- A whole slew of objects removed for clarity -->
                <xs:element ref="j:CrashAugmentationPoint" minOccurs="0" maxOccurs="un
bounded"/>
```

```
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

`j:DrivingIncidentType` is, in turn, based on an even more generic type, `nc:IncidentType`:

```
<xs:complexType name="DrivingIncidentType">
    <xs:annotation>
        <xs:documentation>A data type for details of an incident involving a vehicle.<
/xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="nc:IncidentType">
            <xs:sequence>
                <xs:element ref="j:DrivingAccidentSeverityAbstract" minOccurs="0" maxO
ccurs="unbounded"/>
                <xs:element ref="j:DrivingIncidentCMVAbstract" minOccurs="0" maxOccurs
="unbounded"/>
                <!-- A whole slew of objects removed for clarity -->
                <xs:element ref="j:DrivingIncidentAugmentationPoint" minOccurs="0" max
Occurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

`nc:IncidentType` is, also in turn, based on a very generic type, `nc:ActivityType`:

```
<xs:complexType name="IncidentType">
    <xs:annotation>
        <xs:documentation>A data type for an occurrence or an event that may require a
 response.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="nc:ActivityType">
            <xs:sequence>
                <xs:element ref="nc:IncidentEvent" minOccurs="0" maxOccurs="unbounded"
/>
                <xs:element ref="nc:IncidentJurisdictionalOrganization" minOccurs="0"
maxOccurs="unbounded"/>
                <!-- A whole slew of objects removed for clarity -->
                <xs:element ref="nc:IncidentAugmentationPoint" minOccurs="0" maxOccurs
="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

And we finally get to `nc:ActivityType`, which contains `nc:ActivityDate`:

```
<xs:complexType name="ActivityType">
    <xs:annotation>
        <xs:documentation>A data type for a single or set of related actions, events,
or process steps.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:ObjectType">
            <xs:sequence>
                <xs:element ref="nc:ActivityIdentification" minOccurs="0" maxOccurs="u
nbounded"/>
                <xs:element ref="nc:ActivityActualDuration" minOccurs="0" maxOccurs="u
nbounded"/>
                <!-- A whole slew of objects removed for clarity -->
                <xs:element ref="nc:ActivityDate" minOccurs="0" maxOccurs="unbounded"/
>
                <!-- A whole slew of objects removed for clarity -->
                <xs:element ref="nc:ActivityAugmentationPoint" minOccurs="0" maxOccurs
="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

What this all means is that a `j:Crash` object can contain a `nc:ActivityDate`, which is, contextually, a Crash Date.

Instance Documents

Instance in XML:

```
<j:Crash>
    <nc:ActivityDate>
        <nc:Date>1900-05-04</nc:Date>
    </nc:ActivityDate>
</j:Crash>
```

Instance in JSON:

```
"j:Crash": {
    "nc:ActivityDate": {
        "nc:Date": "1900-05-04"
    }
}
```

You need to understand this concept in order to know to look for these cases, which are very common, but just use tools to figure out the details, e.g:

- `j:Crash` in the SSGT
- `j:Crash` in Wayfarer

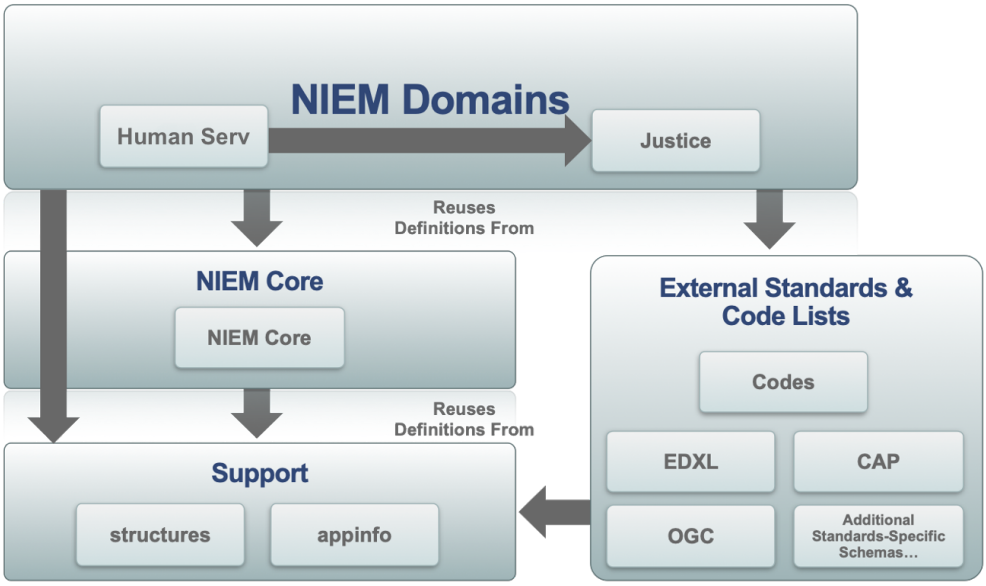
Linking Things Together

- NIEM is relational in many senses
- Objects can refer to each other across an XML hierarchy
- Allows NIEM to represent real world, many-to-many relationships
- `structures` namespace provides the infrastructure to make this work

NIEM Structural "Layers"

NIEM has several conceptual layers which build on top of each other:

- `structures` provides infrastructure
  - All namespaces draw from it
- `niem-core` provides common objects to be used or built upon
  - Domains draw from it
- Domains provide domain-specific content, often built from `niem-core`
  - Domains can draw from each other, although this is limited in practice
- Code table namespaces define many of the code tables, built with infrastructure from `structures`
  - Domains draw from it for code definitions
- Wrappers for external standards, built with infrastructure from `structures`



Referencing - XML Schema

- NIEM lets you assign unique IDs to object
- Other objects can then link to those objects by referencing the ID
- Everything in NIEM can have attributes for this:
  - `id`
  - `ref`
  - `metadata`
- These leverage built-in XML Schema attributes `ID`, `IDREF`, and `IDREFS`
- `id` assigns an ID to an object
  - Just a string
  - *Must* be unique within an instance document
  - Can't include a space
- `ref` references an ID of another object
  - Contains a single `id` to match
  - The matching `id` *must* exist in the instance document
  - Validators do *not* check that the linking makes sense
- `metadata` references IDs of metadata object
  - Conceptually, an object is saying that this is the metadata that applies to itself
  - Can contain multiple IDs, separated with spaces
  - The matching `id`s must exist in the instance document

Examples of how NIEM uses these are the next few sections.

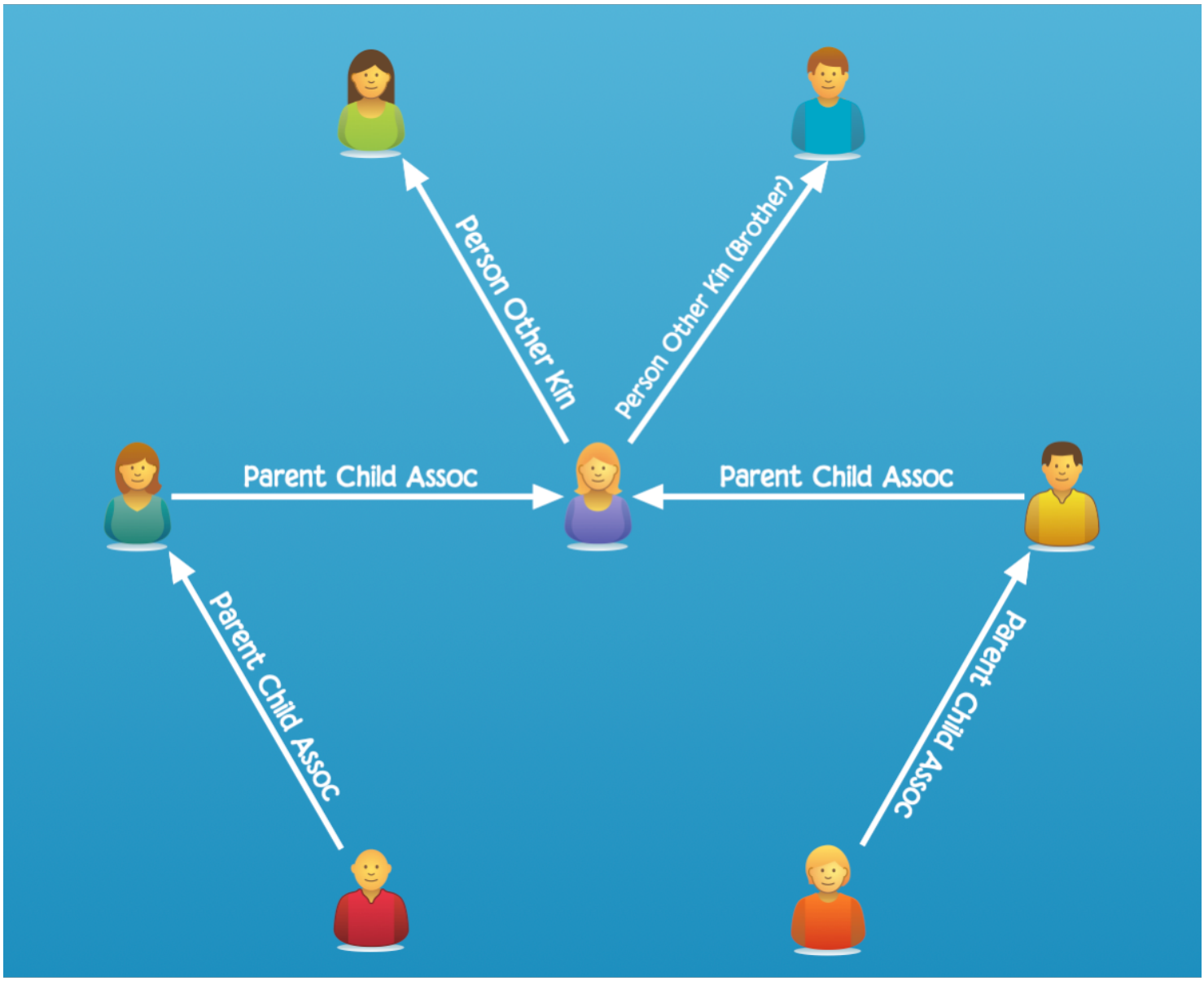
Referencing - JSON-LD

JSON itself doesn't provide a means of linking objects together. NIEM leverages JSON-LD to provide that functionality. We've seen earlier how the `@context` section provides a mapping from a JSON instance back to NIEM object. NIEM also uses `@id` objects to both mark objects with an ID as well as refer to IDs applied to other objects.

Again, we will see example throughout the next few sections.

Associations

- Relationships can be complex
- NIEM provides powerful Association objects
- Trade-off can be implementation complexity



- Associations link objects together
  - Associations reference the associated objects
  - Can link to multiple objects
  - Can thus form many-to-many relationships between them
- Associations also hold information about the association itself
  - `Date` or `DateRange` is a common example
- Associations can also just include them, if applicable
- Examples:
  - `j:PersonChargeAssociation` (SSGT/Wayfarer)
  - Anything ending in “Association” (SSGT/Wayfarer)

Schemas

`j:PersonChargeAssociation` is used, as the name suggests, to link together a Person and a Charge. It's of `j:PersonChargeAssociationType`:

```
<xs:element name="PersonChargeAssociation" type="j:PersonChargeAssociationType" nillable="true">
    <xs:annotation>
        <xs:documentation>An association between a person and a charge issued to that person.</xs:documentation>
    </xs:annotation>
</xs:element>
```

`j:PersonChargeAssociationType` includes a `nc:Person` and a `j:Charge`, but also includes information *about* the association itself, in this case `j:JuvenileAsAdultIndicator`. `PersonChargeAssociationType` extends `nc:AssociationType`:

```
<xs:complexType name="PersonChargeAssociationType">
    <xs:annotation>
        <xs:documentation>A data type for an association between a person and a charge.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="nc:AssociationType">
            <xs:sequence>
                <xs:element ref="nc:Person" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="j:Charge" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="j:JuvenileAsAdultIndicator" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="j:PersonChargeAssociationAugmentationPoint" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

`nc:AssociationType` adds in generic association information like a start and end date and a description. Remember that these are inherited by any type based on `nc:AssociationType`, so things of `j:PersonChargeAssociationType` automatically get these common objects.

```
<xs:complexType name="AssociationType">
    <xs:annotation>
        <xs:documentation>A data type for an association, connection, relationship, or involvement somehow linking people, things, and/or activities together.</xs:documentation>
    </xs:annotation>
```

```
    <xs:complexContent>
        <xs:extension base="structures:AssociationType">
            <xs:sequence>
                <xs:element ref="nc:AssociationDateRange" minOccurs="0" maxOccurs="unb
ounded"/>
                <xs:element ref="nc:AssociationDescriptionText" minOccurs="0" maxOccur
s="unbounded"/>
                <xs:element ref="nc:AssociationAugmentationPoint" minOccurs="0" maxOcc
urs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Instance Documents

In the instance document, the association object can specify the objects being associated together by pointing to then with `ref` attributes, or by including the object inside the association object.

First, here's the association using referencing. The applicable `nc:Person` and `j:Charge` objects are external to `j:PersonChargeAssociation`. While shown next to the `j:PersonChargeAssociation` object, they could be *anywhere* in the instance document.

This method is useful when the same `nc:Person` object is being used in multiple contexts. In the example IEPD, this `nc:Person` is also the `j:CrashDriver` and the `j:CrashPerson`. This method allows us to define the `nc:Person` once and refer to it multiple times.

This beings with it two advantages:

1. This `nc:Person` object is not being replicated; there is no duplication of data
2. We know that the `nc:Person` in the Association *and* the `j:CrashDriver` *and* the `j:CrashPerson` are the exact same person, and not three people with the same name

```
<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>
<nc:Person structures:id="P01">
    <nc:PersonName>
        <nc:PersonGivenName>Tommy</nc:PersonGivenName>
    </nc:PersonName>
</nc:Person>
<j:Charge structures:id="CH01">
    <j:ChargeDescriptionText>Furious Driving</j:ChargeDescriptionText>
    <j:ChargeFelonyIndicator>false</j:ChargeFelonyIndicator>
```

```
</j:Charge>
```

The disadvantage with using referencing is that it can be more difficult to implement. Implementations need to look for `ref` attributes and find matching `id` attributes.

The alternative to referencing is to just include the `nc:Person` and `j:Charge` information inside the association:

```
<j:PersonChargeAssociation>
    <nc:Person>
        <nc:PersonName>
            <nc:PersonGivenName>Tommy</nc:PersonGivenName>
        </nc:PersonName>
    </nc:Person>
    <j:Charge>
        <j:ChargeDescriptionText>Furious Driving</j:ChargeDescriptionText>
        <j:ChargeFelonyIndicator>false</j:ChargeFelonyIndicator>
    </j:Charge>
    <j:JuvenileAsAdultIndicator>true</j:JuvenileAsAdultIndicator>
</j:PersonChargeAssociation>
```

You can also mix and match. You could reference a `nc:Person` while including the `j:Charge` inside the association.

NIEM *never* requires you to do referencing. If the trade-offs of duplicated data aren't an issue, you can have multiple identical `nc:Person` objects.

JSON-LD works similarly. `j:PersonChargeAssociation` contains a `nc:Person` object with an `@id`. `j:Charge` is the same. These IDs match the `@id` objects in the actual `nc:Person` and `j:Charge` object outside the association:

```
"j:PersonChargeAssociation": {
    "nc:Person": {
        "@id": "#P01"
    },
    "j:Charge": {
        "@id": "#CH01"
    },
    "j:JuvenileAsAdultIndicator": true,
    "j:CriminalInformationIndicator": true
},
"nc:Person": {
    "@id": "#P01",
    "nc:PersonName": {
        "nc:PersonGivenName": "Tommy"
    }
},
```

```
"j:Charge": {
    "@id": "#CH01",
    "j:ChargeDescriptionText": "Furious Driving",
    "j:ChargeFelonyIndicator": false,
    "j:CriminalInformationIndicator": true
}
```

As with the XML version, you can also just include the `nc:Person` and `j:Charge` objects inside the association:

```
"j:PersonChargeAssociation": {
    "nc:Person": {
        "nc:PersonName": {
            "nc:PersonGivenName": "Tommy"
        }
    },
    "j:Charge": {
        "j:ChargeDescriptionText": "Furious Driving",
        "j:ChargeFelonyIndicator": false,
        "j:CriminalInformationIndicator": true
    },
    "j:JuvenileAsAdultIndicator": true,
    "j:CriminalInformationIndicator": true
}
```

The same trade-offs apply as in XML, but JSON developers may lean more towards inclusion rather than referencing based on implementation effort required. Again, NIEM *never* requires you to do referencing.

Roles

- Modeling some objects as specialized things gets complicated
- Roles are, well, roles that objects play
  - People, organizations, items are the major ones
- Roles reference the objects playing the role
  - Objects can thus play multiple roles
- Roles can also just include the object playing the role
- Roles contain other information about the role
- Examples:
  - `j:CrashPerson` (SSGT/Wayfarer)
  - Anything containing an `nc:RoleOfPerson`(SSGT/Wayfarer), `nc:RoleOfOrganization` (SSGT/Wayfarer), or `nc:RoleOfItem` (SSGT/Wayfarer), plus a few lesser used ones (SSGT/Wayfarer)

Roles - Not Special Kinds of People

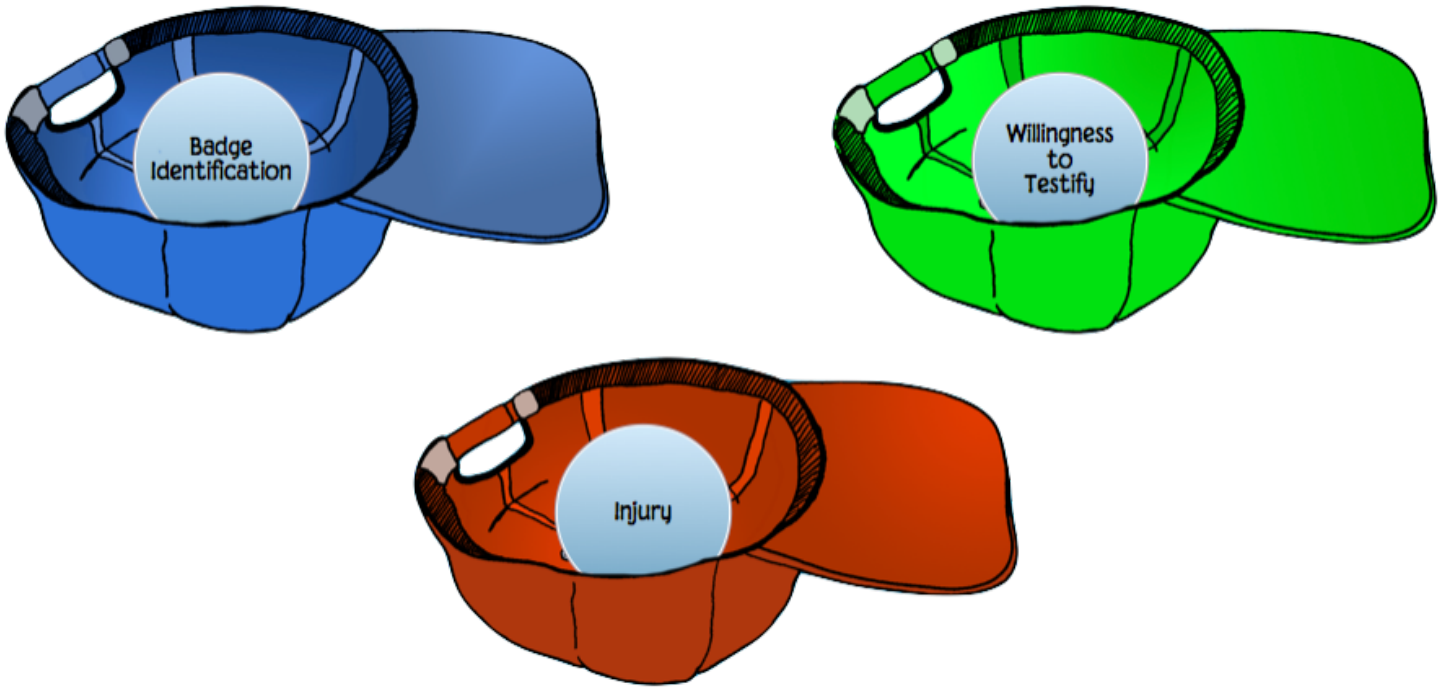
Roles	Just a Guy
	

Roles – Allows an Object to Play Multiple Roles

Roles - How They Work

Roles Point to the Object Playing the Role:

Roles Contain Info About the Role



Schemas

`j:CrashPerson` is of `j:CrashPersonType`:

```
<xs:element name="CrashPerson" type="j:CrashPersonType" nillable="true">
    <xs:annotation>
        <xs:documentation>A person involved in a traffic accident.</xs:documentation>
    </xs:annotation>
</xs:element>
```

`j:CrashPersonType` contains a `nc:RoleOfPerson` object, along with information specific to this role. In the sample IEPD, we're also using `j:CrashPersonInjury`:

```
<xs:complexType name="CrashPersonType">
    <xs:annotation>
        <xs:documentation>A data type for any person involved in a traffic accident.</
xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:ObjectType">
            <xs:sequence>
                <xs:element ref="nc:RoleOfPerson" minOccurs="0" maxOccurs="unbounded"/
>
                <xs:element ref="j:CrashPersonEMSTransportation" minOccurs="0" maxOccu
rs="unbounded"/>
                <xs:element ref="j:CrashPersonInjury" minOccurs="0" maxOccurs="unbound
ed"/>
                <!-- A whole slew of objects removed for clarity -->
                <xs:element ref="j:CrashPersonAugmentationPoint" minOccurs="0" maxOccu
rs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

`nc:RoleOfPerson` is of `nc:PersonType`. That means it can hold person-specific information, but like everything in NIEM, it can also point at other objects:

```
<xs:element name="RoleOfPerson" type="nc:PersonType" substitutionGroup="nc:RoleOfAbstr
act" nillable="true">
    <xs:annotation>
        <xs:documentation>A person of whom the role object is a function.</xs:document
ation>
    </xs:annotation>
</xs:element>
```

Instance Documents

As with associations, we can implement the role in two different ways. Here we see it used as a reference. `j:CrashDriver` contains a `nc:RoleOfPerson`, which uses the `ref` attribute to point to the `nc:Person` object playing that role:

```
<j:CrashDriver>
    <nc:RoleOfPerson structures:ref="P01" xsi:nil="true"/>
</j:CrashDriver>

<nc:Person structures:id="P01">
    <nc:PersonName>
        <nc:PersonGivenName>Peter</nc:PersonGivenName>
    </nc:PersonName>
</nc:Person>
```

As with associations, `nc:RoleOfPerson` can also simply contain the Person-specific information. So it can contain `nc:PersonName` instead of referencing an object elsewhere:

```
<j:CrashDriver>
    <nc:RoleOfPerson>
        <nc:PersonName>
            <nc:PersonGivenName>Peter</nc:PersonGivenName>
        </nc:PersonName>
    </nc:RoleOfPerson>
</j:CrashDriver>
```

As with associations, the choice comes down to a balance between how often the Person is used, whether you care about duplicate data in an exchange, and implementation effort.

The JSON-LD versions are, again, similar to the ones for associations. Here the `nc:RoleOfPerson` object has an `@id` that matches the one in the `nc:Person` object:

```
"j:CrashPerson": {
    "nc:RoleOfPerson": {
        "@id": "#P01"
    }
},

"nc:Person": {
    "@id": "#P01",
    "nc:PersonName": {
        "nc:PersonGivenName": "Tommy"
    }
}
```

And, again as with associations, you can simply include the Person-specific information inside `nc:RoleOfPerson`:

```
"j:CrashPerson": {
    "nc:RoleOfPerson": {
        "nc:PersonName": {
            "nc:PersonGivenName": "Tommy"
        }
    }
}
```

The choices here are also a balance, although inclusion will likely be more attractive to most JSON developers.

Code Tables

- Codes help ensure accurate information
- Codes are, essentially, strings, simple data
- A few in NIEM are integers
  - Elements defined in the domains
  - Types are often defined in their own namespaces
- NIEM wraps them in a complex type in order to apply some attributes needed for infrastructure
  - Which we will need in the next section…
- Examples:
  - `j:InjurySeverityCode` (SSGT/Wayfarer)
    - In the SSGT, the actual codes are viewable on the page for the base simple type, e.g. `aamva_d20:AccidentSeverityCodeSimpleType`
    - In Wayfarer, there should be a link on the element page bringing up the codes in a separate window, e.g. `aamva_d20:AccidentSeverityCodeSimpleType`
  - Anything ending in “Code” (SSGT/Wayfarer)
- Codes nearly always have a text alternative

Schemas

`j:InjurySeverityCode` is a code table, with its codes defined in another namespace, the one for AAMVA. Here's the schema for the element:

```
<xs:element name="InjurySeverityCode" type="aamva_d20:AccidentSeverityCodeType" substi
tutionGroup="nc:InjurySeverityAbstract" nillable="true">
    <xs:annotation>
        <xs:documentation>A severity of an injury received by a person, such as in a t
raffic accident or crash.</xs:documentation>
    </xs:annotation>
</xs:element>
```

The actual codes are defined in a pair of types. The first, `aamva_d20:AccidentSeverityCodeSimpleType`, defines the actual codes and their definitions, one per `enumeration` below. This simple type is then wrapped in a complex type, `aamva_d20:AccidentSeverityCodeType`, to add infrastructure attributes that we've seen used in associations and roles. The two types are grouped together in the `aamva_d20` namespace so they can be governed by AAMVA without needing to change the `j` domain:

```
<xs:simpleType name="AccidentSeverityCodeSimpleType">
    <xs:annotation>
        <xs:documentation>A data type for severity levels of an accident.</xs:document
ation>
    </xs:annotation>
    <xs:restriction base="xs:token">
        <xs:enumeration value="1">
            <xs:annotation>
                <xs:documentation>Fatal Accident</xs:documentation>
            </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="2">
```

```
            <xs:annotation>
                <xs:documentation>Incapacitating Injury Accident</xs:documentation>
            </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="3">
            <xs:annotation>
                <xs:documentation>Non-incapacitating Evident Injury</xs:documentation>
            </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="4">
            <xs:annotation>
                <xs:documentation>Possible Injury Accident</xs:documentation>
            </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="5">
            <xs:annotation>
                <xs:documentation>Non-injury Accident</xs:documentation>
            </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="9">
            <xs:annotation>
                <xs:documentation>Unknown</xs:documentation>
            </xs:annotation>
        </xs:enumeration>
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="AccidentSeverityCodeType">
    <xs:annotation>
        <xs:documentation>A data type for severity levels of an accident.</xs:document
ation>
    </xs:annotation>
    <xs:simpleContent>
        <xs:extension base="aamva_d20:AccidentSeverityCodeSimpleType">
            <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

Instance Documents

The code is what shows up in the instance document. The longer definition does not. If you want to know that “3” means “Non-incapacitating Evident Injury,” you need to refer to the schemas. Schema validation *will* verify that the value is one of the enumerations. Here’s the XML:

```
<j:CrashPersonInjury>
    <nc:InjuryDescriptionText>Broken Arm</nc:InjuryDescriptionText>
    <j:InjurySeverityCode>3</j:InjurySeverityCode>
```

```
</j:CrashPersonInjury>
```

And here's the JSON. Note that NIEM does not yet support JSON Schema, so there's no means for validating the value of the code short of writing your own JSON Schema:

```
"j:CrashPersonInjury": {
    "nc:InjuryDescriptionText": "Broken Arm",
    "j:InjurySeverityCode": "3"
}
```

Metadata

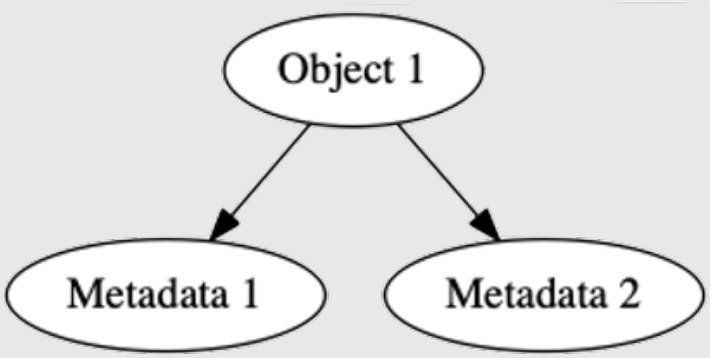
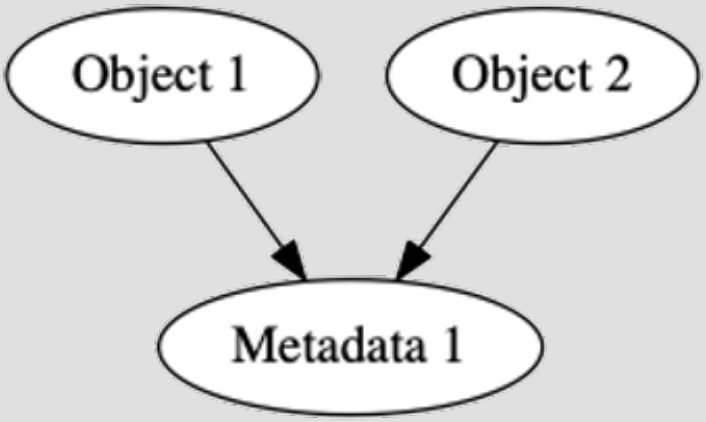
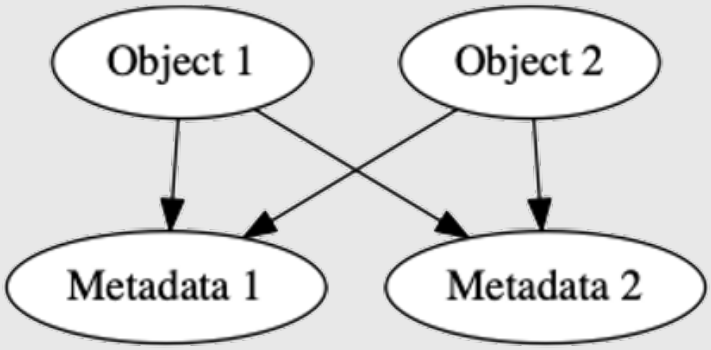
Metadata is Data about Data. What does that mean? Here's an example:

Data	Metadata
Name: **Jett**	*Reporting Organization:* **Godspeed Animal Care**
Sex: **Female**	*Reporting Person:* **Dr. Shiller**
Weight: **8 pounds**	*Reported Date:* **2019-12-13**
	Last Verified Date: **2021-06-10**

- Objects reference the metadata objects that apply to them
- An objects can reference more than one metadata object

- More than one object can reference the same metadata object
- Example: `j:Metadata` (containing `j:CriminalInformationIndicator`) (SSGT/Wayfarer)

Connecting Metadata to Objects

Connections	Diagram
Easy to apply multiple metadata objects to one object	
Easy to apply same metadata to many objects	
Many-to-many relationships can get messy	

Powerful, but can be difficult to implement

Schemas

The `j:Metadata` object is of `j:MetadataType`. The `appinfo:appliesToTypes` attribute is information for tools to potentially use.

