Creating a NIEM 2.1 IEPD

Priscilla Walmsley <pwalmsley@datypic.com>

May 18, 2010

Abstract

NIEM is rapidly becoming the most important XML exchange standard for the U.S. government and its information partners. This article provides an overview of the process of defining a NIEM information exchange (IEPD). It then takes you through the steps required to create the IEPD. A simple case study is used to illustrate the process.

Table of Contents

Introducing NIEM

The National Information Exchange Model (NIEM) is a U.S. government-sponsored initiative to facilitate information sharing among public and private sector organizations. Its initial focus was on law enforcement, public safety, and emergency management, but it is continuously being expanded into other domains. New XML initiatives within the U.S. Department of Justice and Department of Homeland Security, along with other sectors of the U.S. government, use NIEM as a common base data model and methodology to promote interoperability of data and software, reduce design and development time for information exchange applications, and allow the reuse of intellectual capital and skills across multiple projects.

NIEM is described as a *framework*, because it is not just an XML vocabulary for information exchange. It has several components:

- A common XML-based data model called *NIEM core* that provides data components for describing universal objects such as people, locations, activities, and organizations
- More specialized XML data models for individual use cases, called domains (examples include Justice, Immigration, and Emergency Management)
- A methodology for using and extending the building blocks that come from the common and domain-specific models to turn them into a complete information exchange, known as an *information exchange package*
- Tools to help develop, validate, document, and share the information exchange packages
- A governance organization that provides training and support and oversees NIEM's evolution over time

How do you use NIEM?

The NIEM XML data model provides building blocks for common objects. A building block may be at a very granular level, such as "person name" or "birth date," or a much more complex component, such as "arrest" or "court case." However, the NIEM model itself doesn't define complete information exchange messages such as "Arrest Report" or "Suspicious Activity Report." It does not designate any specific message types or root elements of XML documents.

To actually use NIEM, you need to build an IEPD. The IEPD pulls the necessary components from the NIEM core and domain models and extends them to create an information exchange. An IEPD contains several artifacts:

- XML schemas that define the subset of the NIEM model used in this exchange, known as the *subset schema*
- A schema that defines the root element of the exchange, known as the *exchange schema*
- A schema that defines extensions to the NIEM model, known as the *extension schema*
- Documentation of the exchange, such as UML diagrams, narrative descriptions, and samples

Developing an IEPD

The first task in any information exchange project is to gather and analyze your requirements. NIEM does not require any particular method of defining requirements, so this article doesn't describe this process. In fact, this article assumes that before you sit down to actually create your IEPD, you have an idea of the data elements you want to exchange and the type of messages you want to structure them into.

This article will work through a simple example from start to finish, with the result being a complete IEPD. The example case study will be to implement a simple theft report that covers registered vehicles. Hypothetically, local law enforcement would use the theft report to inform interested parties, such as the Division of Motor Vehicles or the City Bicycle Registration Bureau, of thefts of motor vehicles and bicycles. During my requirements gathering phase, I have gathered general information on the data that needs to be shared and determined that only one type of message is required: the theft report. In reality, an IEPD often consists of multiple related message types.

Because a main goal of NIEM is data interoperability, it makes sense to consider reusing an existing IEPD before creating a new one from scratch. NIEM provides a Shared IEPDs site that allows you to search for existing IEPDs submitted by other organizations.

If you can't find an existing IEPD that suits your needs, you will need to build one. Developing a new NIEM IEPD requires five steps:

1. Model your exchange.
2. Map your exchange to the NIEM data model.
3. Create a subset of the NIEM model for use in your exchange.
4. Create exchange and extension schemas to describe your custom components.
5. Assemble an IEPD with all of the appropriate artifacts.

This article describes these five steps, each in a separate section. Even if you choose to reuse an existing IEPD, this article might still be a useful guide to help you to understand the content and structure of the IEPDs you are using.

Step 1: Model your NIEM exchange

Understanding the NIEM model

Before you create a model for your exchange, it is useful to understand how the NIEM data model is structured. NIEM defines concepts—such as types, properties, and associations—that are probably familiar to you from other data modeling paradigms.

NIEM model concepts

Types represent things, both tangible and intangible. Some of the most fundamental types in the NIEM model are `PersonType`, `ActivityType`, `ItemType`, `LocationType`, and `OrganizationType`. There are also thousands more, with varying degrees of granularity. Types might be known as *classes* or *entities* in other modeling paradigms.

Properties are attributes of types. They can themselves have complex types. For example, `PersonName` is a property of `PersonType`, but it is also has a type `PersonNameType` that has its own structure containing `PersonGivenName`, `PersonSurName`, and so on.

Types can be derived from other types and inherit their properties, which is analogous to generalizations in an object-oriented model. For example, `ItemType` is a generic type that has many types derived from it, including `VehicleType`, `JewelryType` and `RealEstateType`.

Associations are relationships between two types. You might have an association between an `Incident` and a `Person`, or a `Person` and a `Location`. Associations in NIEM are separate from the types they relate.

Roles represent temporary affiliations that a type might have in a particular context. For example, in a theft incident, a person might play the role of either `Victim`, `Subject`, or `Witness`.

Augmentations are bundles of properties that you can reuse and share. These are more commonly used in the NIEM domain models than in your IEPDs.

Metadata is information about the content of a message, such as when the information was gathered and how reliable it is. NIEM makes special provisions in its model for relating data to metadata.

The NIEM model in XML

The NIEM model is implemented entirely as a set of W3C XML Schema documents. Annotations and references within the XML schemas are used to indicate whether something is a type, an association, and so on. Fortunately, you do not have to read the XML schema documents themselves to navigate the model; NIEM provides tools to search and navigate the model in a more graphical fashion.

In general, NIEM types are implemented as XML Schema complex types, and properties are elements contained within those types. Example 1 shows how an activity is represented by an `ActivityType` complex type, with properties such as `ActivityIdentification` and `ActivityCategoryText` implemented as child elements.

Example 1. Partial NIEM `ActivityType` implementation in XML Schema

```
<xsd:complexType name="ActivityType">
 <xsd:complexContent>
  <xsd:extension base="s:ComplexObjectType">
   <xsd:sequence>
    <xsd:element ref="nc:ActivityIdentification" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="nc:ActivityCategoryText" minOccurs="0" maxOccurs="unbounded"/>
    <!-- ... -->
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

NIEM uses XML Schema extensions for type derivation. Example 2 shows how a more specific kind of activity—the `AssessmentType` complex type—is derived from `ActivityType`.

Example 2. NIEM type derivation in XML Schema

```
<xsd:complexType name="AssessmentType">
 <xsd:complexContent>
  <xsd:extension base="nc:ActivityType">
   <xsd:sequence>
    <xsd:element ref="nc:AssessmentScoreText" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="nc:AssessmentFee" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="nc:AssessmentProgram" minOccurs="0" maxOccurs="unbounded"/>
    <!-- ... -->
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Associations are special kinds of complex types that contain references to the types they associate. Example 3 shows how an association between a person and an activity—`ActivityPersonAssociationType`—is implemented. All association types are extensions (directly or indirectly) of the NIEM core `AssociationType`.