```
<xs:element name="Metadata" type="j:MetadataType" nillable="true" appinfo:appliesToTyp
es="structures:AssociationType structures:ObjectType">
    <xs:annotation>
        <xs:documentation>Information that further qualifies the kind of data represen
ted.</xs:documentation>
    </xs:annotation>
</xs:element>
```

There's nothing special about `j:MetadataType`. It's just an object holding some other objects, in this case a couple booleans indicators, `j:CriminalInformationIndicator` and `j:IntelligenceInformationIndicator`. It's based on `structures:MetadataType`, which just adds the linking infrastructure we've seen with associations and roles:

```
<xs:complexType name="MetadataType">
    <xs:annotation>
        <xs:documentation>A data type for information that further qualifies the kind
of data represented.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:MetadataType">
            <xs:sequence>
                <xs:element ref="j:CriminalInformationIndicator" minOccurs="0" maxOccu
rs="unbounded"/>
                <xs:element ref="j:IntelligenceInformationIndicator" minOccurs="0" max
Occurs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Instance Documents

```
<j:Metadata structures:id="JMD01">
    <j:CriminalInformationIndicator>true</j:CriminalInformationIndicator>
</j:Metadata>
```

```
<j:Charge structures:id="CH01" structures:metadata="JMD01">
    <j:ChargeDescriptionText>Furious Driving</j:ChargeDescriptionText>
    <j:ChargeFelonyIndicator>false</j:ChargeFelonyIndicator>
</j:Charge>
```

JSON-LD doesn't support a specific metadata link, so for JSON we just include the metadata inside the object to which it applies.

```
"j:Charge": {
```

```
    "@id": "#CH01",
    "j:ChargeDescriptionText": "Furious Driving",
    "j:ChargeFelonyIndicator": false,
    "j:CriminalInformationIndicator": true
}
```

Combining Domains (Augmentations)

- Sometimes you just want to add properties from other domains to an object without making a special kind of thing
- Augmentations are bags of stuff
- Augmentation Points are hooks onto which to hang the bags of stuff
- Augmentations are an easy way to extend objects to meet your exchange needs
- Examples:
  - `j:DriverLicense` (SSGT)
  - `j:DriverLicenseAugmentationPoint` (SSGT)

Schemas

`j:DriverLicense` is defined to be of `j:DriverLicenseType`:

```
<xs:element name="DriverLicense" type="j:DriverLicenseType" nillable="true">
    <xs:annotation>
        <xs:documentation>A license issued to a person granting driving privileges.</x
s:documentation>
    </xs:annotation>
</xs:element>
```

`j:DriverLicenseType` contains `j:DriverLicenseAugmentationPoint` as a hook for a augmentation bags:

```
<xs:complexType name="DriverLicenseType">
    <xs:annotation>
        <xs:documentation>A data type for a license issued to a person granting drivin
g privileges.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="j:DriverLicenseBaseType">
            <xs:sequence>
                <xs:element ref="j:DriverLicenseEnhancedIndicator" minOccurs="0" maxOc
curs="unbounded"/>
                <xs:element ref="j:DriverLicenseCommercialClassAbstract" minOccurs="0"
 maxOccurs="unbounded"/>
```

```
                <xs:element ref="j:DriverLicenseCommercialStatusAbstract" minOccurs="0
" maxOccurs="unbounded"/>
                <xs:element ref="j:DriverLicenseNonCommercialClassText" minOccurs="0"
maxOccurs="unbounded"/>
                <xs:element ref="j:DriverLicenseNonCommercialStatusAbstract" minOccurs
="0" maxOccurs="unbounded"/>
                <xs:element ref="j:DriverLicensePermit" minOccurs="0" maxOccurs="unbou
nded"/>
                <xs:element ref="j:DriverLicensePermitQuantity" minOccurs="0" maxOccur
s="unbounded"/>
                <xs:element ref="j:DriverLicenseWithdrawal" minOccurs="0" maxOccurs="u
nbounded"/>
                <xs:element ref="j:DriverLicenseWithdrawalPendingIndicator" minOccurs=
"0" maxOccurs="unbounded"/>
                <xs:element ref="j:DriverLicenseCardIdentification" minOccurs="0" maxO
ccurs="unbounded"/>
                <xs:element ref="j:DriverLicenseRestriction" minOccurs="0" maxOccurs="
unbounded"/>
                <xs:element ref="j:DriverLicenseEndorsement" minOccurs="0" maxOccurs="
unbounded"/>
                <xs:element ref="j:DriverLicenseAugmentationPoint" minOccurs="0" maxOc
curs="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

While NIEM does have some augmentations pre-defined, they're particularly useful for adding new objects, or just putting existing NIEM objects somewhere else. Here we do the latter, creating a bag called `LicenseAugmentation` with `nc:ContactInformation` inside. We can now "hang" it on the `j:DriverLicenseAugmentationPoint` hook:

```
<xs:complexType name="LicenseAugmentationType">
    <xs:annotation>
        <xs:documentation>
            A data type for additional information about a license.
        </xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:AugmentationType">
            <xs:sequence>
                <xs:element ref="nc:ContactInformation"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:element name="LicenseAugmentation" type="ext:LicenseAugmentationType" substitution
Group="j:DriverLicenseAugmentationPoint">
```

```
    <xs:annotation>
        <xs:documentation>
            Additional information about a license.
        </xs:documentation>
    </xs:annotation>
</xs:element>
```

Our new `ext:LicenseAugmentationType` is based on the built-in `structures:AugmentationType`. That base type merely adds in infrastructure support for linking objects together:

```
<xs:complexType name="AugmentationType" abstract="true">
    <xs:annotation>
        <xs:documentation>A data type for a set of properties to be applied to a base type.</xs:documentation>
    </xs:annotation>
    <xs:attribute ref="structures:id"/>
    <xs:attribute ref="structures:ref"/>
    <xs:attribute ref="structures:uri"/>
    <xs:anyAttribute namespace="urn:us:gov:ic:ism urn:us:gov:ic:ntk" processContents="lax"/>
</xs:complexType>
```

The resulting XML Instance Document looks like:

```
<j:DriverLicense>
    <j:DriverLicenseCardIdentification>
        <nc:IdentificationID>A1234567</nc:IdentificationID>
    </j:DriverLicenseCardIdentification>
    <ext:LicenseAugmentation>
        <nc:ContactInformation>
            <nc:ContactEmailID>peter@wimsey.org</nc:ContactEmailID>
        </nc:ContactInformation>
    </ext:LicenseAugmentation>
</j:DriverLicense>
```

The equivalent JSON-LD would be:

```
"j:DriverLicense": {
    "j:DriverLicenseCardIdentification": {
        "nc:IdentificationID": "A1234567"
    },
    "ext:LicenseAugmentation": {
        "nc:ContactInformation": {
            "nc:ContactEmailID": "peter@wimsey.org"
        }
    }
```

```
}
```

External Standards

- NIEM supports external standards, if they're XML
- Wraps them in NIEM-conformant adapters
- Example:
  - NIEM: `nc:LocationGeospatialCoordinateAbstract` (SSGT/Wayfarer)
  - NIEM-conformant adapter: `geo:LocationGeospatialPoint` (SSGT)
  - External standard: `gml:Point` (SSGT)
- You'll probably never use this, but should know it's there
- SSGT supports them; Wayfarer does not

Schemas

`LocationType` contains a `nc:LocationGeospatialCoordinateAbstract`:

```
<xs:complexType name="LocationType">
    <xs:annotation>
        <xs:documentation>A data type for geospatial location.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:ObjectType">
            <xs:sequence>
                <xs:element ref="nc:LocationAddressAbstract" minOccurs="0" maxOccurs="
unbounded"/>
                <xs:element ref="nc:LocationArea" minOccurs="0" maxOccurs="unbounded"/
>
                <xs:element ref="nc:LocationCategoryAbstract" minOccurs="0" maxOccurs=
"unbounded"/>
                <xs:element ref="nc:LocationContactInformation" minOccurs="0" maxOccur
s="unbounded"/>
                <xs:element ref="nc:LocationDescriptionText" minOccurs="0" maxOccurs="
unbounded"/>
                <xs:element ref="nc:LocationDirectionsText" minOccurs="0" maxOccurs="u
nbounded"/>
                <xs:element ref="nc:LocationGeospatialCoordinateAbstract" minOccurs="0
" maxOccurs="unbounded"/>
                <xs:element ref="nc:LocationHeightAbstract" minOccurs="0" maxOccurs="u
nbounded"/>
                <xs:element ref="nc:LocationIdentification" minOccurs="0" maxOccurs="u
nbounded"/>
                <xs:element ref="nc:LocationLandmarkAbstract" minOccurs="0" maxOccurs=
"unbounded"/>
```

```
                <xs:element ref="nc:LocationLocale" minOccurs="0" maxOccurs="unbounded
"/>
                <xs:element ref="nc:LocationMapLocation" minOccurs="0" maxOccurs="unbo
unded"/>
                <xs:element ref="nc:LocationName" minOccurs="0" maxOccurs="unbounded"/
>
                <xs:element ref="nc:LocationPart" minOccurs="0" maxOccurs="unbounded"/
>
                <xs:element ref="nc:LocationRangeDescriptionText" minOccurs="0" maxOcc
urs="unbounded"/>
                <xs:element ref="nc:LocationRelativeLocation" minOccurs="0" maxOccurs=
"unbounded"/>
                <xs:element ref="nc:LocationSurroundingAreaDescriptionText" minOccurs=
"0" maxOccurs="unbounded"/>
                <xs:element ref="nc:LocationAugmentationPoint" minOccurs="0" maxOccurs
="unbounded"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

`nc:LocationGeospatialCoordinateAbstract` is the head of a substitution group:

```
<xs:element name="LocationGeospatialCoordinateAbstract" abstract="true">
    <xs:annotation>
        <xs:documentation>A data concept for a geospatial location.</xs:documentation>
    </xs:annotation>
</xs:element>
```

One of the members of that substitution group is `geo:LocationGeospatialPoint`, which is an adapter to the external standard:

```
<xs:element name="LocationGeospatialPoint" type="geo:PointType" substitutionGroup="nc:
LocationGeospatialCoordinateAbstract" nillable="true">
    <xs:annotation>
        <xs:documentation>A 2D or 3D geometric point.   A gml:Point is defined by a si
ngle coordinate tuple. The direct position of a point is specified by the gml:pos elem
ent which is of type gml:DirectPositionType.</xs:documentation>
    </xs:annotation>
</xs:element>
```

Its type is `geo:PointType`. We're still in NIEM, but `gml:Point` isn't:

```
<xs:complexType name="PointType" appinfo:externalAdapterTypeIndicator="true">
    <xs:annotation>
        <xs:documentation>A data type for a 2D or 3D geometric point.   A gml:Point is
 defined by a single coordinate tuple. The direct position of a point is specified by
```

```
the gml:pos element which is of type gml:DirectPositionType.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:ObjectType">
            <xs:sequence>
                <xs:element ref="gml:Point" minOccurs="1" maxOccurs="1">
                    <xs:annotation>
                        <xs:documentation>A gml:Point is defined by a single coordinat
e tuple. The direct position of a point is specified by the gml:pos element which is o
f type gml:DirectPositionType.</xs:documentation>
                    </xs:annotation>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

And here is the `gml:Point` object from the external standard.