Example 3. NIEM association type in XML Schema

```
<xsd:complexType name="ActivityPersonAssociationType">
 <xsd:complexContent>
  <xsd:extension base="nc:AssociationType">
   <xsd:sequence>
    <xsd:element ref="nc:ActivityReference" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="nc:PersonReference" minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Modeling your exchange

NIEM does not require that you use any particular methodology or diagram types to model your XML exchange or even that you model it at all. However, modeling is an important step in IEPD design. The modeling process fleshes out the requirements, and the final result provides documentation that is helpful for both business and technical users. The model also serves as useful input into the subsequent steps in the IEPD creation process.

Choosing a modeling paradigm

A good option is UML—in particular, UML class diagrams—because UML concepts map easily onto NIEM model concepts. Of course, you can create other UML diagrams, such as use case diagrams and sequence diagrams, to document your exchange. This article focuses on the class diagram, because that is most crucial to the IEPD development process.

If you use UML, there is an advantage to using an XMI-enabled UML tool, such as IBM Rational Modeler or ArgoUML. This is because you can use the XMI to automatically generate a mapping spreadsheet, which you can use in the next step. For this article, I used ArgoUML, an open source UML editor.

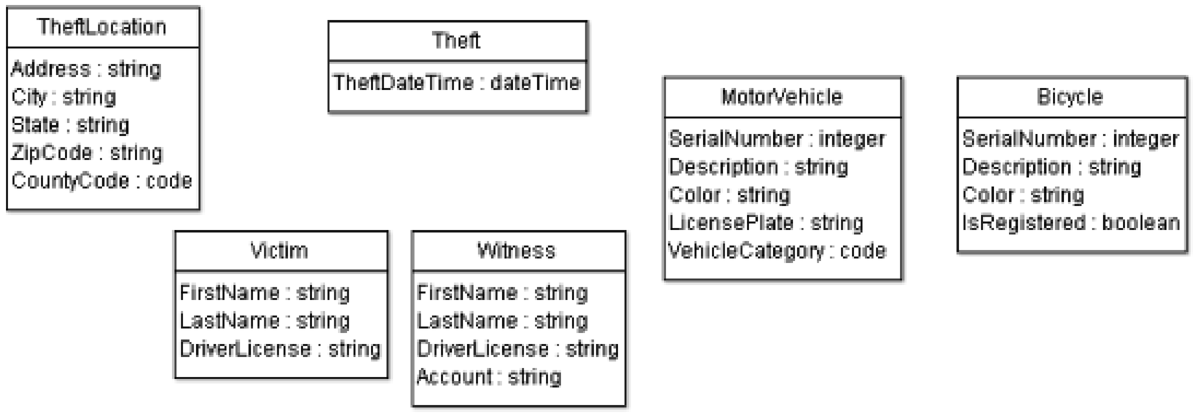
It is best to create a first draft of your model independently without trying to fit it into the NIEM model. You want to get the model right for your business needs without being unduly influenced by the NIEM way of doing things. Later in the IEPD process, it not uncommon to make small alterations to the model to better harmonize it with NIEM (in cases where it makes sense). However, there will always be differences between your model and the NIEM model.

Types and properties

This article isn't long enough to provide a complete introduction to UML modeling, so it focuses on the NIEM-specific pointers. As you might expect, NIEM types are represented by classes in a class diagram. Properties are represented by attributes of the class.

In my example case study, I determine that I have several classes that need to be exchanged—for example `Theft`, `MotorVehicle`, `Bicycle`, `Victim`, `Witness`, and `TheftLocation`. These types are depicted in Figure 1, along with their properties.

Figure 1. Initial UML model with types and properties



When specifying the data types of the properties, it is useful to use XML Schema primitive data types, because the properties will eventually be represented in an XML schema and it will be easier to determine whether the existing NIEM model fits yours if you use a common set of data types. The most commonly used XML Schema data types are listed in Table 1.

Table 1. Common XML Schema data types

Data type name	Description	Example(s)
`string`	Any text string	abc, this is a string
`integer`	An integer of any size	1, 2
`decimal`	A decimal number	1.2, 5.0
`date`	A date, in `YYYY-MM-DD` format	2009-12-25
`time`	A time, in `HH:MM:ss` format	12:05:04
`boolean`	A true/false value	true, false

Some properties have an enumerated list of valid values, also known as a *code list*. Code list values can be described in the UML model in comments or documented elsewhere in the system documentation. In my example, to keep the model clean, I simply list these properties as having a data type `code`. I will record the valid values in the mapping spreadsheet created in the next step of the IEPD process.

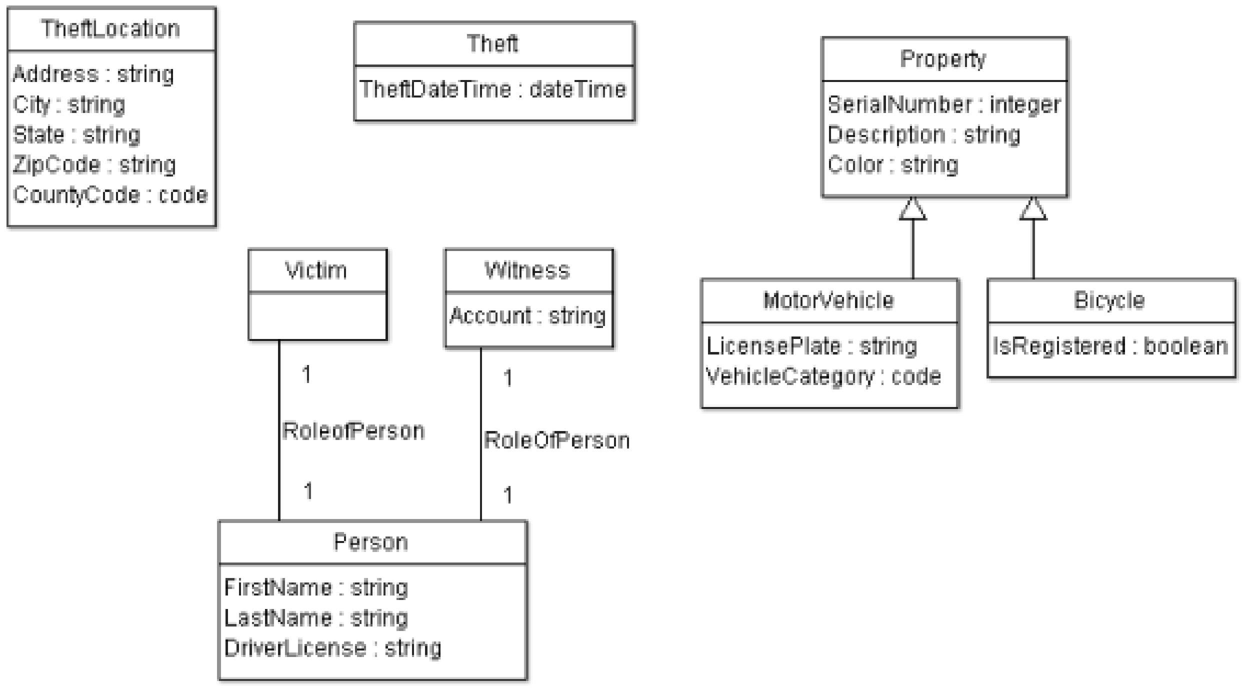
Generalizations and roles

The NIEM model uses generalizations, and when appropriate, you should use them in your model, too. In the case study, `MotorVehicle` and `Bicycle` are both specific kinds of property that might be stolen. So, I decide to add a more generic `Property` class and derive `MotorVehicle` and `Bicycle` from that. This allows me to define the common properties such as `SerialNumber` only once, and will also simplify associations by allowing the `Property` class to be the associated with `Theft` class.

`Victim` and `Witness` appear to follow the same rule. After all, they are both just more specific kinds of people. However, a person's state of being a witness or a victim is temporary, so it is better represented as a role. In fact, in this case, the same person could be both a victim and a witness in a particular theft. In that case, you would only want to represent one person with two different roles. I show that in my model by adding a separate `Person` class and creating associations to the `Victim` and `Witness` classes. I label the associations "Role Of Person" to indicate that they are related via a role rather than a normal association.

Figure 2 shows the model after I have added my generalizations and roles.

Figure 2. UML model with generalizations and roles added



Relationships

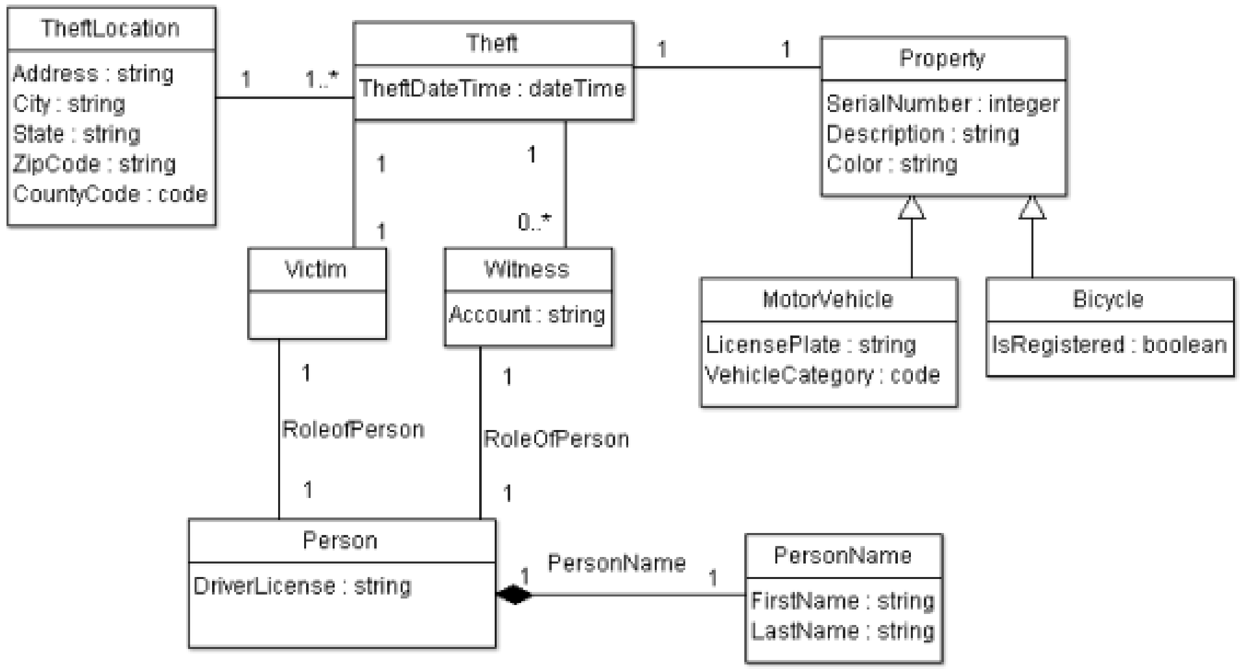
UML has three ways of representing relationships: aggregation, composition, and association.

Aggregation and composition relationships generally represent "has a" relationships, where one class is subordinate to another. In the example case study, a `Person` "has a" `PersonName`. The `PersonName` class is not useful without a person to relate it to. Aggregations and compositions are treated the same way in NIEM. In the eventual XML structure, the subordinate class will be contained in the other class. For example, there will be a Person element that contains a `PersonName` element.

In contrast, associations are between two classes that can stand on their own. In the example case study, a `Theft` and a `TheftLocation` are two separate things; one can exist without the other. To represent these in your model, you can use generic UML associations, or, if there are additional properties relating to the association itself, add separate association classes to the model. Either way, in the NIEM XML structure, the classes will each be represented as distinct elements with a separate association element that contains references to the elements that it is relating—in this case, `Theft` and a `TheftLocation`.

In the example case study, I use composition to represent the `Person`/`PersonName` relationship, and simple UML associations to relate the rest of the classes to each other. Figure 3 shows the model after I have added relationships.

Figure 3. UML model with relationships added

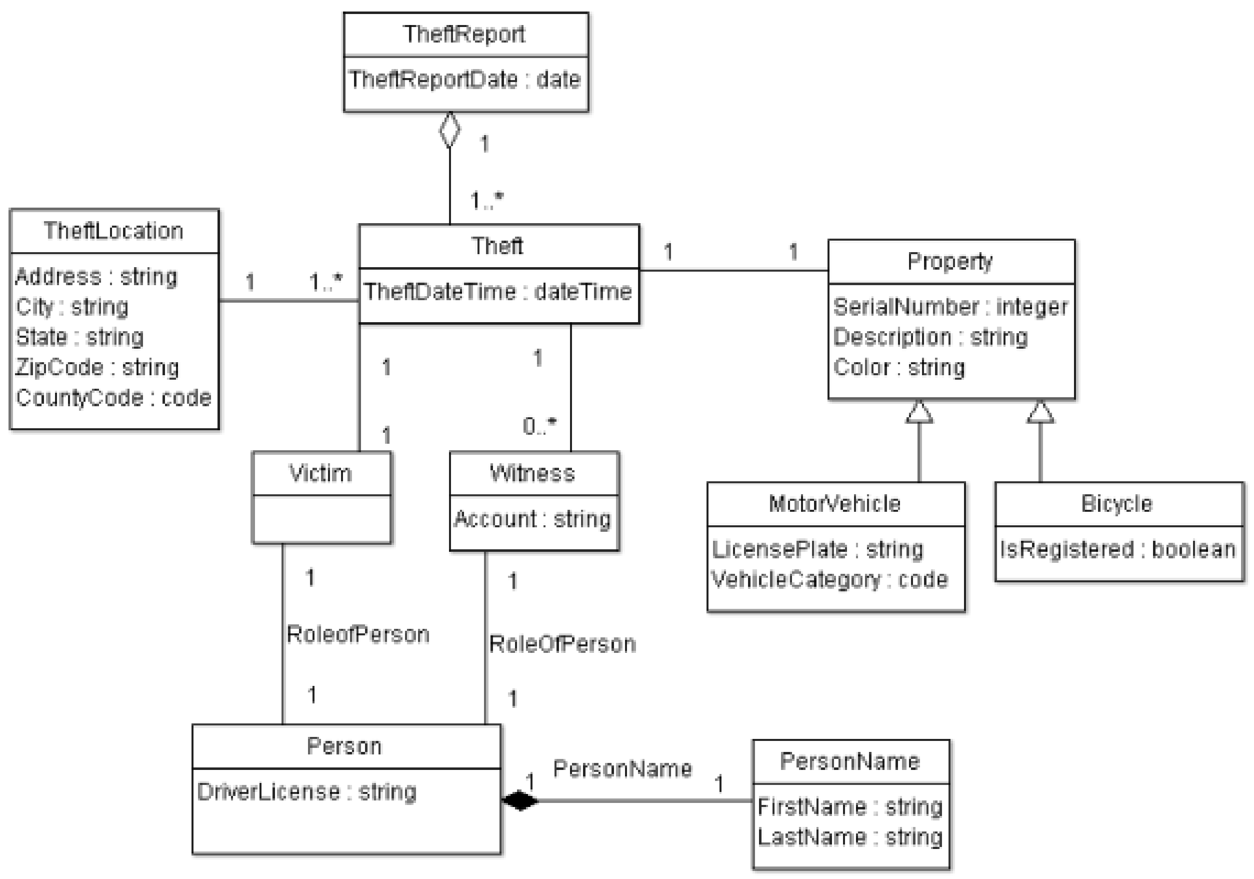


Choosing a root

Every XML message must have a single root. Generally, in an XML exchange, there is a single focal point or purpose for the message. In my case, it is the theft report itself. I add a class for `TheftReport` to my model, along with a property `TheftReportDate`. I create an aggregation relationship between `TheftReport` and `Theft`, indicating that the Theft Report consists of a set of thefts.

The complete UML model is shown in Figure 4. This model is not yet perfect, nor does it have to be. It is common to make iterative changes to the model throughout the IEPD development process. For example, it might be useful to modify the structure or names to better fit the NIEM model, where appropriate.

Figure 4. Completed UML model



Summary

This section described at a high level the steps involved in creating a NIEM IEPD and delved into detail on the first step: creating the model. The result is a working draft of a UML model that you can use to continue IEPD development. Using NIEM-targeted concepts like roles and XML Schema data types during the modeling process makes the rest of the IEPD development process easier.

Step 2: Map and subset NIEM

Mapping your model to NIEM

Now that you've created a UML model of your exchange, the next step is to map your model to NIEM to determine what parts of NIEM you will reuse in your messages. This mapping is most commonly done in a spreadsheet, known as a *Component Mapping Template* (CMT). The CMT is useful for several reasons:

- It provides a detailed, human-readable definition of your exchange model with places for comments and extra documentation.
- It makes explicit which parts of the model reuse NIEM components and which are custom to the IEPD.
- It serves as a convenient checklist when you make a subset of the NIEM model.

Creating a component mapping template (CMT)

You typically create CMTs in Microsoft® Office Excel® or other spreadsheet software, such as OpenOffice.org Calc. However, you can create them in any tabular format. There is no one required format for a CMT, but a typical CMT has, at a minimum, the following columns:

- *Source type*: The name of the class in the UML model
- *Source property*: The name of the property in the UML model
- *Data type*: The data type of the property
- *Description*: A short description of the type or property
- *Cardinality*: How many of the properties are allowed to appear
- *Extension indicator*: Whether the model matches a component in the NIEM model
- *XPath*: The path to the element in an XML message

Some NIEM implementers add more columns to the CMT to represent the details of extending NIEM. The next section of this article will look further into extending NIEM.

Recording your model in the CMT

Your first step is to record your UML model in the first five columns of the CMT. Based on our UML class diagram, Table 2 shows the `TheftLocation` class in CMT format. Descriptions are omitted from the table to save space, but a completed example CMT is available from the Artifacts page.

Table 2. Representing a type and properties in the CMT

Source type	Source property	Data type	Description	Cardinality
TheftLocation			...	
TheftLocation	Address	string	...	0..1
TheftLocation	City	string	...	0..1
TheftLocation	State	string	...	0..1
TheftLocation	ZipCode	string	...	0..1
TheftLocation	CountyCode	CountyCode	...	0..1

In the *Data type* column, XML Schema simple type names are used. In the case of code lists, a code list name is specified, and the valid values are documented in another tab of the spreadsheet. *Cardinality* shows the minimum and maximum number of occurrences, where * represents an unbounded number.

Each association should have a row in the CMT, along with rows for references to the types involved in the association. Table 3 shows a CMT representation of the `Theft/TheftLocation` association.

Table 3. Representing an association in the CMT

Source type	Source property	Data type	Description	Cardinality
Theft/TheftLocation Assn				
Theft/TheftLocation Assn	Theft	reference	...	1..1
Theft/TheftLocation Assn	TheftLocation	reference	...	1..1

Role types should be shown with references from the role to the type that is playing the role. The `Witness` role type shown in Table 4 contains a reference to `Person`, labelled `RoleOfPerson`.

Table 4. Representing a role in the CMT

Source type	Source property	Data type	Description	Cardinality
Witness				
Witness	Account	string	...	0..1
Witness	RoleOfPerson	reference	...	1..1

Searching for NIEM equivalents

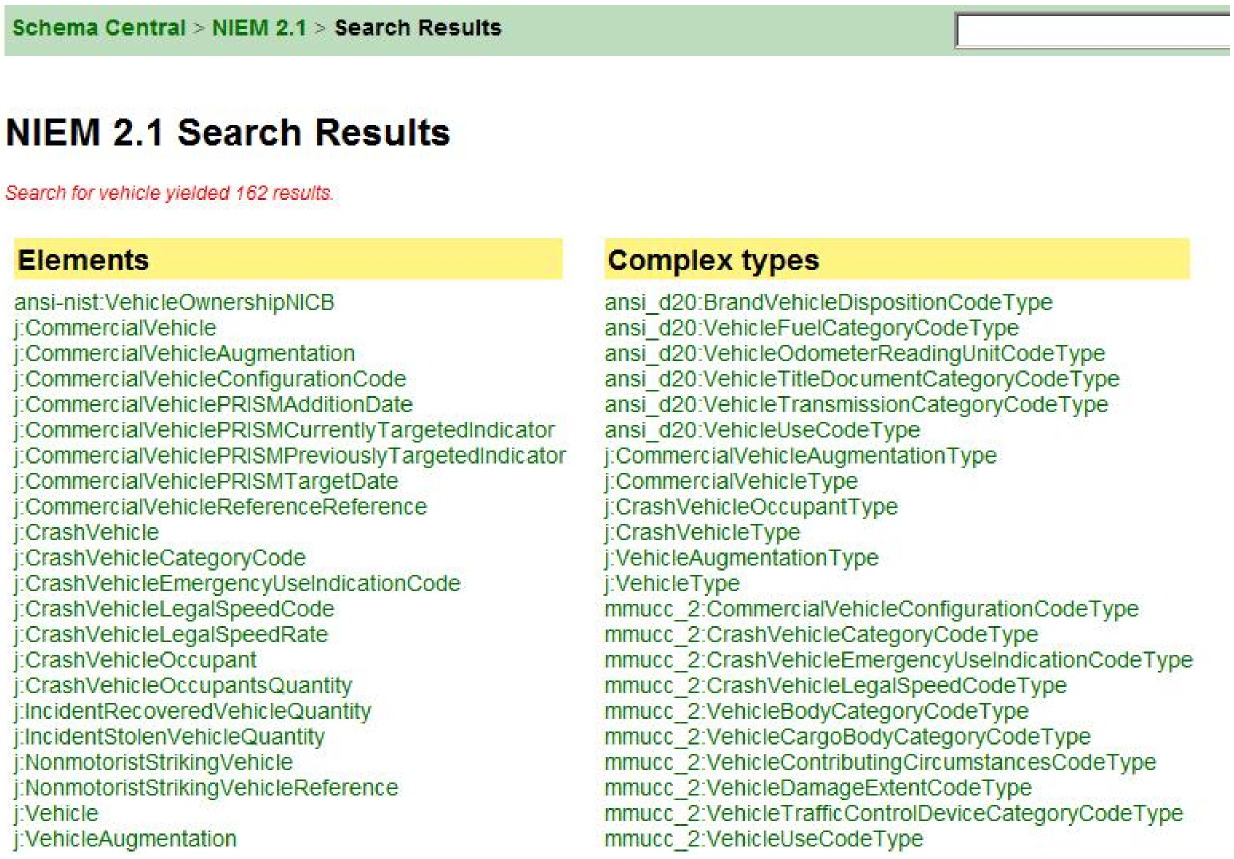
The next task in mapping your exchange is to determine where your model overlaps with NIEM and record those elements in the CMT. You want to reuse NIEM as much as possible to maximize interoperability with other NIEM applications. An IEPD is not NIEM-conformant if it adds new components when semantically equivalent components already exist in the NIEM model. That said, you should not force data into NIEM if it really doesn't fit. This article will explain later how to add new components to the model.

Because the NIEM model is very large, you do not want to scan the schemas by hand looking for matching components. Fortunately, several online tools are available to find components in the NIEM model:

- NIEM Wayfarer allows you to search for NIEM components and traverse through the model with one page per component.
- Schema Central has similar capabilities to NIEM Wayfarer but works with a variety of XML vocabularies, not just NIEM.
- The NIEM Schema Subset Generation Tool (SSGT) lets you search and navigate the NIEM model in a slightly more graphical fashion. It has the added capability of generating a NIEM subset once you find the components of interest.

Use one of these tools to look for all the components in your CMT that might already exist in NIEM. As an example, when you search for the term "Vehicle" in Schema Central, you see the search results page in Figure 5.

Figure 5. Schema Central search results page



Schema Central > NIEM 2.1 > Search Results

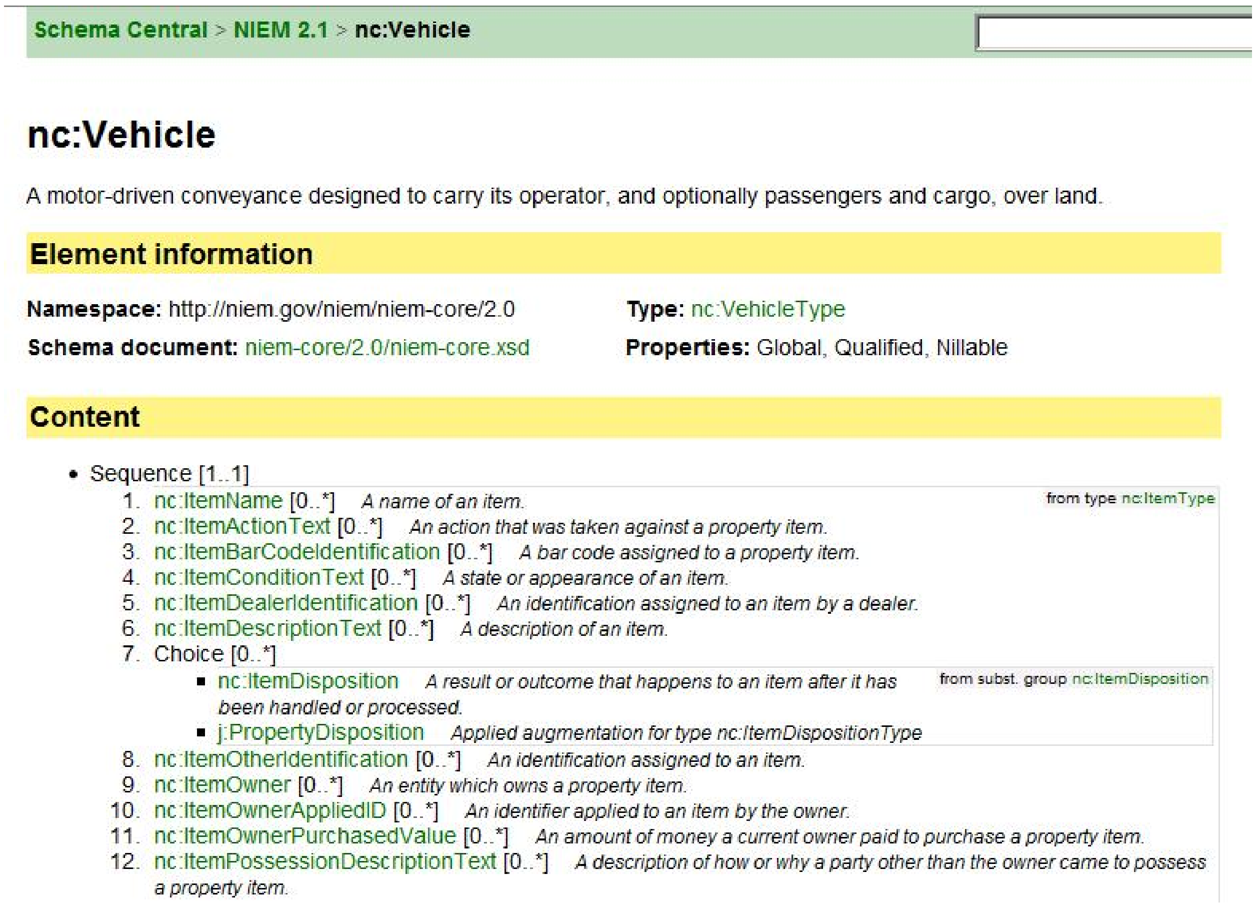
NIEM 2.1 Search Results

Search for vehicle yielded 162 results.

Elements	Complex types
ansi-nist:VehicleOwnershipNICB	ansi_d20:BrandVehicleDispositionCodeType
j:CommercialVehicle	ansi_d20:VehicleFuelCategoryCodeType
j:CommercialVehicleAugmentation	ansi_d20:VehicleOdometerReadingUnitCodeType
j:CommercialVehicleConfigurationCode	ansi_d20:VehicleTitleDocumentCategoryCodeType
j:CommercialVehiclePRISMAdditionDate	ansi_d20:VehicleTransmissionCategoryCodeType
j:CommercialVehiclePRISMCurrentlyTargetedIndicator	ansi_d20:VehicleUseCodeType
j:CommercialVehiclePRISMPreviouslyTargetedIndicator	j:CommercialVehicleAugmentationType
j:CommercialVehiclePRISMTargetDate	j:CommercialVehicleType
j:CommercialVehicleReferenceReference	j:CrashVehicleOccupantType
j:CrashVehicle	j:CrashVehicleType
j:CrashVehicleCategoryCode	j:VehicleAugmentationType
j:CrashVehicleEmergencyUseIndicationCode	j:VehicleType
j:CrashVehicleLegalSpeedCode	mmucc_2:CommercialVehicleConfigurationCodeType
j:CrashVehicleLegalSpeedRate	mmucc_2:CrashVehicleCategoryCodeType
j:CrashVehicleOccupant	mmucc_2:CrashVehicleEmergencyUseIndicationCodeType
j:CrashVehicleOccupantsQuantity	mmucc_2:CrashVehicleLegalSpeedCodeType
j:IncidentRecoveredVehicleQuantity	mmucc_2:VehicleBodyCategoryCodeType
j:IncidentStolenVehicleQuantity	mmucc_2:VehicleCargoBodyCategoryCodeType
j:NonmotoristStrikingVehicle	mmucc_2:VehicleContributingCircumstancesCodeType
j:NonmotoristStrikingVehicleReference	mmucc_2:VehicleDamageExtentCodeType
j:Vehicle	mmucc_2:VehicleTrafficControlDeviceCategoryCodeType
j:VehicleAugmentation	mmucc_2:VehicleUseCodeType

When you click on **nc:Vehicle**, the page shown in Figure 6 is displayed. It shows some general characteristics of the element, followed by a complete listing of its possible children.

Figure 6. Schema Central element display page



Schema Central > NIEM 2.1 > nc:Vehicle

nc:Vehicle

A motor-driven conveyance designed to carry its operator, and optionally passengers and cargo, over land.

Element information

Namespace: http://niem.gov/niem/niem-core/2.0
Type: nc:VehicleType
Schema document: niem-core/2.0/niem-core.xsd
Properties: Global, Qualified, Nillable

Content

- Sequence [1..1]
  1. nc:ItemName [0..*] *A name of an item.* from type nc:ItemType
  2. nc:ItemActionText [0..*] *An action that was taken against a property item.*
  3. nc:ItemBarCodeIdentification [0..*] *A bar code assigned to a property item.*
  4. nc:ItemConditionText [0..*] *A state or appearance of an item.*
  5. nc:ItemDealerIdentification [0..*] *An identification assigned to an item by a dealer.*
  6. nc:ItemDescriptionText [0..*] *A description of an item.*
  7. Choice [0..*]
     - nc:ItemDisposition *A result or outcome that happens to an item after it has been handled or processed.* from subst. group nc:ItemDisposition
     - j:PropertyDisposition *Applied augmentation for type nc:ItemDispositionType*
  8. nc:ItemOtherIdentification [0..*] *An identification assigned to an item.*
  9. nc:ItemOwner [0..*] *An entity which owns a property item.*
  10. nc:ItemOwnerAppliedID [0..*] *An identifier applied to an item by the owner.*
  11. nc:ItemOwnerPurchasedValue [0..*] *An amount of money a current owner paid to purchase a property item.*
  12. nc:ItemPossessionDescriptionText [0..*] *A description of how or why a party other than the owner came to possess a property item.*

All of the NIEM components have a namespace prefix. *nc* refers to NIEM Core, the namespace where the most fundamental types reside. There is also a namespace for each of the domains (e.g. "*j*" for Justice.) Feel free to use NIEM components from any domain as long as they are semantically equivalent to your model. You don't have to be implementing an immigration-related exchange to use an element from the immigration domain.

Guidelines for searching the NIEM model

Regardless of which tool you use, you can make searching the model easier by following these tips:

- It is often easier to start by looking for the highest-level types/classes (in the example case, `Theft`, `Property`, `Location`, and so on) first, and then finding the appropriate properties.
- Don't forget to search for synonyms. If you don't find "License Plate", look for "Registration".
- If you can't find your specific component, look for a more general one. Some of the most general types in NIEM are `Person`, `Organization`, `Location`, `Activity`, and `Item`. For example, if you don't find Theft Location, you can look for Location more generally and use `nc:Location`. If there is not a specific type for "Theft", consider using the more generic `nc:Activity`.
- Don't just search names. If you expand your scope to search descriptions and enumerations, it might lead you to the appropriate type.

Finding components in the NIEM model might seem daunting at first, but it gets easier as you become more familiar with the general naming and structural patterns of the NIEM model.

Recording NIEM components in the CMT

When you find an equivalent NIEM component, record it in the CMT in the XPath column. Generally, simple XPath expressions are used—element and/or attribute names are separated by slashes (/). Type names do not need to be included in the XPath. Use namespace prefixes such as `nc:`, because element names are not necessarily unique across namespaces.

Table 5 shows the XPath mappings for `TheftLocation`.

Table 5. TheftLocation XPath mappings

Source type	Source property	...	Ext?	XPath
TheftLocation		...		nc:Location
TheftLocation	Address	...	N	nc:Location/nc:LocationAddress/ nc:StructuredAddress/nc:LocationStreet/ nc:StreetFullText
TheftLocation	City	...	N	nc:Location/nc:LocationAddress/ nc:StructuredAddress/nc:LocationCity Name
TheftLocation	State	...	N	nc:Location/nc:LocationAddress/ nc:StructuredAddress/nc:LocationState USPostalServiceCode
TheftLocation	Zip	...	N	nc:Location/nc:LocationAddress/ nc:StructuredAddress/nc:LocationPostal Code
TheftLocation	CountyCode	...	Y	

You should include enough steps in the XPath to uniquely identify it. For example, don't just put `nc:StreetFullText` in the row for `Address`. Sometimes, multiple paths can lead to an element in NIEM, and the entire path is needed for precision.

In the example, the `CountyCode` property, which is a state-specific county code, is not found in NIEM, so it will require an extension. Therefore, the *Ext?* column is set to Y, and the XPath is left blank for now. The next section of this article will walk through the process of filling in the XPaths for extensions.

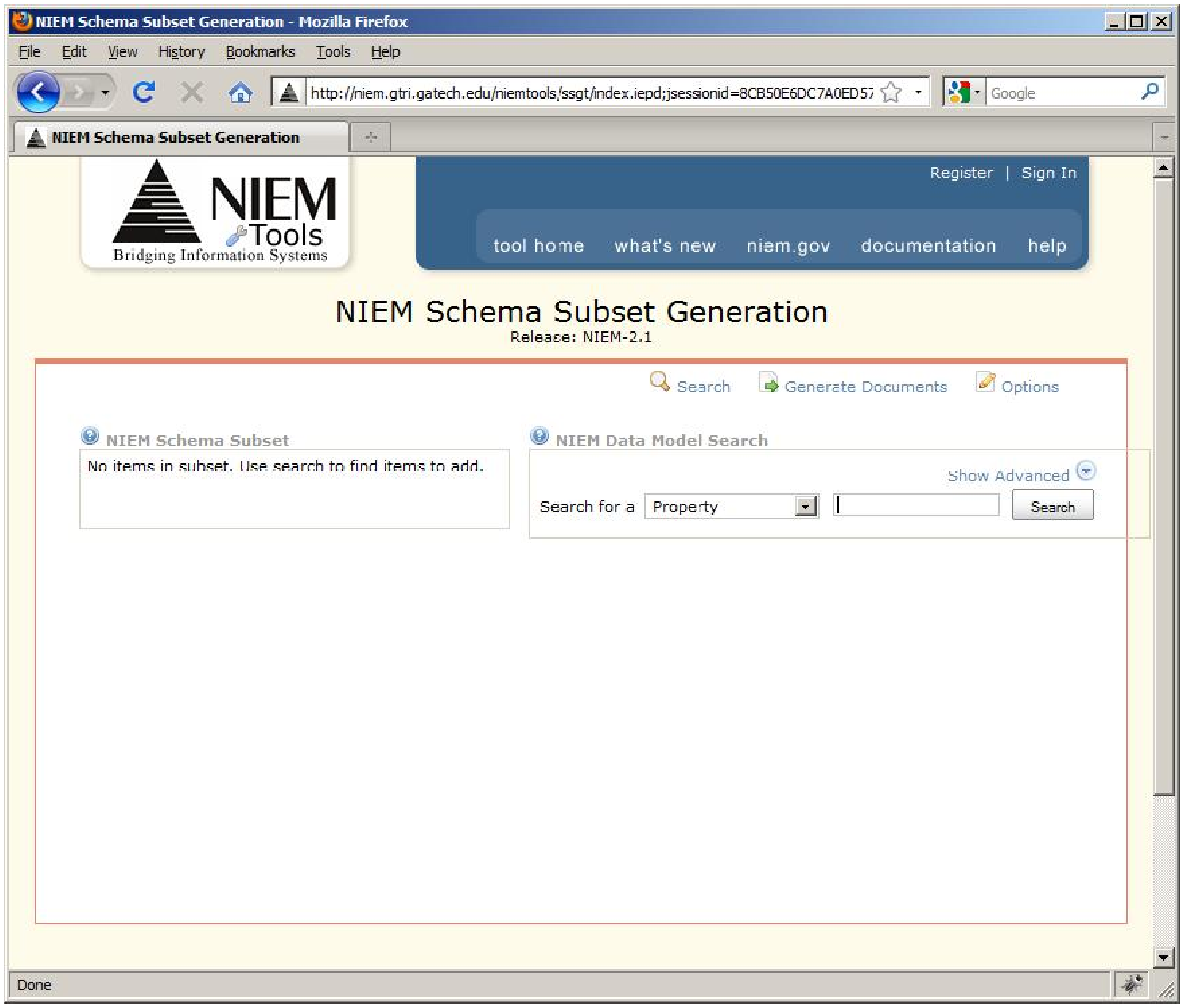
A complete mapping of the Theft Report example model to NIEM is available from the Artifacts page.

Creating a NIEM subset

When you have decided which components of NIEM you want to use in your exchange, you create a subset of the NIEM model that takes the form of a set of XML Schema documents. Because the full NIEM model is so large and loosely constrained, a NIEM subset is necessary to validate your exchange more precisely. The NIEM subset restricts the elements and attributes allowed, the number of times they can occur, and—in some cases—their allowed values. Creating a NIEM subset also speeds up validation of XML messages, because the schemas are much smaller.

You create NIEM subsets using the NIEM SSGT. The initial page of the SSGT in Figure 7 has two panes. The right pane is where you search and navigate the model, and the left pane shows your subset as you add components to it.