```
<element name="Point" type="gml:PointType" substitutionGroup="gml:AbstractGeometricPri
mitive">
    <annotation>
        <documentation>A Point is defined by a single coordinate tuple. The direct pos
ition of a point is specified by the pos element which is of type DirectPositionType.<
/documentation>
    </annotation>
</element>

<complexType name="PointType">
    <complexContent>
        <extension base="gml:AbstractGeometricPrimitiveType">
            <sequence>
                <choice>
                    <element ref="gml:pos"/>
                    <element ref="gml:coordinates"/>
                </choice>
            </sequence>
        </extension>
    </complexContent>
</complexType>
```

The resulting XML instance looks like this:

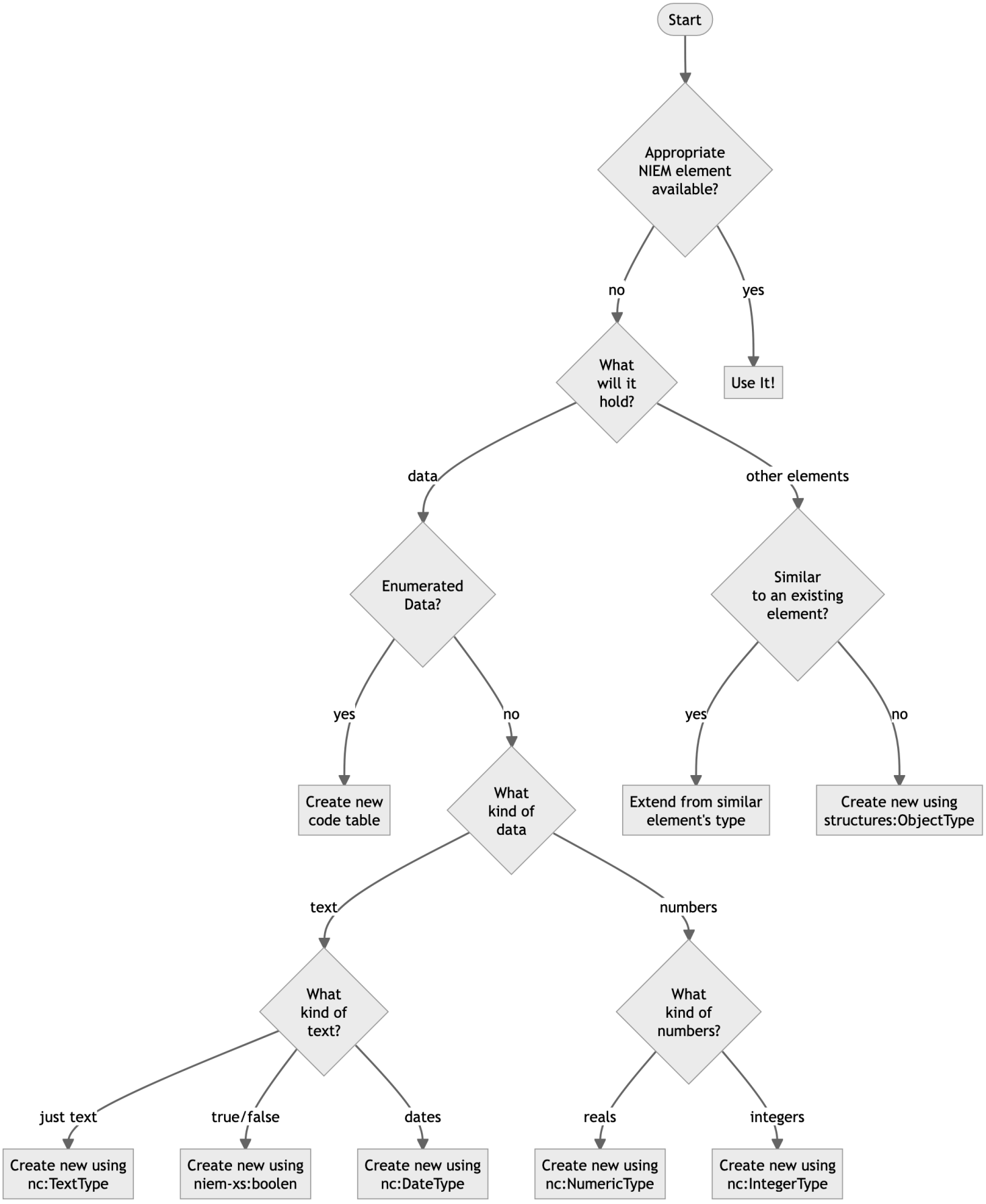
```
<nc:Location>
    <geo:LocationGeospatialPoint>
        <gml:Point gml:id="point1">
            <gml:pos srsName="urn:ogc:def:crs:EPSG::4326" srsDimension="2">40.689167 -
74.044444</gml:pos>
```

```
        </gml:Point>
    </geo:LocationGeospatialPoint>
</nc:Location>
```

It doesn't make much sense to try and represent this in JSON in a NIEM-conformant way. NIEM doesn't (can't) provide rules for converting non-NIEM XML to JSON. This was auto-generated with an XML editor:

```
"nc:Location": {
    "geo:LocationGeospatialPoint": {
        "gml:Point": {
            "gml:id": "point1",
            "gml:pos": {
                "srsName": "urn:ogc:def:crs:EPSG::4326",
                "srsDimension": 2,
                "#text": "40.689167 -74.044444"
            }
        }
    }
}
```

Creating New Objects


Start
Appropriate NIEM element available?
no
yes
Use It!
What will it hold?
data
other elements
Enumerated Data?
Similar to an existing element?
yes
no
yes
no
Create new code table
What kind of data
Extend from similar element's type
Create new using structures:ObjectType
text
numbers
What kind of text?
What kind of numbers?
just text
true/false
dates
reals
integers
Create new using nc:TextType
Create new using niem-xs:boolen
Create new using nc:DateType
Create new using nc:NumericType
Create new using nc:IntegerType

Creating Simple Data Elements

- Base them on whichever XML Schema data type is appropriate
  - They're all in the niem-xs schema
- Follow the flowchart for help
- Base on the `niem-xs` adapters if you need referencing
  - Folks generally do this anyway, for consistency
- Follow the naming format of existing NIEM elements of that type, especially regarding the last term, e.g. "Text", "Code", etc.

Create `PersonDefenestrationIndicator` using the existing NIEM type `niem-xs:boolean`. By giving it `j:CrashPersonAugmentationPoint` as a substitution group head, it can go inside of the `j:CrashPerson`:

```
<xs:element name="PersonDefenestrationIndicator" type="niem-xs:boolean"
    substitutionGroup="j:CrashPersonAugmentationPoint">
    <xs:annotation>
        <xs:documentation>True if this person was thrown through a window; false other
wise.
        </xs:documentation>
    </xs:annotation>
</xs:element>
```

The resulting XML instance is then:

```
<j:CrashPerson>
    <nc:RoleOfPerson structures:ref="P01" xsi:nil="true"/>
    <j:CrashPersonInjury structures:metadata="PMD01 PMD02">
        <nc:InjuryDescriptionText>Broken Arm</nc:InjuryDescriptionText>
        <j:InjurySeverityCode>3</j:InjurySeverityCode>
    </j:CrashPersonInjury>
    <ext:PersonDefenestrationIndicator>false</ext:PersonDefenestrationIndicator>
</j:CrashPerson>
```

And the equivalent JSON-LD is:

```
"j:CrashPerson": {
    "nc:RoleOfPerson": {
      "@id": "#P01"
    },
    "j:CrashPersonInjury": {
      "nc:InjuryDescriptionText": "Broken Arm",
      "j:InjurySeverityCode": "3",
    },
    "ext:PersonDefenestrationIndicator": "false"
```

```
}
```

Note the `@id` that links to an identical `@id` in the nc:Person object.

For a more complicated example, we can create the `ext:PrivacyCode` element along with a `ext:PrivacyMetadata` to hold it.

The actual codes are defined in the simple type, `ext:PrivacyCodeSimpleType`. Each `enumeration` defines a code. The `documentation` tags provide a longer text description of the code:

```
<xs:simpleType name="PrivacyCodeSimpleType">
    <xs:annotation>
        <xs:documentation>A data type for a code representing a kind of
            property.</xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:token">
        <xs:enumeration value="PII">
            <xs:annotation>
                <xs:documentation>Personally Identifiable Information</xs:documentatio
n>
            </xs:annotation>
        </xs:enumeration>
        <xs:enumeration value="MEDICAL">
            <xs:annotation>
                <xs:documentation>Medical Information</xs:documentation>
            </xs:annotation>
        </xs:enumeration>
    </xs:restriction>
</xs:simpleType>
```

The complex type `ext:PrivacyCodeType` is then extended from `ext:PrivacyCodeSimpleType`. All this does is add infrastructure attributes to allow things of this type to refer to other elements, or be referred to in turn:

```
<xs:complexType name="PrivacyCodeType">
    <xs:annotation>
        <xs:documentation>A data type for a code representing a kind of
            property.</xs:documentation>
    </xs:annotation>
    <xs:simpleContent>
        <xs:extension base="ext:PrivacyCodeSimpleType">
            <xs:attributeGroup ref="structures:SimpleObjectAttributeGroup"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

Then `ext:PrivacyCode` is created to be of `ext:PrivacyCodeType`:

```
<xs:element name="PrivacyCode" type="ext:PrivacyCodeType">
    <xs:annotation>
        <xs:documentation>A code representing a kind of property.</xs:documentation>
    </xs:annotation>
</xs:element>
```

In this exchange, this code is metadata to be applied to other objects, so we can create `ext:PrivacyMetadata` to hold it:

```
<xs:element name="PrivacyMetadata" type="ext:PrivacyMetadataType">
    <xs:annotation>
        <xs:documentation>Metadata about Privacy.</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:complexType name="PrivacyMetadataType">
    <xs:annotation>
        <xs:documentation>A data type for metadata about Privacy.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:MetadataType">
            <xs:sequence>
                <xs:element ref="ext:PrivacyCode" minOccurs="0" maxOccurs="unbounded"/
>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Instance Documents

The resulting XML instance document is:

```
<ext:PrivacyMetadata structures:id="PMD01">
    <ext:PrivacyCode>PII</ext:PrivacyCode>
    <ext:PrivacyCode>MEDICAL</ext:PrivacyCode>
</ext:PrivacyMetadata>
```

An equivalent JSON document would be:

Instead of linking to separate metadata objects, we've embedded those into the `j:CrashPeson`.