Figure 7. SSGT main page


NIEM Schema Subset Generation - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://niem.gtri.gatech.edu/niemtools/ssgt/index.iepd;jsessionid=8CB50E6DC7A0ED57
Google
NIEM Schema Subset Generation
NIEM Tools
Bridging Information Systems
Register | Sign In
tool home
what's new
niem.gov
documentation
help
Search
Generate Documents
Options
Show Advanced
Search
Done

NIEM Schema Subset Generation
Release: NIEM-2.1

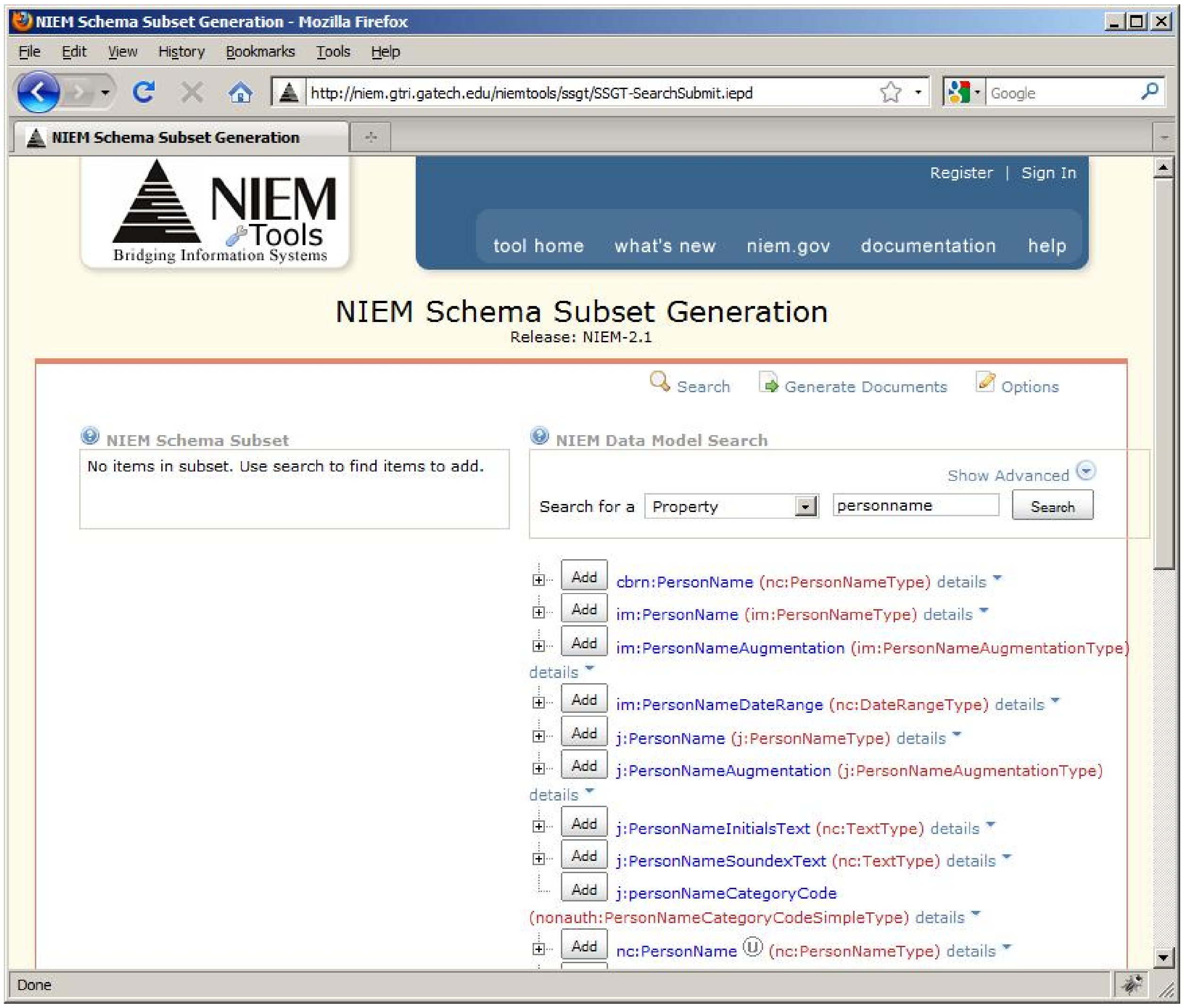
NIEM Schema Subset
No items in subset. Use search to find items to add.

NIEM Data Model Search
Search for a Property

Based on your CMT, you perform searches to find components to add to your subset. Using the SSGT, you can choose to search either properties (element or attribute names), types, associations, or other components. Because you have the names of the element in your CMT, it makes sense to search properties. Sample search results are in Figure 8.

You might wonder why mapping and subsetting are two separate steps when you can perform the tasks in the same tool (the SSGT). It is certainly possible to perform the mapping and subsetting at the same time using the SSGT. However, many NIEM practitioners find it easier to do the mapping with NIEM Wayfarer or Schema Central, which shows the actual (flattened) structure of the types more clearly. The SSGT requires more knowledge of NIEM (and more clicking) to navigate, so going to the SSGT prepared with a CMT that lists exactly what you want from NIEM makes subsetting more efficient.

Figure 8. SSGT search results page


NIEM Schema Subset Generation - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://niem.gtri.gatech.edu/niemtools/ssgt/SSGT-SearchSubmit.iepd
NIEM Schema Subset Generation
NIEM Tools
Bridging Information Systems
Register | Sign In
tool home what's new niem.gov documentation help
Search Generate Documents Options
Show Advanced
Search
Done

NIEM Schema Subset Generation
Release: NIEM-2.1

NIEM Schema Subset
No items in subset. Use search to find items to add.

NIEM Data Model Search
Search for a Property personname

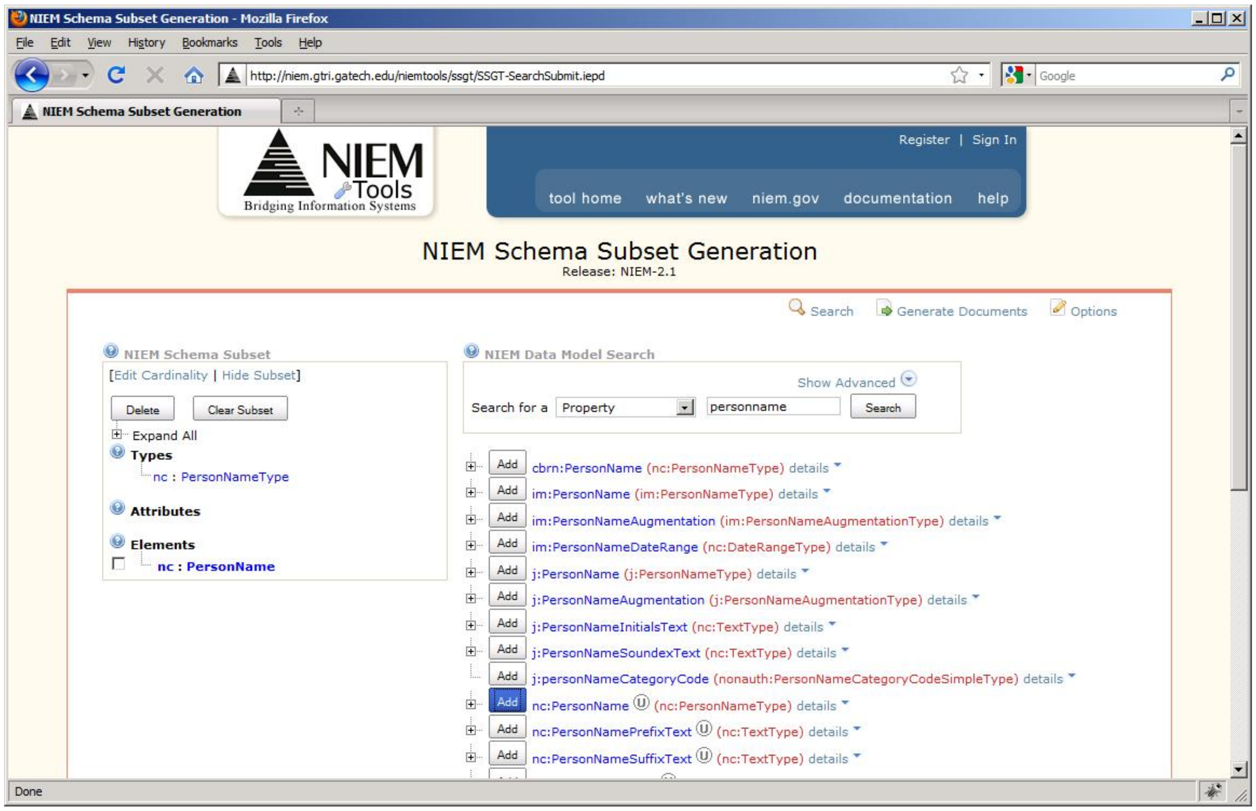
- Add cbrn:PersonName (nc:PersonNameType) details
- Add im:PersonName (im:PersonNameType) details
- Add im:PersonNameAugmentation (im:PersonNameAugmentationType) details
- Add im:PersonNameDateRange (nc:DateRangeType) details
- Add j:PersonName (j:PersonNameType) details
- Add j:PersonNameAugmentation (j:PersonNameAugmentationType) details
- Add j:PersonNameInitialsText (nc:TextType) details
- Add j:PersonNameSoundexText (nc:TextType) details
- Add j:personNameCategoryCode (nonauth:PersonNameCategoryCodeSimpleType) details
- Add nc:PersonName Ⓤ (nc:PersonNameType) details

Adding properties to the subset

When the NIEM component of interest is displayed, click **Add** to add it to your subset. It then appears in the left pane under **NIEM Schema Subset** as shown in Figure 9.

When you add a property, its type automatically goes with it. For example, if you add `nc:PersonName`, `nc:PersonNameType` is automatically added to the subset, as well. The components you explicitly selected appear in the left pane in bold, with a check box next to them, while the dependent components are not in bold.

Figure 9. SSGT subset



NIEM Schema Subset Generation - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://niem.gtri.gatech.edu/niemtools/ssgt/SSGT-SearchSubmit.iepd

NIEM Schema Subset Generation

Register | Sign In

tool home what's new niem.gov documentation help

NIEM Schema Subset Generation
Release: NIEM-2.1

Search Generate Documents Options

NIEM Schema Subset
[Edit Cardinality | Hide Subset]
Delete Clear Subset
Expand All
Types
nc : PersonNameType
Attributes
Elements
nc : PersonName

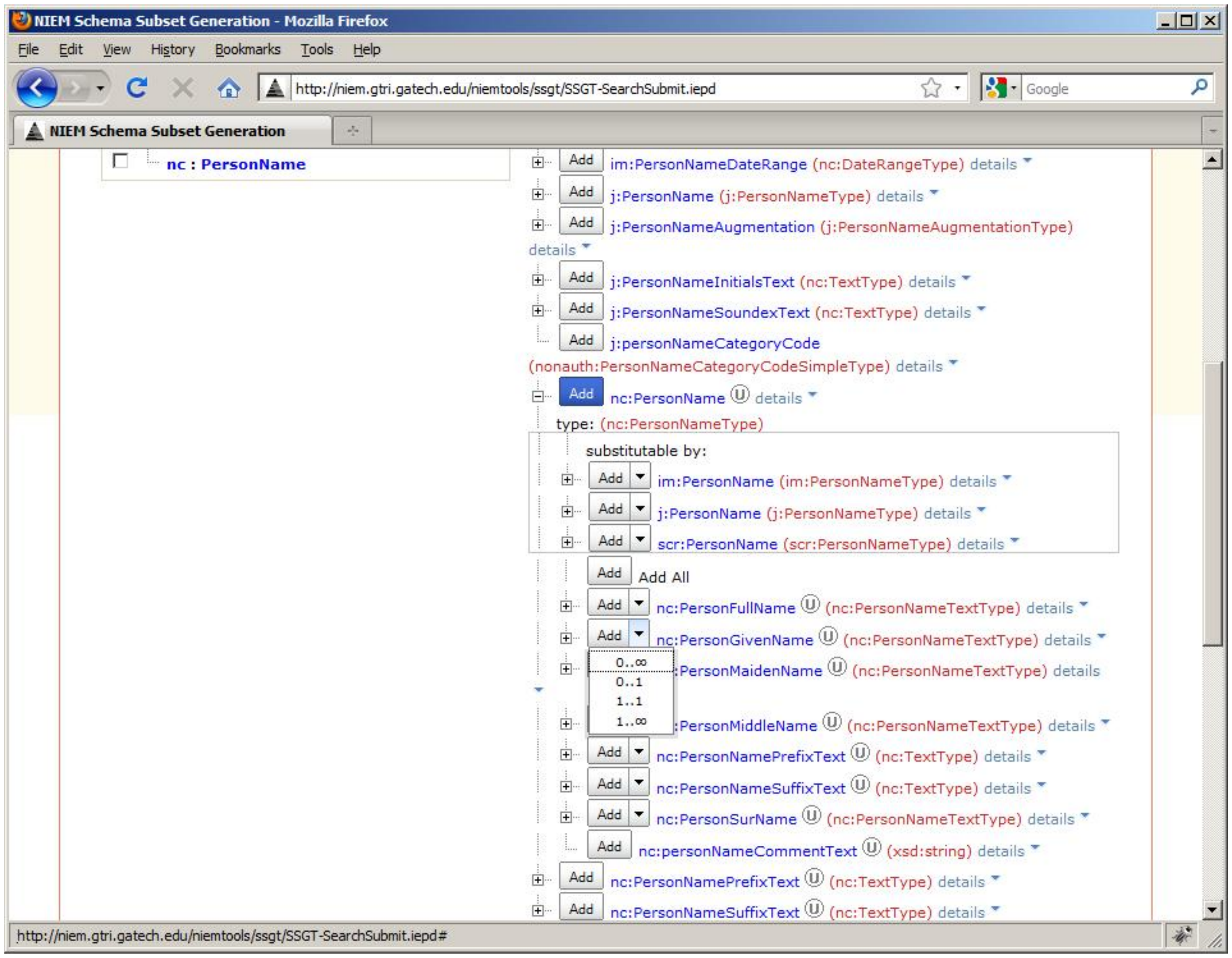
NIEM Data Model Search
Show Advanced
Search for a Property personname Search

Add cbrn:PersonName (nc:PersonNameType) details
Add im:PersonName (im:PersonNameType) details
Add im:PersonNameAugmentation (im:PersonNameAugmentationType) details
Add im:PersonNameDateRange (nc:DateRangeType) details
Add j:PersonName (j:PersonNameType) details
Add j:PersonNameAugmentation (j:PersonNameAugmentationType) details
Add j:PersonNameInitialsText (nc:TextType) details
Add j:PersonNameSoundexText (nc:TextType) details
Add j:personNameCategoryCode (nonauth:PersonNameCategoryCodeSimpleType) details
Add nc:PersonName (U) (nc:PersonNameType) details
Add nc:PersonNamePrefixText (U) (nc:TextType) details
Add nc:PersonNameSuffixText (U) (nc:TextType) details

Done

The SSGT does not add the child properties of a type by default. For example, if you add `nc:PersonName`, it does not include the properties `nc:PersonGivenName` and `nc:PersonSurName`. These you must add to the subset separately. When you add them, you must do so in the context of `nc:PersonName`, so that the parent-child relationship between, for example, `nc:PersonName` and `nc:PersonGivenName` is maintained. To do this, expand the `nc:PersonName` tree in the SSGT search results and click **Add** next to `nc:PersonGivenName`, as in Figure 10.

Figure 10. Adding a child using the SSGT


NIEM Schema Subset Generation - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://niem.gtri.gatech.edu/niemtools/ssgt/SSGT-SearchSubmit.iepd

NIEM Schema Subset Generation

nc : PersonName

Add im:PersonNameDateRange (nc:DateRangeType) details
Add j:PersonName (j:PersonNameType) details
Add j:PersonNameAugmentation (j:PersonNameAugmentationType) details
Add j:PersonNameInitialsText (nc:TextType) details
Add j:PersonNameSoundexText (nc:TextType) details
Add j:personNameCategoryCode (nonauth:PersonNameCategoryCodeSimpleType) details
Add nc:PersonName Ⓤ details
type: (nc:PersonNameType)
substitutable by:
Add im:PersonName (im:PersonNameType) details
Add j:PersonName (j:PersonNameType) details
Add scr:PersonName (scr:PersonNameType) details
Add Add All
Add nc:PersonFullName Ⓤ (nc:PersonNameTextType) details
Add nc:PersonGivenName Ⓤ (nc:PersonNameTextType) details
0..∞
0..1
1..1
1..∞
:PersonMaidenName Ⓤ (nc:PersonNameTextType) details
:PersonMiddleName Ⓤ (nc:PersonNameTextType) details
Add nc:PersonNamePrefixText Ⓤ (nc:TextType) details
Add nc:PersonNameSuffixText Ⓤ (nc:TextType) details
Add nc:PersonSurName Ⓤ (nc:PersonNameTextType) details
Add nc:personNameCommentText Ⓤ (xsd:string) details
Add nc:PersonNamePrefixText Ⓤ (nc:TextType) details
Add nc:PersonNameSuffixText Ⓤ (nc:TextType) details

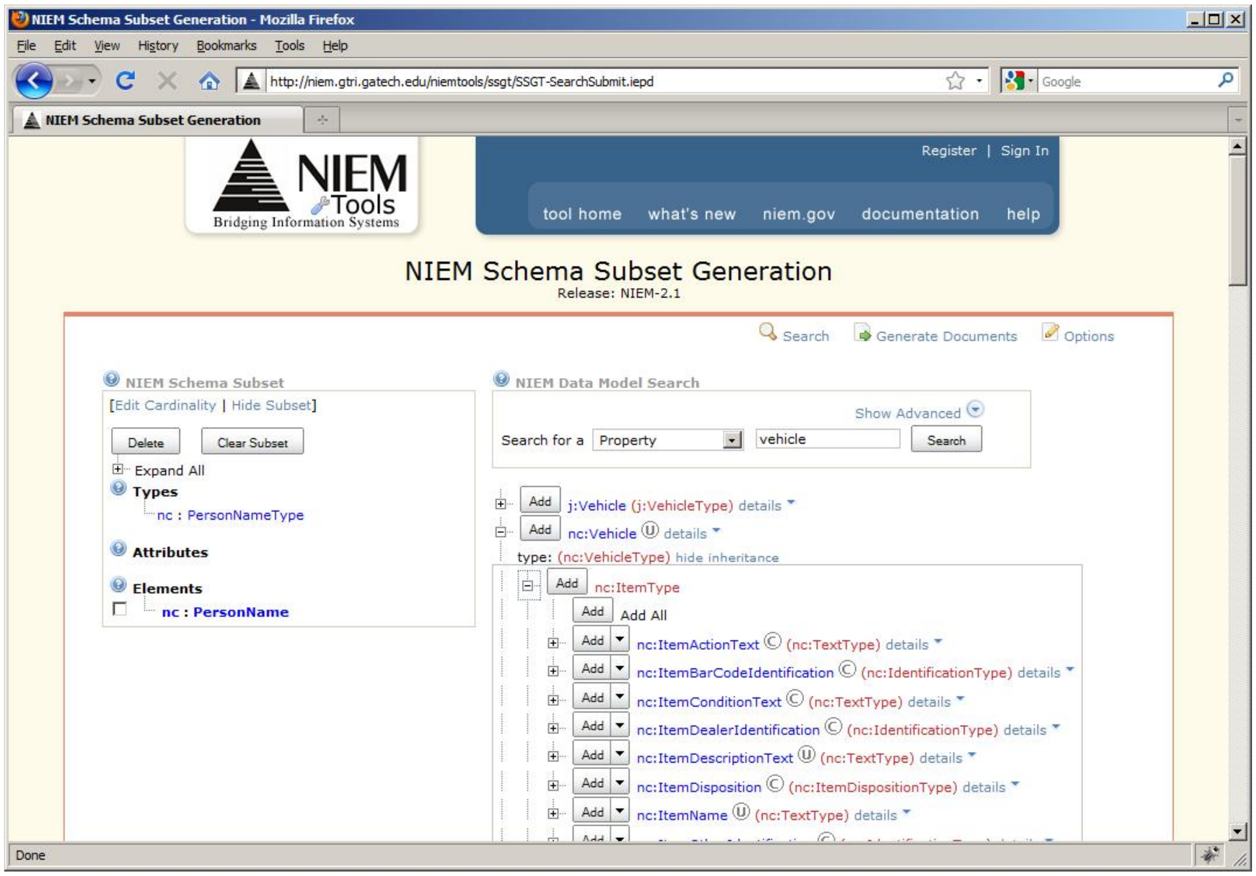
http://niem.gtri.gatech.edu/niemtools/ssgt/SSGT-SearchSubmit.iepd#

If instead you separately searched for `PersonGivenName` and added it from the search results, the element would be added but not as a child of `nc:PersonName`.

Figure 10 also shows that when you add a property of a type, you can specify the cardinality. Clicking the right down arrow on the **Add** button brings up a drop-down menu that shows possible cardinalities. The default is 0 to infinity.

If a property is included by inheritance, it is not displayed in the SSGT hierarchy by default. For example, expanding `nc:Vehicle` in the SSGT search results does not automatically show the `nc:ItemDescriptionText` that is mapped to the Property Description property. To see these inherited properties, click **show inheritance** (next to `nc:VehicleType`) and expand the type that contains the property of interest—in this case, `nc:ItemType`, as in Figure 11.

Figure 11. Adding an inherited property using the SSGT

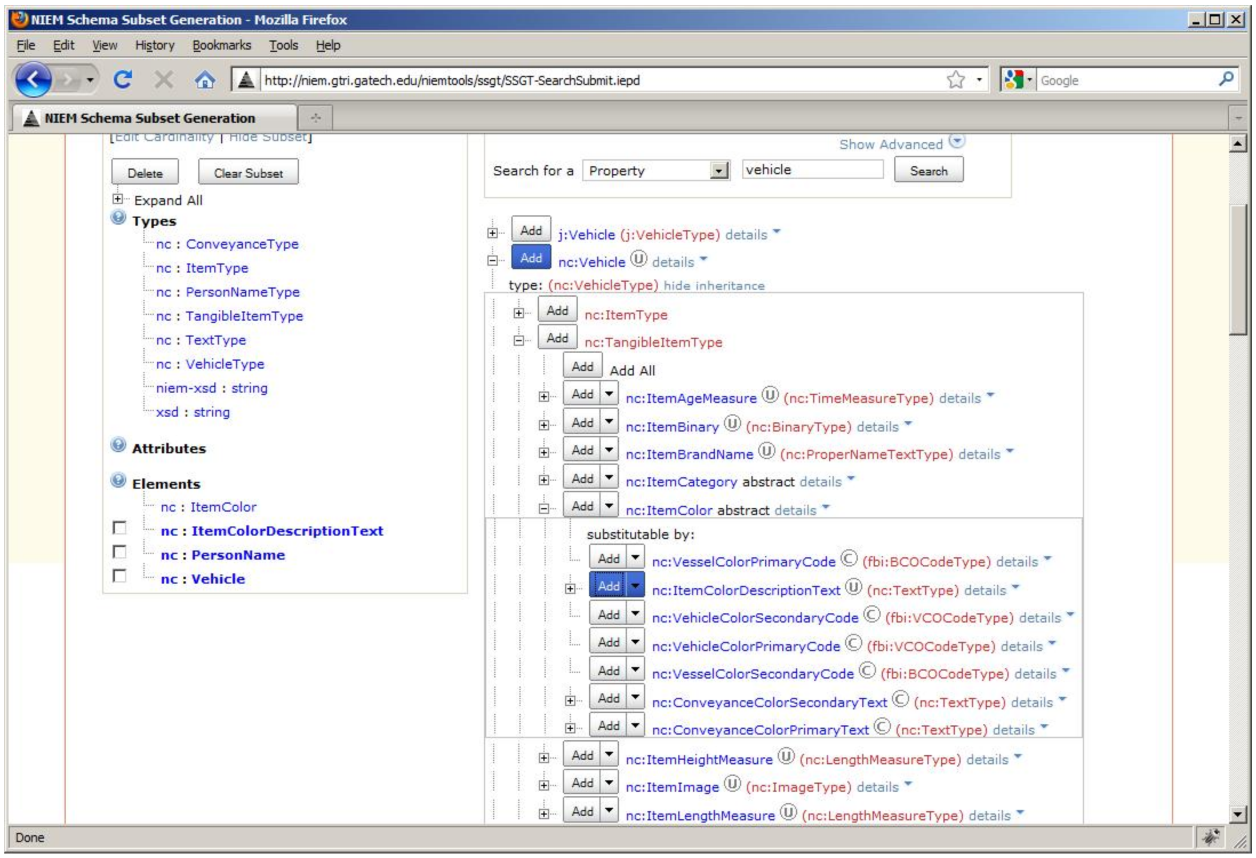

NIEM Schema Subset Generation - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://niem.gtri.gatech.edu/niemtools/ssgt/SSGT-SearchSubmit.iepd
NIEM Tools Bridging Information Systems
Register | Sign In
tool home what's new niem.gov documentation help
NIEM Schema Subset Generation
Release: NIEM-2.1
Search Generate Documents Options
NIEM Schema Subset
[Edit Cardinality | Hide Subset]
Delete Clear Subset
Expand All
Types
nc : PersonNameType
Attributes
Elements
nc : PersonName
NIEM Data Model Search
Show Advanced
Search for a Property vehicle Search
Add j:Vehicle (j:VehicleType) details
Add nc:Vehicle details
type: (nc:VehicleType) hide inheritance
Add nc:ItemType
Add Add All
Add nc:ItemActionText (nc:TextType) details
Add nc:ItemBarCodeIdentification (nc:IdentificationType) details
Add nc:ItemConditionText (nc:TextType) details
Add nc:ItemDealerIdentification (nc:IdentificationType) details
Add nc:ItemDescriptionText (nc:TextType) details
Add nc:ItemDisposition (nc:ItemDispositionType) details
Add nc:ItemName (nc:TextType) details
Done

Abstract elements and the subset

The NIEM model commonly uses XML Schema abstract elements and substitution groups. For example, there are several ways to represent the color of an item. NIEM has an *abstract element*—`nc:ItemColor`—that cannot appear anywhere in an XML instance. Instead, it must be substituted by one of several elements, such as `nc:VehicleColorPrimaryCode` or `nc:ItemColorDescriptionText`. In XML Schema terminology, `nc:VehicleColorPrimaryCode` and `nc:ItemColorDescriptionText` are said to be members of a *substitution group* whose *head* is `nc:ItemColor`.

The abstract elements add some complexity to the creation of a subset, because you are required to add the substitutable elements in your subset, not just the abstract element. The SSGT marks all abstract elements with the word "abstract" and allows you to expand them to see the substitutable elements, as in Figure 12.

Figure 12. Adding a substitutable element using the SSGT



NIEM Schema Subset Generation - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://niem.gtri.gatech.edu/niemtools/ssgt/SSGT-SearchSubmit.iepd

NIEM Schema Subset Generation

[Edit Cardinality | Hide Subset]

Delete Clear Subset

Expand All

Types
- nc : ConveyanceType
- nc : ItemType
- nc : PersonNameType
- nc : TangibleItemType
- nc : TextType
- nc : VehicleType
- niem-xsd : string
- xsd : string

Attributes

Elements
- nc : ItemColor
- nc : ItemColorDescriptionText
- nc : PersonName
- nc : Vehicle

Show Advanced

Search for a Property vehicle Search

```
Add j:Vehicle (j:VehicleType) details
Add nc:Vehicle (U) details
type: (nc:VehicleType) hide inheritance
  Add nc:ItemType
  Add nc:TangibleItemType
    Add Add All
    Add nc:ItemAgeMeasure (U) (nc:TimeMeasureType) details
    Add nc:ItemBinary (U) (nc:BinaryType) details
    Add nc:ItemBrandName (U) (nc:ProperNameTextType) details
    Add nc:ItemCategory abstract details
    Add nc:ItemColor abstract details
      substitutable by:
        Add nc:VesselColorPrimaryCode (C) (fbi:BCOCodeType) details
        Add nc:ItemColorDescriptionText (U) (nc:TextType) details
        Add nc:VehicleColorSecondaryCode (C) (fbi:VCOCodeType) details
        Add nc:VehicleColorPrimaryCode (C) (fbi:VCOCodeType) details
        Add nc:VesselColorSecondaryCode (C) (fbi:BCOCodeType) details
        Add nc:ConveyanceColorSecondaryText (C) (nc:TextType) details
        Add nc:ConveyanceColorPrimaryText (C) (nc:TextType) details
    Add nc:ItemHeightMeasure (U) (nc:LengthMeasureType) details
    Add nc:ItemImage (U) (nc:ImageType) details
    Add nc:ItemLengthMeasure (U) (nc:LengthMeasureType) details
```

Done

Most date-related types also contain an abstract element `nc:DateRepresentation` that is substitutable by `nc:Date`, `nc:DateTime`, and so on. It is an easy mistake to simply add a date-related property, such as `nc:ActivityDate`, without expanding it to click on `nc:DateRepresentation` and then `nc:Date` to allow for the appropriate child elements.

Fine-tuning your subset

When you have created your subset, you can modify it using the left pane of the SSGT. You can choose to delete any component by selecting the check box next to it, and then clicking **Delete**. You can also delete allowed code list values by expanding the appropriate simple types in the left pane. By default, all code list values from a simple type are included in the subset.

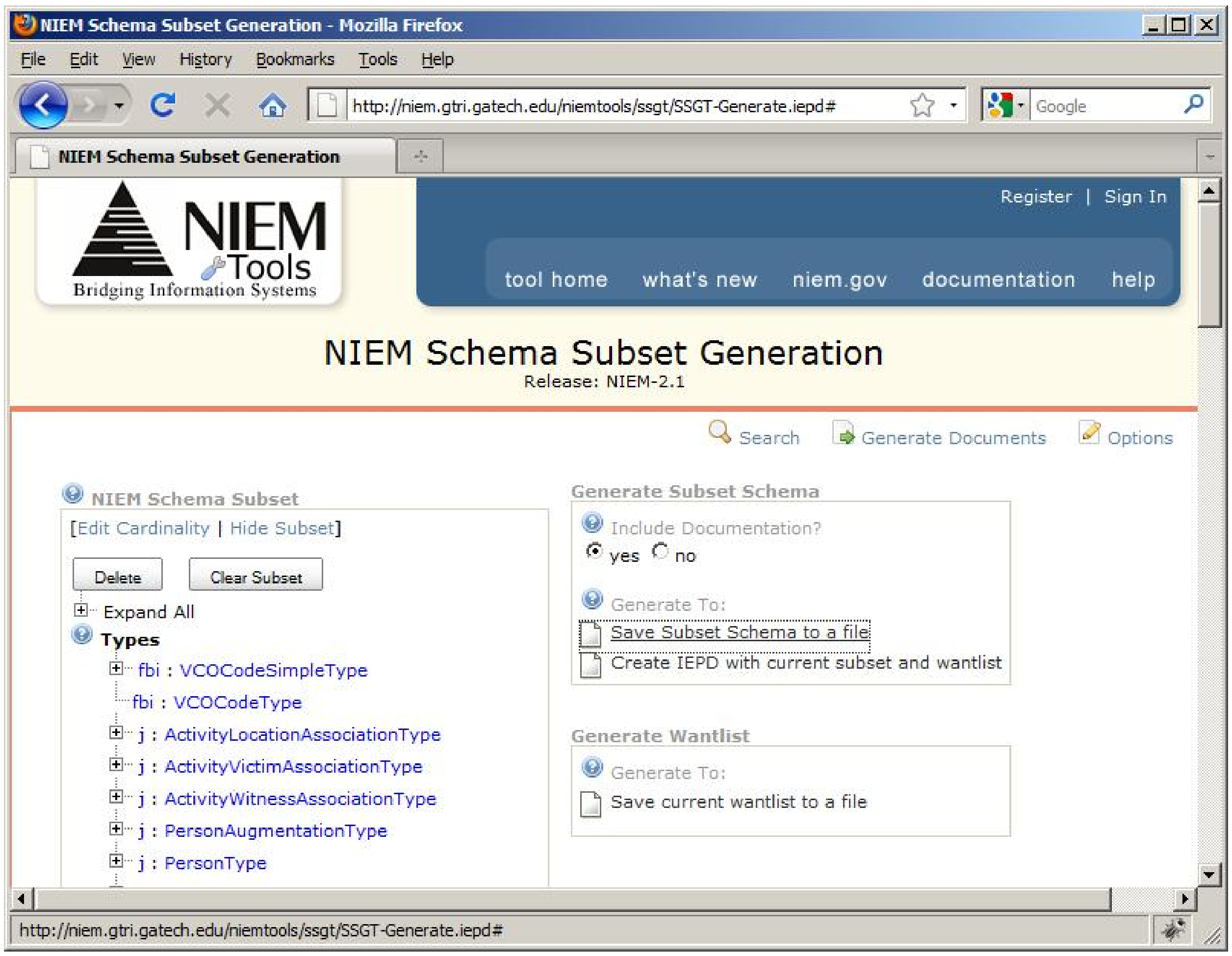
You can also choose to change the cardinalities by clicking **Edit Cardinality** at the top of the left pane. Doing so gives you another opportunity to decide how many of a particular property are allowed in a parent type.

Your NIEM subset does not have to be perfect at this point. NIEM subsetting is often an iterative process. You can save and modify your subset as needed during the final stages of IEPD development.

Generating your NIEM subset

To generate your subset, click **Generate Documents** in the upper right corner of the page. Doing so brings up a window similar to Figure 13 that shows some generation options. Select **Save Subset Schema to a file**, and choose the location in which to save it.

Figure 13. Generating a subset using the SSGT


NIEM Schema Subset Generation - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://niem.gtri.gatech.edu/niemtools/ssgt/SSGT-Generate.iepd#
Google
NIEM Schema Subset Generation
NIEM Tools
Bridging Information Systems
Register | Sign In
tool home what's new niem.gov documentation help
Search Generate Documents Options
Delete Clear Subset
Expand All

NIEM Schema Subset Generation
Release: NIEM-2.1

NIEM Schema Subset
[Edit Cardinality | Hide Subset]

Types
- fbi : VCOCodeSimpleType
- fbi : VCOCodeType
- j : ActivityLocationAssociationType
- j : ActivityVictimAssociationType
- j : ActivityWitnessAssociationType
- j : PersonAugmentationType
- j : PersonType

Generate Subset Schema
Include Documentation?
[✓ yes] no
Generate To:
Save Subset Schema to a file
Create IEPD with current subset and wantlist

Generate Wantlist
Generate To:
Save current wantlist to a file

http://niem.gtri.gatech.edu/niemtools/ssgt/SSGT-Generate.iepd#

Doing so creates a .zip file called `Subset.zip` with a `niem` subfolder that contains the NIEM subset. It has a schema document for every namespace from which you chose elements in the SSGT plus a few standard schemas that come with all subsets.

Only the types you chose are included in the schema documents, and those types only contain the chosen properties. For example, although the `nc:PersonNameType` has seven possible children in the entire NIEM model and they all have cardinalities 0..*, your subset schema will contain only what is in Example 4.

Example 4. `nc:PersonNameType` in NIEM subset