```
"j:CrashPerson": {
```

```
    "nc:RoleOfPerson": {
      "@id": "#P01"
    },
    "j:CrashPersonInjury": {
      "nc:InjuryDescriptionText": "Broken Arm",
      "j:InjurySeverityCode": "3",
      "ext:PrivacyCode": [
        "PII", "MEDICAL"
      ]
    },
    "ext:PersonDefenestrationIndicator": "false"
}
```

Creating Complex Objects

- If you're making a special kind of existing NIEM type:
  - Use that type as a base to extend from and add things to it, or
  - Create an Augmentation to hold your new things
- If you're starting from scratch:
  - Use `structures:ObjectType` as a base to extend from and add things to it

Here we're making the root element that will hold everything else. We create `CrashDriverInfoType`, basing it on `structures:ObjectType` so that it's just an empty object.

To that empty object, we add all the major objects in our exchange, `nc:Person`, `j:Crash`, and `j:Charge`. We also add a `j:PersonChargeAssociation` which lets us link together people and charges. Finally, we add a couple metadata object, the built-in `j:Metadata`, and `ext:PrivacyMetadata`, which we create in the extension schema and is in the prior example.

```
<xs:element name="CrashDriverInfo" type="exch:CrashDriverInfoType">
    <xs:annotation>
        <xs:documentation>A collection of legal charges associated with the driver of
a vehicle in a crash.</xs:documentation>
    </xs:annotation>
</xs:element>

<xs:complexType name="CrashDriverInfoType">
    <xs:annotation>
        <xs:documentation>A data type for a collection of legal charges associated wit
h the driver of a vehicle in a crash.</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="structures:ObjectType">
            <xs:sequence>
                <xs:element ref="nc:Person" maxOccurs="unbounded"/>
                <xs:element ref="j:Crash"/>
```

```
                <xs:element ref="j:PersonChargeAssociation" minOccurs="0" maxOccurs="u
nbounded"/>
                <xs:element ref="j:Charge" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="j:Metadata" minOccurs="0"/>
                <xs:element ref="ext:PrivacyMetadata" minOccurs="0" maxOccurs="unbound
ed"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Another example is when we created `ext:LicenseAugmentation` and `ext:LicenseAugmentationType` when talking about augmentations.

Adding New Content to the Exchange

To summarize, there are two major ways to add new content to an exchange. We've seen them both above.

If you're *already* creating a new complex object and type, like `CrashDriverInfo` and `CrashDriverInfoType` above, you can simply add new elements to the new type, as we did with `ext:PrivacyMetadata`. This is called "concrete extension."

But if you're not already creating the new type for other reasons, you should use augmentations. We used added `j:CrashPersonAugmentationPoint` to our exchange and used it as a hook on which we hung `ext:PersonDefenestrationIndicator`.

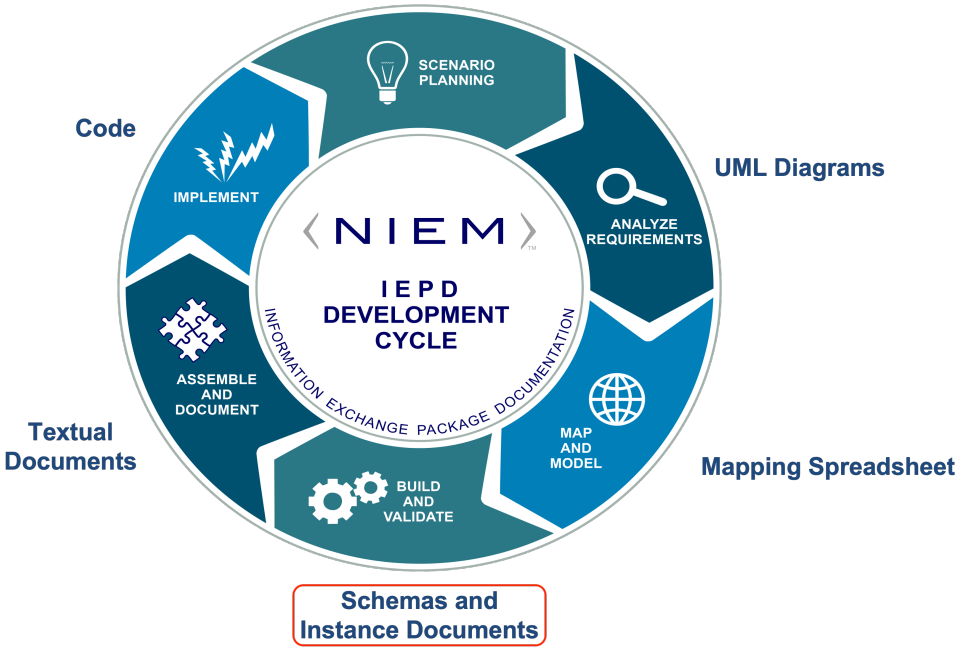
Problems with Concrete Extensions

The issue with concrete extensions is that if the extension is happening deep inside an object, all the surrounding objects will also need to be extended. For example, if instead of using `j:DriverLicenseAugmentationPoint`, suppose we extended `j:DriverLicenseType`, making a new `ext:DriverLicenseType` to hold our new object. Now we need a new element for it, `ext:DriverLicense`.

But `j:CrashDriver` doesn't hold an `ext:DriverLicense`, so we need to make a new `ext:CrashDriverType` and `ext:CrashDriver` that can hold an `ext:DriverLicense`.

But `j:Crash` doesn't hold an `ext:CrashDriver`, so we need to make a new `ext:CrashType` and `ext:Crash` that can hold `ext:CrashDriver`.

That's a lot of extra work and muddies the semantics of the elements.



Creating and Validating Schemas

- Conformance
- Subsets
- Extension and Exchange Schemas
- How things fit together

NIEM Conformance

- Follow the rules in the Naming and Design Rules (NDR)
- Follow the rules in the IEPD Spec
- Many NDR rules exist as Schematron
  - Can check these via the Conformance Testing Assistant (ConTesA)
  - Can run the Schematron directly oXygen, with a plug-in in XMLSpy

Schema Subsets

- NIEM has ~13,000 elements
- You don't want them all
- NIEM supports mini versions of NIEM
  - With the elements / types you want
  - Plus things needed to make the elements / types you want work
- Use the SSGT for this! **Demo Time!**

Extension Schema(s)

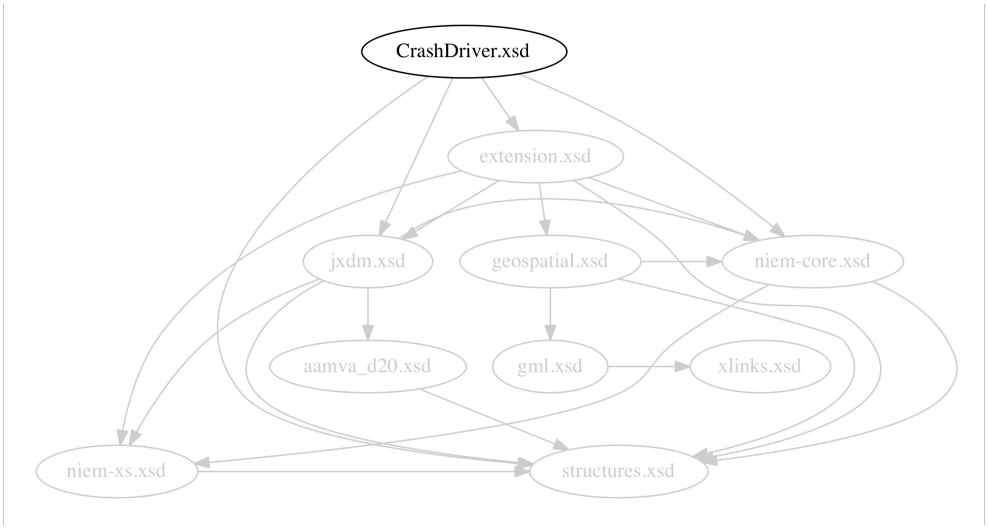
- Create new elements for your exchange
- Emulate how NIEM does it
- Utilize Augmentations to add your new objects to existing NIEM objects
    - Concrete extension is an alternative
- Can have multiple extension schemas
- Some folks put code tables into a separate extension schema