```
<xsd:complexType name="PersonNameType">
 <xsd:complexContent>
  <xsd:extension base="s:ComplexObjectType">
   <xsd:sequence>
    <xsd:element ref="nc:PersonGivenName" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="nc:PersonSurName" minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

The subset also contains an XML document called `wantlist.xml` which lists all of the components you added to your subset along with their cardinalities. The wantlist is useful if you need to make changes later: You can re-upload the wantlist to the SSGT, modify the subset, and regenerate. Example 5 shows part of the wantlist.

Example 5. Partial NIEM subset wantlist

```
<w:WantList w:release="2.1" w:product="NIEM">
 <w:Element w:name="j:Person" w:isReference="false"/>
 <w:Element w:name="j:Witness" w:isReference="false"/>
 ...
 <w:Type w:name="j:PersonType" w:isRequested="false">
  <w:ElementInType w:minOccurs="0" w:maxOccurs="1"
    w:name="j:PersonAugmentation" w:isReference="false"/>
 </w:Type>
 <w:Type w:name="j:WitnessType" w:isRequested="false">
  <w:ElementInType w:minOccurs="0" w:maxOccurs="1"
    w:name="j:WitnessAccountDescriptionText" w:isReference="false"/>
  <w:ElementInType w:minOccurs="1" w:maxOccurs="1"
    w:name="nc:RoleOfPerson" w:isReference="true"/>
 </w:Type>
 ...
</w:WantList>
```

Summary

This section showed how to map a UML exchange model to NIEM using a Component Mapping Template (CMT). It then described the process of creating a NIEM subset using the NIEM Schema Subset Generation Tool (SSGT). The next section will cover the rows of the CMT that were not filled in yet: the extensions. It will explain the different approaches to extending NIEM and take you through the process of creating Exchange and Extension schemas.

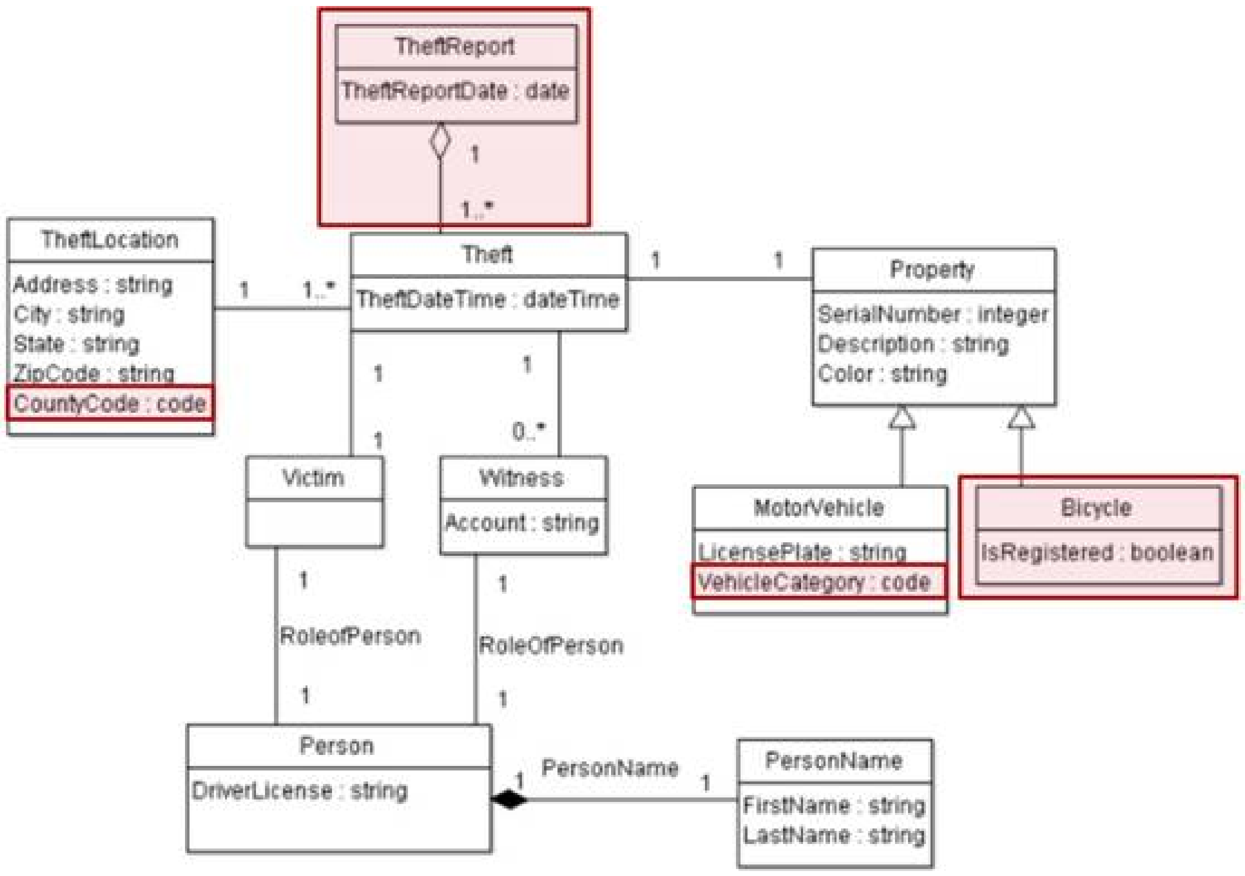
Step 3: Extend NIEM

NIEM is large—over 6000 elements—but it most likely does not contain everything you want to include in an XML exchange. It is not intended to cover every possible scenario but rather the most common information building blocks. In most IEPDs you create, you will need to write an *extension schema* that adds types and properties that are unique to your exchange. NIEM provides detailed guidelines for how to extend the model in a way that maximizes interoperability among NIEM IEPDs.

The NIEM model also does not define specific message types or structures for assembling all of the objects in an exchange. It is up to the creator of the IEPD to write an *exchange schema* to declare the root element and the basic structure of the messages.

Earlier in this article I created a UML model of my exchange and mapped as much of it as I could to NIEM. Figure 14 shows a revised UML model, where the areas in red indicate properties and types that I was unable to map to the base NIEM model.

Figure 14. IEPD model showing extensions



In this case, NIEM met most of my needs. But I will need to create two new schemas:

- An extension schema to define the `Bicycle` type, and the `IsRegistered`, `VehicleCategory` and `CountyCode` properties
- An exchange schema to define the `TheftReport` type (because that is the root) and provide a structure that allows all of the other types to be included in the message

Writing NIEM schemas

NIEM extension and exchange schemas (as well as the generated subset schemas) are written in XML Schema. This article shows examples of NIEM-conformant schemas but does not provide a complete explanation of the XML Schema language. If you are a newcomer to schemas, I recommend the XML Schema Primer.

In addition to the constraints imposed by XML Schema, NIEM adds its own rules that are documented in the NIEM Naming and Design Rules (NDR) 1.3 document. These rules cover, among other things, naming and documentation standards for NIEM components, the kinds of XML Schema constructs that are allowed and disallowed, and approved ways to use and extend NIEM. To be NIEM conformant, the schemas in an IEPD must follow the NDR rules.

Each schema document must have its own target namespace. For my example IEPD, I choose to use `http://datypic.com/theftreport/extension/1.0` (with the prefix `trext:`) as the namespace for the extension schema, and `http://datypic.com/theftreport/exchange/1.0` (with the prefix `tr:`) for the exchange schema.

It is common practice to use a folder structure that reflects the namespace names. Inside the Theft Report IEPD, I will create folders called `extension` and `exchange`, and within each one a subfolder named `1.0` in which I place the respective schema documents.

The beginning of a typical NIEM schema document—in this case, the extension schema for the Theft Report example—is in Example 6.

Example 6. Beginning of a NIEM-conformant schema

```
<xsd:schema version="1.0"
 targetNamespace="http://datypic.com/theftreport/extension/1.0"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:trext="http://datypic.com/theftreport/extension/1.0"
 xmlns:s="http://niem.gov/niem/structures/2.0"
 xmlns:nc="http://niem.gov/niem/niem-core/2.0"
 xmlns:niem-xsd="http://niem.gov/niem/proxy/xsd/2.0"
 xmlns:i="http://niem.gov/niem/appinfo/2.0">
 <xsd:annotation>
  <xsd:documentation>Theft Report extension schema</xsd:documentation>
  <xsd:appinfo>
   <i:ConformantIndicator>true</i:ConformantIndicator>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:import schemaLocation="../../niem/niem-core/2.0/niem-core.xsd"
             namespace="http://niem.gov/niem/niem-core/2.0"/>
 <xsd:import schemaLocation="../../niem/proxy/xsd/2.0/xsd.xsd"
             namespace="http://niem.gov/niem/proxy/xsd/2.0"/>
 <xsd:import schemaLocation="../../niem/structures/2.0/structures.xsd"
             namespace="http://niem.gov/niem/structures/2.0"/>
 <xsd:import schemaLocation="../../niem/appinfo/2.0/appinfo.xsd"
             namespace="http://niem.gov/niem/appinfo/2.0"/>

</xsd:schema>
```

A NIEM schema document must contain an `xsd:annotation` element that has a description (in `xsd:documentation`) and an indicator that it is NIEM conformant (in `xsd:appinfo`).

As with any schema, it declares and imports all of the namespaces that it needs to reference directly. It is also required to import the appinfo schema on the last line of Example 6, which declares the elements used inside the `xsd:appinfo` element.

Complete extension and exchange schema documents that include all of the listings in this article are available from the Artifacts page.

Extension schemas

Depending on the complexity of your IEPD, you might have one extension schema or many. Some IEPD developers choose to break extension schemas into multiple documents by subject area to allow them to reuse the schemas more granularly in various exchanges. Others choose to put components that might be versioned more frequently—for example, code lists—into a separate schema document.

For the Theft Report example, because it is simple, I choose to create one extension schema. After the beginning of the schema in Example 6, I need to define types and declare elements for my custom components. There are several ways of extending NIEM, and I use a different method for each of my customizations.

Using substitution groups

Perhaps the easiest way to extend NIEM is through the use of *substitution groups* which allow you to declare your own element and specify that it is substitutable for a NIEM element. This means that it can appear anywhere the NIEM element is allowed. You can use this method when there is a semantically equivalent element in the NIEM model, but it does not quite meet your needs. For example, in my model, I have a `CountyCode` property that is semantically equivalent to the NIEM abstract element `nc:LocationCounty` that appears inside an address. There are already two elements in the substitution group that are part of the NIEM core model, but they don't meet my needs: `nc:LocationCountyCode` uses a different code list, and `nc:LocationCountyName` is intended for a spelled-out name rather than a code. Instead, I declare a new element, `trext:LocationCountyCode`, that uses my own code list.

Example 7 shows the element declaration for `trext:LocationCountyCode`. To indicate that it is substitutable for `nc:LocationCounty`, I use a `substitutionGroup` attribute.

Example 7. Declaration of the `trext:LocationCounty` element and related types

```
<xsd:element name="LocationCountyCode" type="trext:CountyCodeType"
             substitutionGroup="nc:LocationCounty">
 <xsd:annotation>
  <xsd:documentation>A county code.</xsd:documentation>
 </xsd:annotation>
</xsd:element>

<xsd:simpleType name="CountyCodeSimpleType">
 <xsd:annotation>
  <xsd:documentation>A data type for a county code.</xsd:documentation>
  <xsd:appinfo>
   <i:Base i:namespace="http://niem.gov/niem/structures/2.0" i:name="Object"/>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:restriction base="xsd:token">
  <xsd:enumeration value="A">
   <xsd:annotation>
    <xsd:documentation>Ascot County</xsd:documentation>
   </xsd:annotation>
  </xsd:enumeration>
  <xsd:enumeration value="B">
   <xsd:annotation>
    <xsd:documentation>Burke County</xsd:documentation>
   </xsd:annotation>
  </xsd:enumeration>
  <xsd:enumeration value="C">
   <xsd:annotation>
    <xsd:documentation>Cross County</xsd:documentation>
   </xsd:annotation>
  </xsd:enumeration>
 </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="CountyCodeType">
 <xsd:annotation>
  <xsd:documentation>A data type for a county code.</xsd:documentation>
  <xsd:appinfo>
   <i:Base i:namespace="http://niem.gov/niem/structures/2.0" i:name="Object"/>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:simpleContent>
  <xsd:extension base="trext:CountyCodeSimpleType">
   <xsd:attributeGroup ref="s:SimpleObjectAttributeGroup"/>
  </xsd:extension>
 </xsd:simpleContent>
</xsd:complexType>
```

Example 7 also shows the two type definitions that support the `trext:LocationCountyCode` element. First, a simple type is defined that has `xsd:enumeration` elements for each of the code values. Then, a complex type is defined based on the simple type. The complex type adds universal attributes like `s:id` that are allowed on all NIEM objects, through a reference to `s:SimpleObjectAttributeGroup`.

Creating entirely new types

Another method of NIEM extension is to create a whole new type. In my model, `Bicycle` doesn't have an equivalent at all in the NIEM model, so I need to create a new element and a new corresponding complex type. Whenever you add a new type, you should consider whether it is a specialization of an existing NIEM type—for example, `nc:ActivityType`, `nc:PersonType`, or `nc:ItemType`. For `Bicycle`, I decide that it should be based on `nc:ConveyanceType`, because it represents a means of transportation, which is appropriate for a bicycle. Also, `nc:ConveyanceType` already has most of the properties I need, such as serial number and description.

As with the previous method of extension, I have to define both a new element, `trext:Bicycle`, and a type, `trext:BicycleType`. Example 8 shows these definitions.

Example 8. Declaration of the `trext:Bicycle` element and related type

```
<xsd:element name="Bicycle" type="trext:BicycleType">
 <xsd:annotation>
  <xsd:documentation>A bicycle.</xsd:documentation>
 </xsd:annotation>
</xsd:element>

<xsd:complexType name="BicycleType">
 <xsd:annotation>
  <xsd:documentation>A data type for a bicycle.</xsd:documentation>
 </xsd:annotation>
 <xsd:complexContent>
  <xsd:extension base="nc:ConveyanceType">
   <xsd:sequence>
    <xsd:element ref="trext:BicycleRegisteredIndicator" minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

The type definition for `trext:BicycleType` indicates that it extends `nc:ConveyanceType`. Note that if you are creating a type that is truly new—that is, not based on any concept already in NIEM—you must base your type on `s:ComplexObjectType`, which is the root of all complex types in NIEM.

In `trext:BicycleType`, I reference a `trext:BicycleRegisteredIndicator` element that I have to declare separately. All elements, attributes and types in NIEM schemas are global, uniquely named, top-level components. Example 9 shows the declaration of the `trext:BicycleRegisteredIndicator` element.

Example 9. Declaration of the `trext:BicycleRegisteredIndicator` element

```
<xsd:element name="BicycleRegisteredIndicator" type="niem-xsd:boolean">
 <xsd:annotation>
  <xsd:documentation>Whether a bicycle is registered.</xsd:documentation>
 </xsd:annotation>
</xsd:element>
```

Unlike `trext:LocationCountyCode`, which had its own code list type, `trext:BicycleRegisteredIndicator` has a type that corresponds to one of the XML Schema built-in types, `boolean`. However, instead of giving it the built-in type `xsd:boolean`, I use `niem-xsd:boolean`. This complex type, defined in the "proxy" schema `xsd.xsd`, specifies that the element contains an `xsd:boolean` value but also allows the universal NIEM attributes like `s:id`.

Adding properties to existing types

Another extension situation is where you have a complex type that is semantically equivalent to a NIEM type but you need to alter or add to it in some way. In my model, the `MotorVehicle` class is equivalent to the NIEM `nc:VehicleType` but it needs an extra property, `VehicleCategoryCode`. When doing the mapping, I looked at `nc:ItemCategoryText` as a possible mapping candidate but decided that it was too general. In fact, the `VehicleCategoryCode` property represents a classification of vehicles used for tax purposes, so I decide to call the element `trext:VehicleTaxClassCode`.

The required XML Schema definitions are similar to the `Bicycle` extension. Example 10 shows how I declare a new element—`trext:Vehicle`—and a new complex type—`trext:VehicleType`—that extends `nc:VehicleType`.

Example 10. Declaration of the `trext:Vehicle` element and related type

```
<xsd:element name="Vehicle" type="trext:VehicleType">
 <xsd:annotation>
  <xsd:documentation>A motor vehicle.</xsd:documentation>
 </xsd:annotation>
</xsd:element>

<xsd:complexType name="VehicleType">
 <xsd:annotation>
  <xsd:documentation>A data type for a motor vehicle.</xsd:documentation>
 </xsd:annotation>
 <xsd:complexContent>
  <xsd:extension base="nc:VehicleType">
   <xsd:sequence>
    <xsd:element ref="trext:VehicleTaxClassCode" minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Adding new objects with existing types

Sometimes, you are happy with the NIEM types, but you want to use names that are more specific or relevant to your exchange. In my model, I decided that the `Theft` class corresponded to the NIEM `nc:ActivityType`. However, I'm not completely satisfied with calling my element `nc:Activity`, because it is too general and not descriptive enough. In this case, I choose to declare a new element, named `trext:Theft`, but give it the existing type `nc:ActivityType` rather than define a new type. Example 11 shows the element declaration.

Example 11. Declaration of the `trext:Theft` element

```
<xsd:element name="Theft" type="nc:ActivityType">
 <xsd:annotation>
  <xsd:documentation>A theft incident.</xsd:documentation>
 </xsd:annotation>
</xsd:element>
```

Exchange schemas

Exchange schemas contain definitions that are unique to a message type or group of message types. This generally includes only the root element and its type and possibly some structural elements that form the basic framework of the message. Typically, an exchange schema is IEPD specific, while an extension schema might be shared across several IEPDs.

You are not required to have separate exchange and extension schemas; you can put all of your extensions in the same schema document. You can also have multiple exchange schemas in order to represent different message types or groups of different message types.

Exchange schemas follow all of the same rules described previously for extension schemas. For example, they must have their own target namespace and must have annotations.

In the Theft Report example, the exchange schema will contain the `tr:TheftReport` element, because that is the root, and its type. It will contain a `TheftReportDate`, which is shown in the model. But more importantly, the `tr:TheftReport` element will be what brings together all of the objects and associations defined in the exchange. The element and type for TheftReport are in Example 12.

Example 12. Declaration of the `tr:TheftReport` element and related type

```
<xsd:element name="TheftReport" type="tr:TheftReportType">
 <xsd:annotation>
  <xsd:documentation>A theft report.</xsd:documentation>
 </xsd:annotation>
</xsd:element>

<xsd:complexType name="TheftReportType">
 <xsd:annotation>
  <xsd:documentation>A data type for a theft report.</xsd:documentation>
  <xsd:appinfo>
   <i:Base i:namespace="http://niem.gov/niem/structures/2.0" i:name="Object"/>
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:complexContent>
  <xsd:extension base="s:ComplexObjectType">
   <xsd:sequence>
    <xsd:element ref="tr:TheftReportDate" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="trext:Theft" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="nc:ActivityConveyanceAssociation" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="trext:Vehicle" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="trext:Bicycle" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="j:ActivityLocationAssociation" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="nc:Location" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="j:ActivityVictimAssociation" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="j:Victim" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="j:ActivityWitnessAssociation" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="j:Witness" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="nc:Person" minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

Note that the object and association elements are all siblings of each other. This is typical of a NIEM message, where associations between objects are separate components that reference the related objects through `s:ref` attributes.

A snippet of a message that shows the association between a theft and its location is in Example 13. The objects, `trext:Theft` and `nc:Location`, are siblings, and each has an `s:id` attribute giving it a unique identifier. The association, `j:ActivityLocationAssociation`, is another sibling that links the two objects using child elements with `s:ref` attributes.

Example 13. Sample instance showing association

```
<trext:Theft s:id="T1">
 <nc:ActivityDate>
  <nc:DateTime>2006-05-04T08:15:00</nc:DateTime>
 </nc:ActivityDate>
</trext:Theft>

<j:ActivityLocationAssociation>
 <nc:ActivityReference s:ref="T1"/>
 <nc:LocationReference s:ref="L1"/>
</j:ActivityLocationAssociation>

<nc:Location s:id="L1">
 <nc:LocationAddress>
  <nc:StructuredAddress>
   <nc:LocationStreet>
    <nc:StreetFullText>123 Main Street</nc:StreetFullText>
   </nc:LocationStreet>
   <nc:LocationCityName>Big City</nc:LocationCityName>
   <trext:LocationCountyCode>A</trext:LocationCountyCode>
   <nc:LocationStateUSPostalServiceCode>MI</nc:LocationStateUSPostalServiceCode>
   <nc:LocationPostalCode>49684</nc:LocationPostalCode>
  </nc:StructuredAddress>
 </nc:LocationAddress>
</nc:Location>
```

Another option for expressing relationships among objects is *containment*, where one object is the parent of another object. For example, it is hypothetically possible to create a brand new `TheftType` that contains within it a person and a location or a reference to a person or a location. However, this is not the recommended approach to using NIEM. Separating associations makes the delineation of objects clearer, reduces problems with recursion, and is more adapted to many-to-many relationships.

Naming and documenting NIEM components

You might have noticed some consistency in the names used in the examples. NIEM imposes certain rules for names:

- A name has an *object term* and a *property term*, and, if it is a simple element, a *representation term*. For example, in the name `BicycleRegisteredIndicator`, `Bicycle` is the object term, `Registered` is the property term, and `Indicator` is the representation term. It can also have optional qualifier terms.
- There is a specific set of approved representation terms, among which are `Indicator`, `Code`, `Date`, `Text`, `Value`, and `Quantity`.
- All names use camel case (uppercasing the first letter of each word) rather than separator characters.
- Attribute names start with a lowercase letter, while element and type names start with an uppercase letter.
- All types have the word `Type` at the end of their name.

There are also rules that govern the documentation of NIEM components. All schemas, elements, attributes, types, and enumerations must have definitions, and they must start with one of an approved set of beginning phrases, such as "A name of", or "A relationship".

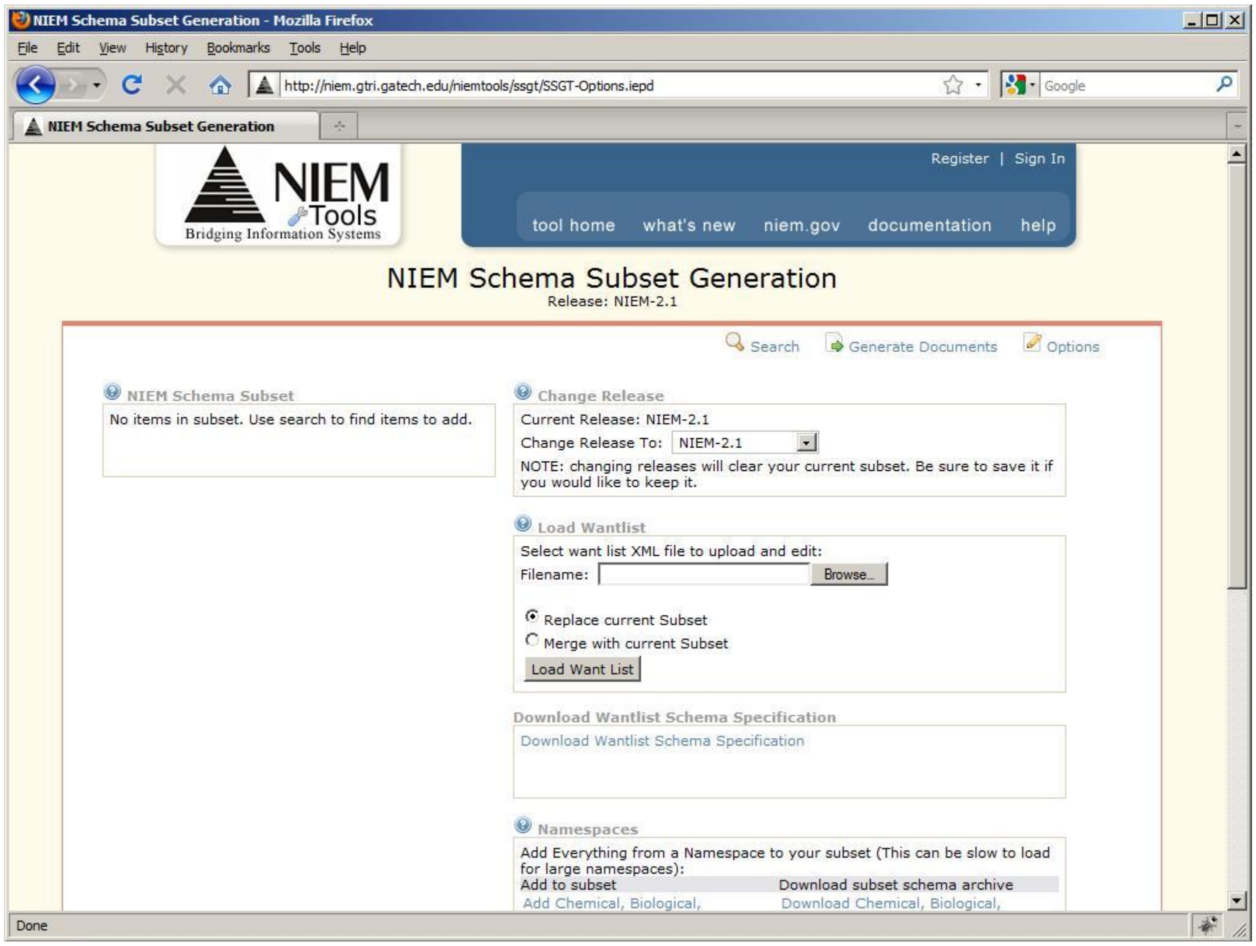
This is just a sampling of the rules; you can find a complete listing of the NIEM rules in the NDR.

Modifying the subset

As you build your extension and exchange schemas, you might need additional components from NIEM that you did not include in your subset. For example, `trext:BicycleRegisteredIndicator` is of type `niem-xsd:boolean`, a type that was not in my original subset.

Fortunately, it is easy to modify your subset using the Schema Subset Generation Tool (SSGT). From the main page of the SSGT, click **Options** in the upper right corner. This will bring up a page shown in Figure 15.

Figure 15. SSGT Options page


NIEM Schema Subset Generation - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://niem.gtri.gatech.edu/niemtools/ssgt/SSGT-Options.iepd
NIEM Schema Subset Generation
NIEM Tools Bridging Information Systems
Register | Sign In
tool home what's new niem.gov documentation help
Search Generate Documents Options
Done

NIEM Schema Subset Generation
Release: NIEM-2.1

NIEM Schema Subset
No items in subset. Use search to find items to add.

Change Release
Current Release: NIEM-2.1
Change Release To: NIEM-2.1
NOTE: changing releases will clear your current subset. Be sure to save it if you would like to keep it.

Load Wantlist
Select want list XML file to upload and edit:
Filename: Browse...
[✓ Replace current Subset]
Merge with current Subset
Load Want List

Download Wantlist Schema Specification
Download Wantlist Schema Specification

Namespaces
Add Everything from a Namespace to your subset (This can be slow to load for large namespaces):

Add to subset	Download subset schema archive
Add Chemical, Biological,	Download Chemical, Biological,

In the section called Load Wantlist, fill in (or browse for) your wantlist.xml file name, and then click **Load Want List**. Doing so brings up your subset in the left pane. You can then click **Search** and use the right pane to search and add components to your NIEM subset. When you are done, regenerate the subset.

When working with extensions, you sometimes want to use the SSGT to look for types rather than properties. To find `niem-xsd:boolean` I can't use the default search on properties, because that only finds element and attribute names, not type names. To specifically look for types, choose **Type** from the **Search for a** drop-down menu on the search page of the SSGT.

Documenting your mapping in the CMT

Be sure to document your extensions in the Component Mapping Template (CMT) that was described earlier in this article. At a minimum, you should fill in the XPath expressions for your extension elements. Table 6 shows the extensions I filled in for the `MotorVehicle` and `Bicycle` classes.

Table 6. XPath mapping for extensions

Source type	Source property	...	Ext?	XPath
MotorVehicle		...	Y	trext:Vehicle
MotorVehicle	LicensePlate	...	Y	trext:Vehicle/nc:ConveyanceRegistrationPlateIdentification/nc:IdentificationID
MotorVehicle	VehicleCategory	...	Y	trext:Vehicle/trext:VehicleTaxClassCode
Bicycle		...	Y	trext:Bicycle
Bicycle	IsRegistered	...	Y	trext:Bicycle/trext:BicycleRegisteredIndicator

Some NIEM practitioners create more formal CMTs that have separate columns indicating the kind of extension, the base types and elements, and the level of semantic alignment. For my CMT, I chose to take a looser approach to defining the dependencies by including this information in a Comments column. The final Theft Report CMT is available from the Artifacts page.

Summary

In this section, I described the process of extending NIEM. I explained the role of extension and exchange schemas, and showed the various methods of adding new elements and types based on NIEM components. The majority of the work in creating a NIEM IEPD is now complete. The next section describes the last and final step, assembling the final IEPD.

Step 4: Assemble the IEPD

An IEPD is a collection of documents describing a NIEM exchange. It typically includes schemas, samples, documentation of various kinds, and rendering instructions. It also includes several NIEM-specific artifacts that are required in a NIEM-conformant exchange, such as a metadata document and a catalog file.

In this article, I have been working through a simple example of a Theft Report IEPD. Now that my UML model, CMT and schemas are complete, I have a few steps left to finish my IEPD:

1. Create more documentation-related artifacts to further explain the exchange.

2. Assemble my IEPD in a NIEM-conformant way, which involves using the NIEM "Work with IEPDs" tool to upload my artifacts and generate additional documentation files.
3. Validate my IEPD for completeness and NIEM conformance using the NIEM Conformance Validation Tool.
4. Publish my IEPD so that it can be discovered by other users.

IEPD documentation

To be NIEM conformant, an IEPD must contain—at a minimum—some form of master documentation and a change log describing the changes since the last version. You can include any documentation that might typically accompany a software application in the IEPD (either in the master documentation or as separate files), such as:

- UML models (sequence diagrams, use cases, class diagrams)
- A CMT
- Business rules (that is, constraints on the data that are not expressed in the schemas or in the model)
- Requirements definitions
- Testing and/or conformance statements
- Memoranda of understanding and letters of endorsement

The documentation artifacts I have developed so far for the Theft Report IEPD are a UML model and a CMT. I need to add master documentation that describes in general terms the purpose and structure of the exchange. I also add a change log, which is essentially empty, because this is the first version of the exchange. See the Artifacts page for a complete IEPD that contains these documents.

Creating sample documents

Sample XML documents are an important part of an IEPD. It's difficult to conceptualize XML documents based solely on a schema, particularly if you're talking about a complex set of interrelated schema documents. Using samples makes it clear which element is the root element in any given document type and provides examples of typical data for each element.

Include at least one sample for each different root element in the exchange schema. If you use a document type for multiple purposes, you should provide a sample for each purpose. It's helpful to create one sample that contains every possible element in order to test the schemas and any software written to process the documents. For a complex exchange, it may also be useful to create a typical sample that contains what is likely to be included in any given document.

Many XML editors generate sample documents for you, but those documents generally contain meaningless data and don't have correctly related ID and IDREF values used in the associations. You should edit any generated samples to make them more typical and meaningful. I also recommend validating your samples using multiple processors because of differences in the way various processors handle schema location hints.

For the Theft Report IEPD, I only have one possible root element (`tr:TheftReport`). Example 14 shows the beginning of one complete sample that I created. It has at least one of each element type allowed in the exchange and has representative values for the data.

Example 14. Beginning of a sample document

```
<?xml-stylesheet type="text/xsl" href="theftreport.xsl" ?>
<tr:TheftReport xmlns:tr="http://datypic.com/theftreport/exchange/1.0"
 xmlns:nc="http://niem.gov/niem/niem-core/2.0"
 xmlns:trext="http://datypic.com/theftreport/extension/1.0"
 xmlns:j="http://niem.gov/niem/domains/jxdm/4.1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:s="http://niem.gov/niem/structures/2.0"
 xsi:schemaLocation="http://datypic.com/theftreport/exchange/1.0
                    ../schema/exchange/1.0/theftreport-exchange.xsd">
 <tr:TheftReportDate>2006-05-05</tr:TheftReportDate>
 <trext:Theft s:id="T1">
  <nc:ActivityDate>
   <nc:DateTime>2006-05-04T08:15:00</nc:DateTime>
  </nc:ActivityDate>
 </trext:Theft>
 <trext:Theft s:id="T2">
  <nc:ActivityDate>
   <nc:DateTime>2006-05-04T09:14:00</nc:DateTime>
  </nc:ActivityDate>
 </trext:Theft>
 <nc:ActivityConveyanceAssociation>
  <nc:ActivityReference s:ref="T1"/>
  <nc:ConveyanceReference s:ref="V1"/>
 </nc:ActivityConveyanceAssociation>
 <nc:ActivityConveyanceAssociation>
  <nc:ActivityReference s:ref="T2"/>
  <nc:ConveyanceReference s:ref="B1"/>
 </nc:ActivityConveyanceAssociation>
 <trext:Vehicle s:id="V1">
  <nc:ItemDescriptionText>2001 Subaru Outback</nc:ItemDescriptionText>
   <nc:ItemSerialIdentification>
    <nc:IdentificationID>123455234234</nc:IdentificationID>
   </nc:ItemSerialIdentification>
   <nc:VehicleColorPrimaryCode>SIL</nc:VehicleColorPrimaryCode>
   <nc:ConveyanceRegistrationPlateIdentification>
    <nc:IdentificationID>BGE112</nc:IdentificationID>
   </nc:ConveyanceRegistrationPlateIdentification>
   <trext:VehicleTaxClassCode>4</trext:VehicleTaxClassCode>
  </trext:Vehicle>