Exchange Schema

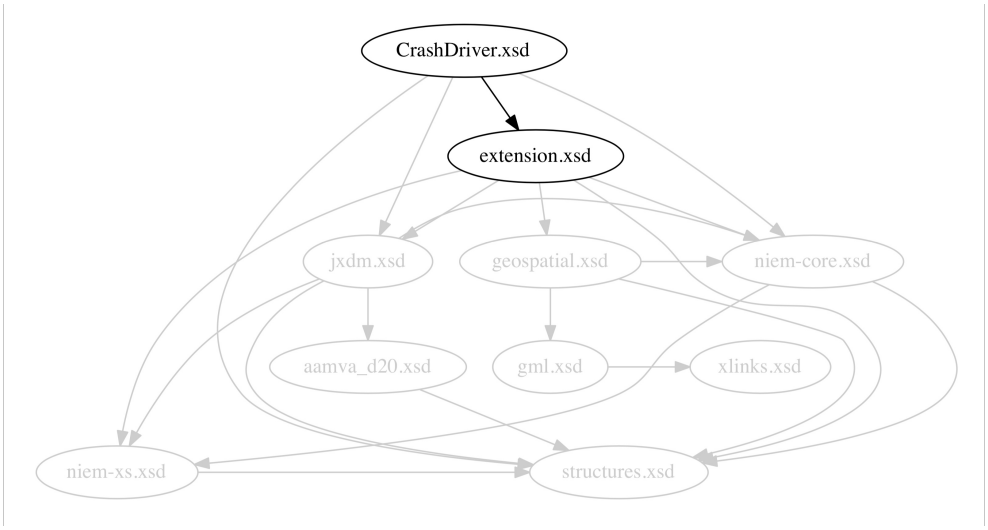
- Entry point to the exchange
- Defines the root element of the exchange
- Some put the definition for the root element here
- Some put that in the extension schema
- Some lump exchange and extension together
- An exchange with multiple messages can have multiple exchange schemas, one per
    - Can share a common extension
    - Or not

How Schemas Fit Together

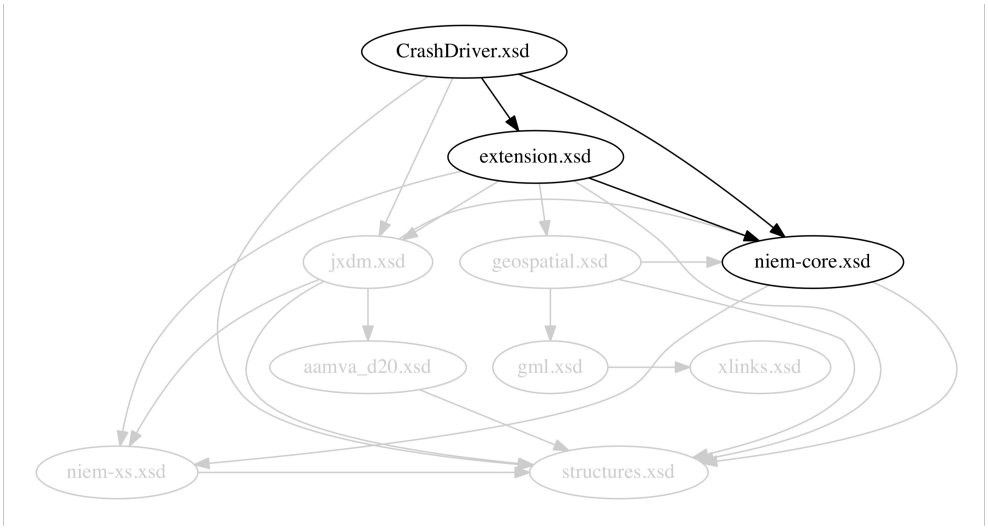
Step 1: Exchange Schema



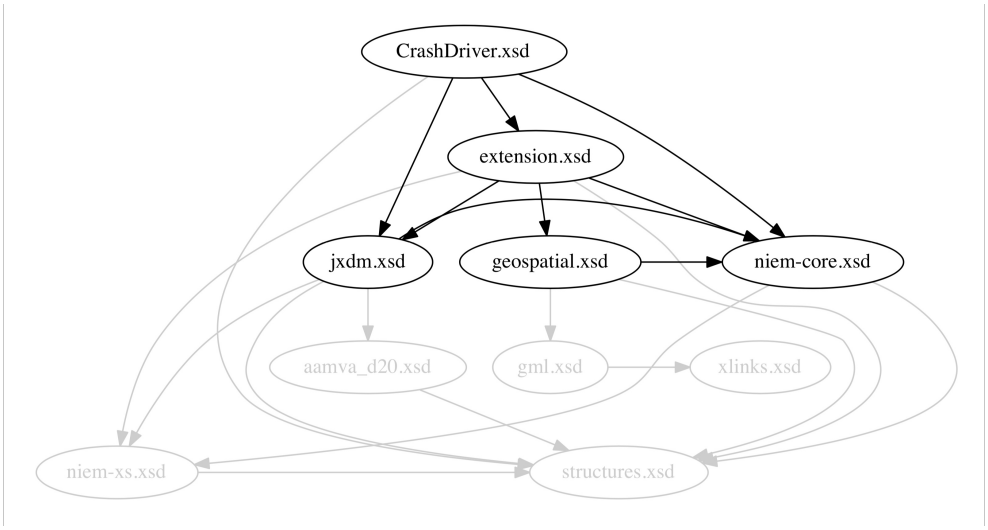
Step 2: Extension Schema



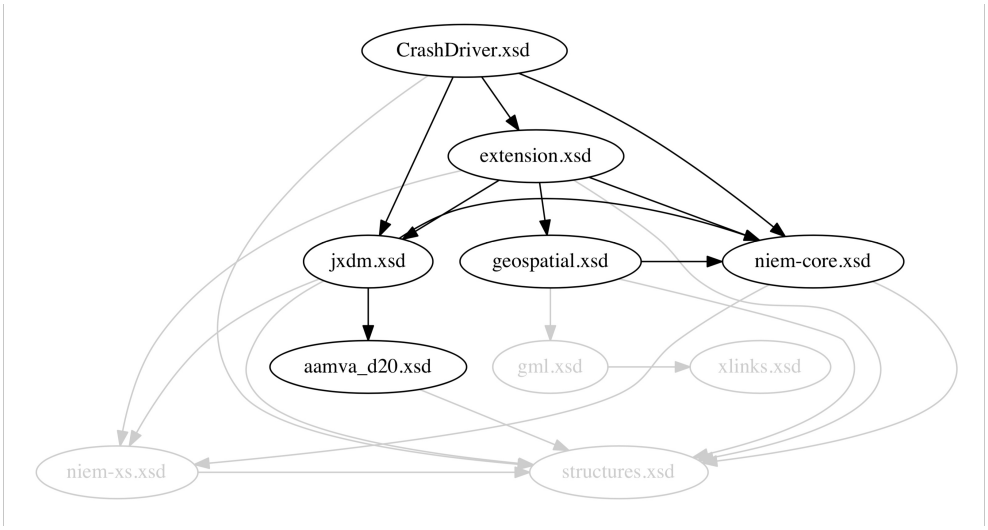
Step 3: NIEM Core



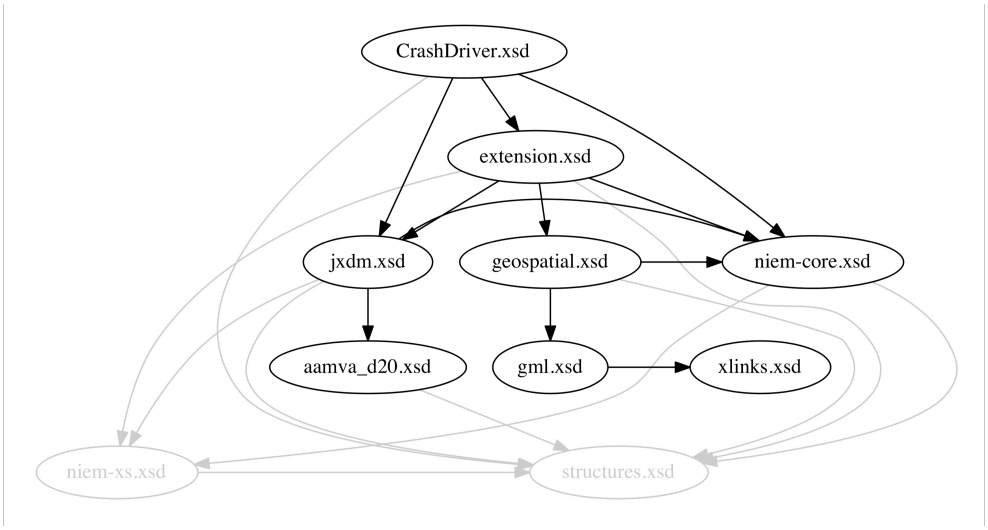
Step 4: Domains



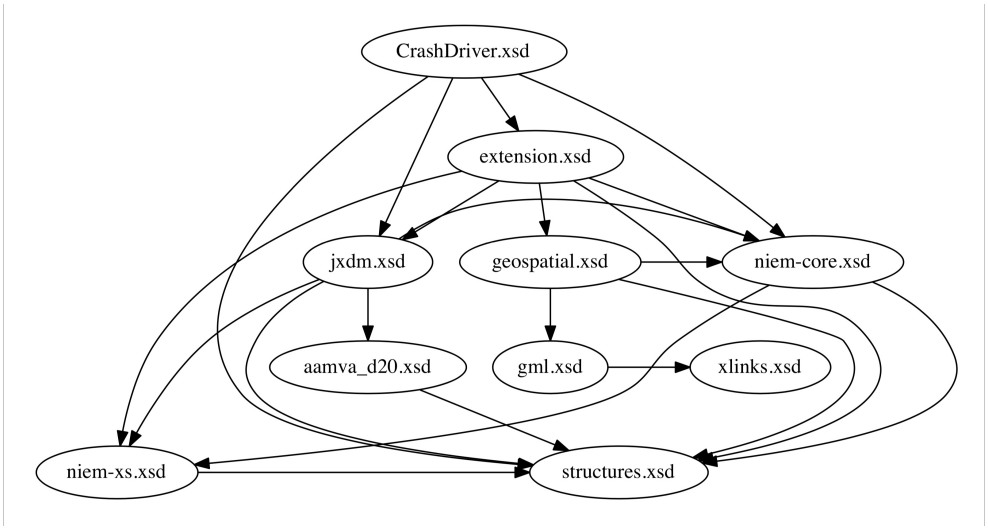
Step 5: Code Tables



Step 6: External Standards



Step 7: Infrastructure



Help with Schemas

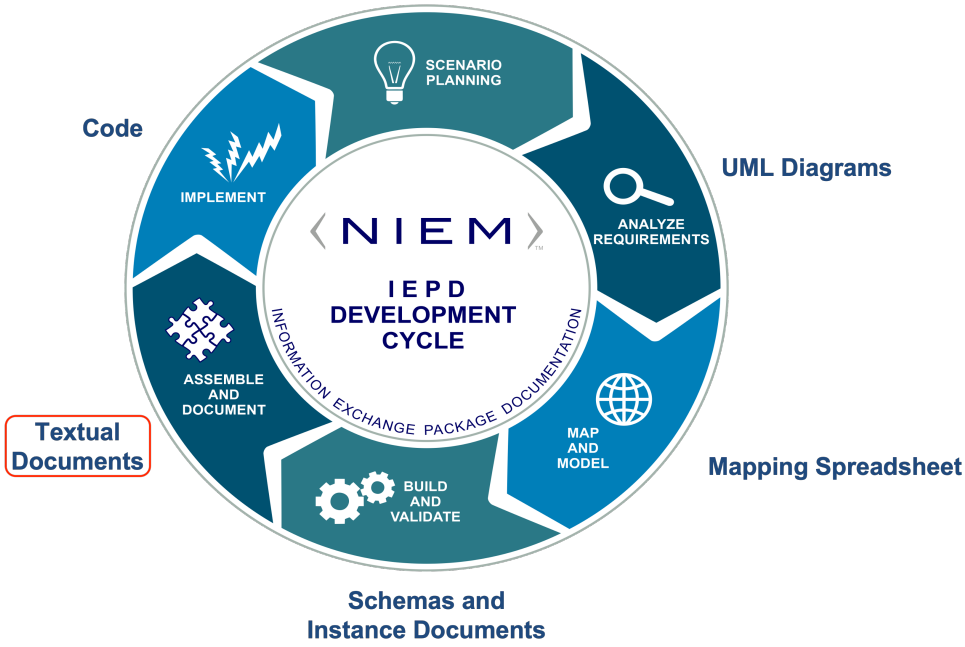
- Use a good XML editor
  - XMLSpy
    - Windows
  - oXygen
    - Windows, macOS, Linux
- Declare a conformance target
  - Just use the same one I am
- Declare and import all schemas used
  - Watch out for schemas incorporated only via substitution groups
  - Group heads don't know who the members are
  - Still need to declare and import those

Example Instances

- Required
- Validating against your schemas ensures your intent
- Some XML editors can create them from schemas
  - But you'll always need to tweak those
- JSON instances are more of a manual process because NIEM doesn't support JSON Schema
  - It's in the works
- Other tools are out there…

Tips and Tricks

- Christina's oXygen Snippets
  - oXygen only
- Use Schematron to check your work as you go
  - oXygen
  - XMLSpy with plug-in
- Tom's BBEdit instance generation scripts
- Tom's Mapping Spreadsheet Linter



Other Artifacts

- IEPD Catalog
  - Metadata about the IEPD
  - What file is what
- Readme
- Changelog
- Conformance Assertion
- Mapping Spreadsheet
- Master Documentation

Master Documentation

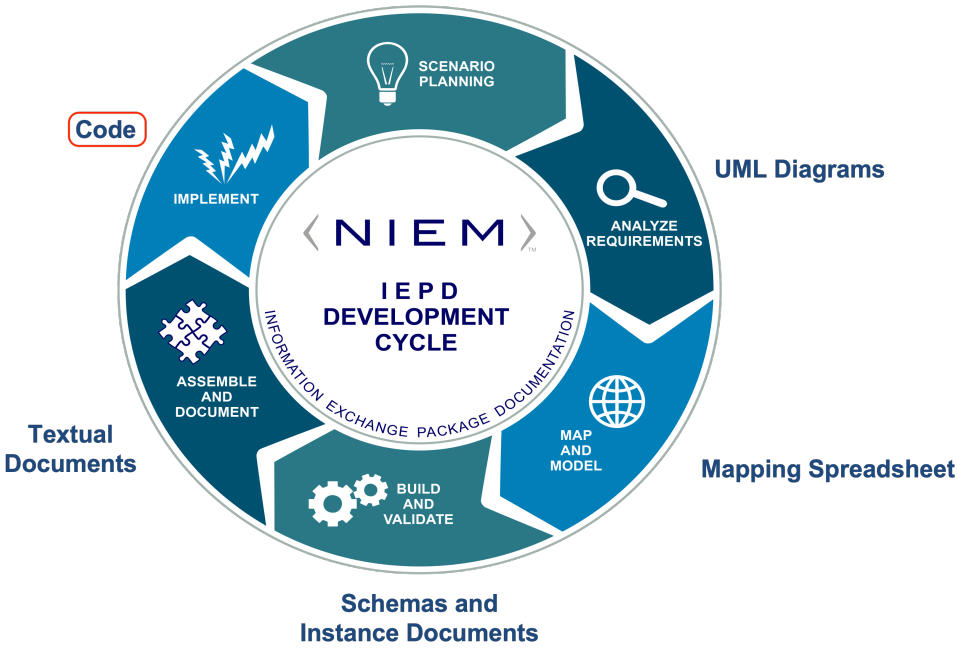
- Just a Word document
  - Or another document format, e.g. Markdown, PDF, etc.
- A NIEM IEPD master document should (at a minimum) describe:
  - IEPD purpose
  - scope
  - business value
  - exchange information
  - senders/receivers
  - interactions
  - references to other documentation, if applicable
- Don’t paste schemas into them!

Message Spec / IEPD Assembly

- IEPDs are just ZIP files full of artifacts
- Can organize how you like
  - The IEPD Catalog is the guide to your organization
  - Generally, keep it simple
- MEP Builder will help with this (coming soon)

Publishing

- Existing repositories are out of date
  - IEPD Clearinghouse
  - Work With IEPDs (currently not working)
- New repositories are coming
  - Restricted (coming soon)
  - Unrestricted (coming later)



Implementation

- Remember that NIEM is just the payload
  - It's just plain old XML
    - Well, okay, with referencing
  - Or it's just plain JSON
    - Well, okay, with JSON-LD
- Remember the scope of NIEM
  - Things outside the payload aren't NIEM
  - You don't have to do anything special for NIEM outside NIEM's scope

Skill level to Implement NIEM

- Developers with the skills to design and implement a data exchange can learn the NIEM approach in a matter of hours
- Developer training is available
  - No-cost online courses are on the NIEM website
- Plenty of example IEPDs to follow
- Don't start from scratch, leverage shared IEPDs
- The NIEM technical specifications are complex, however
  - Most developers do not need to read them
  - Free tools can perform most of the conformance checking

Exercises

Understanding NIEM Objects

Let's think about cardinality a little:

1. What happens if `maxOccurs` is less than `minOccurs`? (You can try this out in an editor, or just think about what *should* happen.)
2. What do you think happens if `maxOccurs` is set to zero? Why might you want to do that?
3. What are some cases where you might want a `maxOccurs` that is greater than 1, but not unbounded?

Native Properties

This is an open-ended exercise. There's no answer. The idea is to get a little familiar with the tools and some of the major NIEM objects. Try using both the SSGT and Wayfarer:

1. Search for, then look around at some common NIEM elements:
    - `nc:Activity`
    - `nc:Identification`
    - `nc:Location`
    - `nc:Organization`
    - `nc:Person`

Substitution Groups

1. What are all the different ways that NIEM can represent a date?
2. Take a look at the following elements and how they fit together via substitution groups. What do you think the reason for all this might be?
    - nc:Entity (SSGT/Wayfarer)
    - nc:EntityRepresentation (SSGT/Wayfarer)
    - nc:EntityOrganization (SSGT/Wayfarer)
    - nc:EntityPerson (SSGT/Wayfarer)

Inherited Properties

Find out some information about commercial vehicles by finding the properties that answer these questions:

1. How could you indicate a vehicle is a commercial vehicle? Can you find a second way?
2. How could you describe the brand name of a commercial vehicle as text rather than a code?
3. How could you indicate the primary color of a commercial vehicle? Can you find a second way? A third? A fourth?

Associations

1. How might you associate a person with their aunt or uncle? (Hint: "Facet" is the term the SSGT uses for code table codes. Play with the search options there to find this. Wayfarer won't be a help with this.)
2. Write up a little XML or JSON, or even YAML, showing this object.

Roles

1. How many kinds of roles can an “item” play? What are they? (Hint: “Kinds of roles” refers to types.)
2. How many different elements can represent these kinds of roles? What are they? (Hint: Wayfarer is the best tool for this.)

Code Tables

1. Going back to the primary color of a commercial vehicle, is there any difference between the different code table options?
2. Why might there be all these different code tables for this? (You won’t be expected to actually know this, but it’s good to ponder it.)
3. Find all the values for a person’s eye color.
4. What if you wanted to describe the eye color as “The color of my favorite pair of faded blue jeans”?
5. What do you think happens if a code table has no enumerations defined? What will validate and what won’t?

Metadata

1. What does the Human Services domain count as metadata?
2. What about the Screening domain?
3. Think up an example of metadata about this class.

Combining Domains - Augmentations

1. Can you mix and match NIEM elements into your own Augmentation?

Creating New Objects - Simple Data

1. Write up XML Schema (sorry JSON folks) to create a pair of elements to rate this class on a scale of 1-5 and provide a comment.
2. Write up the XML or JSON that matches.

Creating New Objects - Complex Objects

1. Write up XML Schema to create an evaluation object, holding the pair of elements from the prior exercise.
2. Write up the XML or JSON that matches.

Resources

Documentation

- NIEM Releases
  - http://niem.github.io/niem-releases/
- NIEM Specifications
  - http://niem.github.io/reference/specifications/
- NIEM JSON Spec
  - https://niem.github.io/NIEM-JSON-Spec/v5.0/
- Materials from this course:
  - https://github.com/NIEM/NIEM-Training
  - Some of these are internal work products
  - `README` has links to the good stuff

Tools

- SSGT:
  - https://tools.niem.gov/niemtools/ssgt/index.iepd
- Conformance Testing Assistant (ConTesA)
  - https://tools.niem.gov/contesa/
- NDR Schematron
  - https://niem.github.io/reference/tools/oxygen/ndr/
- MEP Builder (coming soon)
- Wayfarer:
  - http://niem5.org/wayfarer/
- oXygen Snippets
  - https://niem.github.io/reference/tools/oxygen/snippets/
- Mapping Spreadsheet Linter / Schema Generator
  - http://niem5.org/linter/
- Instance generation scripts (coming soon)

Repositories

- IEPD Clearinghouse
  - https://bja.ojp.gov/program/it/policy-implementation/clearinghouse
- IEPD Repository (Work with IEPDs) - *currently inoperative*
  - https://tools.niem.gov/niemtools/iepdt/index.iepd