    <!-- ... -->
  </tr:TheftReport>
```

Creating rendering instructions

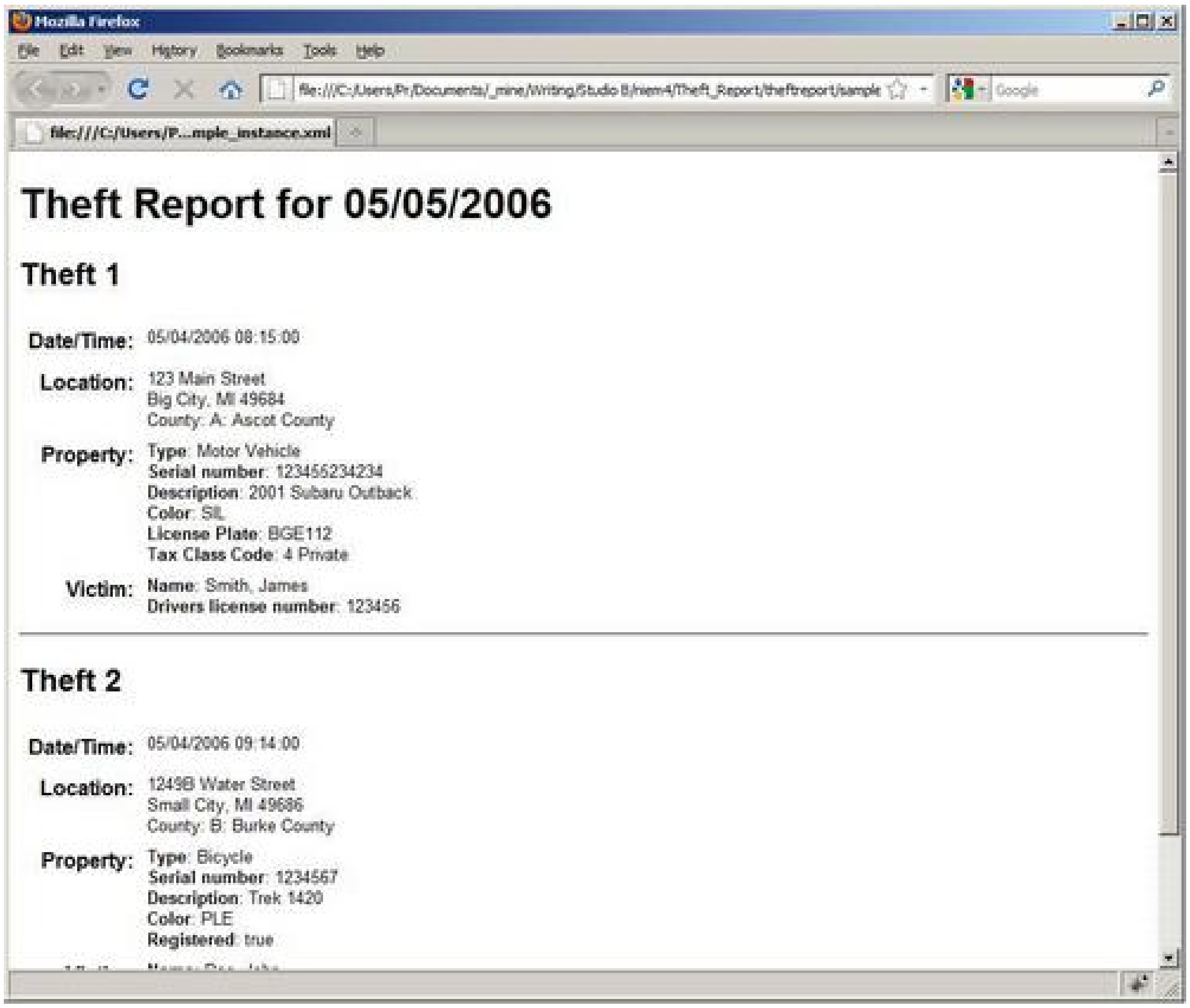
Rendering instructions are another useful tool to help users and implementers of the IEPD conceptualize the exchange. It's difficult to look at a complex XML document and understand its content—particularly in a typical NIEM document, which has a lot of structural elements and associations that connect different parts of a document. Some users get caught up in the complexity of the structure and the names of elements rather than focusing on the actual content.

If you write rendering instructions that turn the XML data into human-readable HTML, you can alleviate this problem. Associated objects can be rejoined and displayed together on the page, extraneous structure can be ignored, and cryptic code list values can be turned into their readable equivalents.

Rendering of XML documents is typically done with an XSLT stylesheet. I generally recommend sticking to XSLT version 1.0 for simple HTML rendering, because that version is supported in major browsers. Providing an XSLT stylesheet in the same directory as the samples allows users to double-click a sample (or open it in an XML editor) and see the rendered HTML version. The processing instruction (starting with `<?xml-stylesheet`) on the first line of Example 14 indicates which XSLT stylesheet to use.

For the Theft Report IEPD, I created a fairly simple XSLT 1.0 stylesheet (included in the IEPD downloadable from the Artifacts page) that presents the Theft Report in a user-friendly way, as in Figure 16. The XSLT joins each theft with its associated location, vehicle, victim, and witness to present the data logically.

Figure 16. Rendered sample


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Users/Pr/Documents/_mine/Writing/Studio B/niem4/Theft_Report/theftreport/sample
Google
file:///C:/Users/P...mple_instance.xml

Theft Report for 05/05/2006

Theft 1

Date/Time: 05/04/2006 08:15:00

Location: 123 Main Street
Big City, MI 49684
County: A: Ascot County

Property: **Type**: Motor Vehicle
Serial number: 123455234234
Description: 2001 Subaru Outback
Color: SIL
License Plate: BGE112
Tax Class Code: 4 Private

Victim: **Name**: Smith, James
Drivers license number: 123456

Theft 2

Date/Time: 05/04/2006 09:14:00

Location: 1249B Water Street
Small City, MI 49686
County: B: Burke County

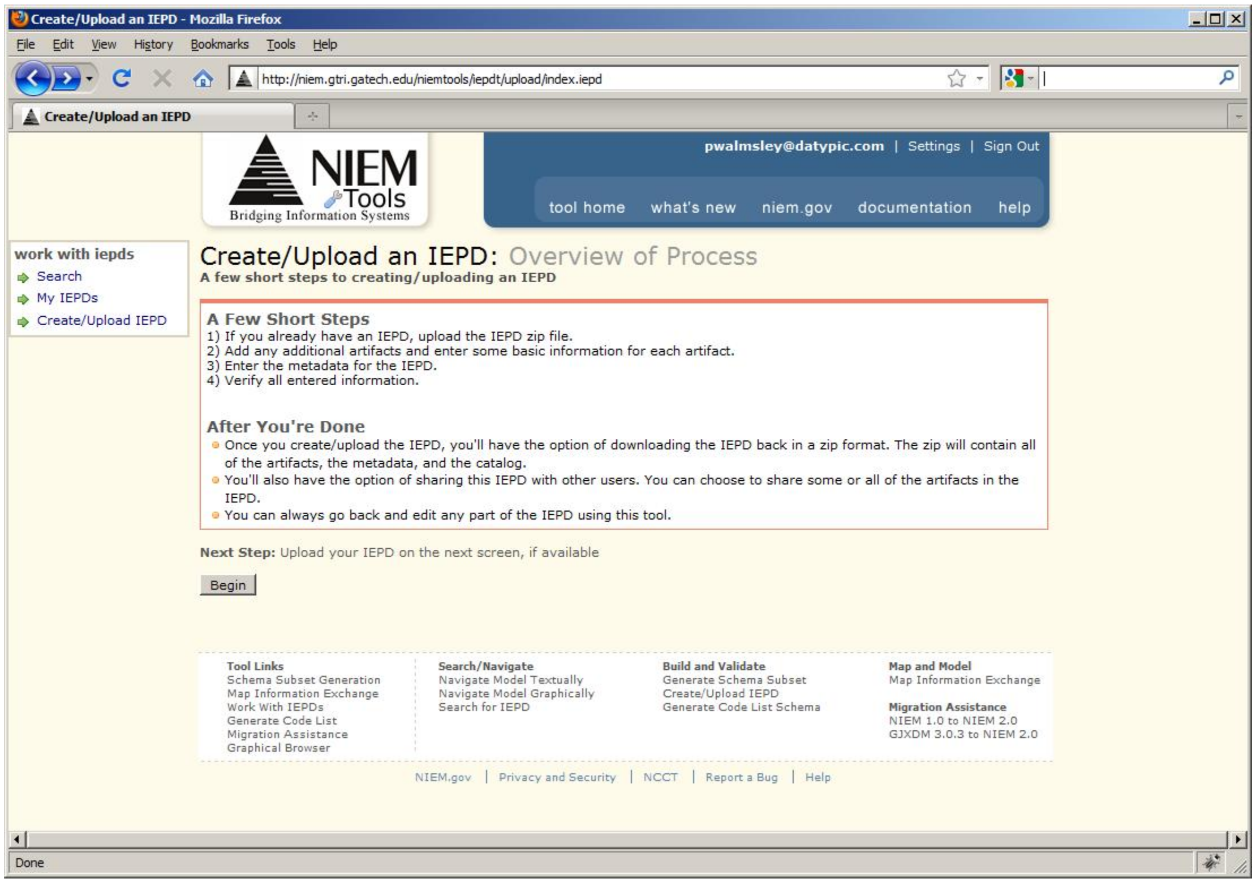
Property: **Type**: Bicycle
Serial number: 1234567
Description: Trek 1420
Color: PLE
Registered: true

Assembling an IEPD

NIEM provides a Work with IEPDs Tool for gathering your IEPD artifacts and assembling the complete package. Although it is possible to assemble an IEPD by hand, using the tool is generally easier, because it allows you to automatically generate two NIEM-specific artifacts that are required for NIEM conformance: the metadata document and the catalog file, which serves as a table of contents. It also results in an IEPD that has a more consistent file structure.

To use the tool, choose **Create/Upload IEPD** from the menu on the left side of the page. Doing so takes you to the page in Figure 17.

Figure 17. Create/Upload an IEPD page


Create/Upload an IEPD - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://niem.gtri.gatech.edu/niemtools/iepdt/upload/index.iepd
Create/Upload an IEPD
NIEM Tools
Bridging Information Systems
pwalmsley@datypic.com | Settings | Sign Out
tool home what's new niem.gov documentation help
work with iepds
Search
My IEPDs
Create/Upload IEPD

Create/Upload an IEPD: Overview of Process

A few short steps to creating/uploading an IEPD

A Few Short Steps

1) If you already have an IEPD, upload the IEPD zip file.
2) Add any additional artifacts and enter some basic information for each artifact.
3) Enter the metadata for the IEPD.
4) Verify all entered information.

After You're Done

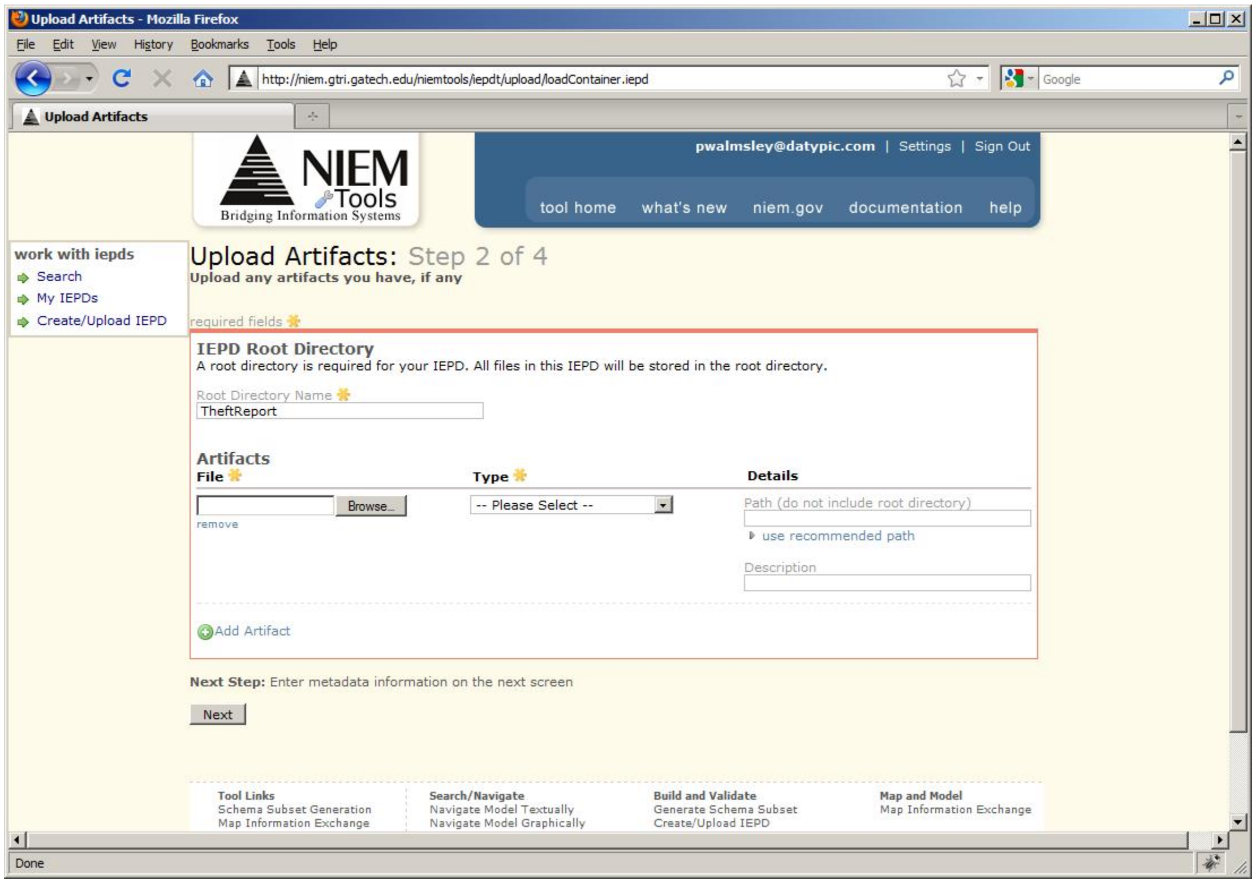
- Once you create/upload the IEPD, you'll have the option of downloading the IEPD back in a zip format. The zip will contain all of the artifacts, the metadata, and the catalog.
- You'll also have the option of sharing this IEPD with other users. You can choose to share some or all of the artifacts in the IEPD.
- You can always go back and edit any part of the IEPD using this tool.

Next Step: Upload your IEPD on the next screen, if available

After you click **Begin**, you will be asked whether you have an existing IEPD zip file that you want to upload. If you already have a zipped set of IEPD artifacts, you can click **yes** and upload the zip file. When working with the tool for the Theft Report example, I clicked **no** to upload each file individually.

When you click **no**, the tool displays the page in Figure 18. This page allows you to specify a root directory name. For the Theft Report, I choose TheftReport as the root directory. Also on this page, to add individual artifacts to the IEPD, click the **Add Artifact** link.

Figure 18. Upload Artifacts page


Upload Artifacts - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://niem.gtri.gatech.edu/niemtools/iepdt/upload/loadContainer.iepd
Upload Artifacts
NIEM Tools Bridging Information Systems
pwalmsley@datypic.com | Settings | Sign Out
tool home what's new niem.gov documentation help
work with iepds
Search
My IEPDs
Create/Upload IEPD

Upload Artifacts: Step 2 of 4

Upload any artifacts you have, if any

required fields

IEPD Root Directory

A root directory is required for your IEPD. All files in this IEPD will be stored in the root directory.

Root Directory Name: TheftReport

Artifacts

File	Type	Details
Browse... remove	-- Please Select --	Path (do not include root directory); use recommended path; Description

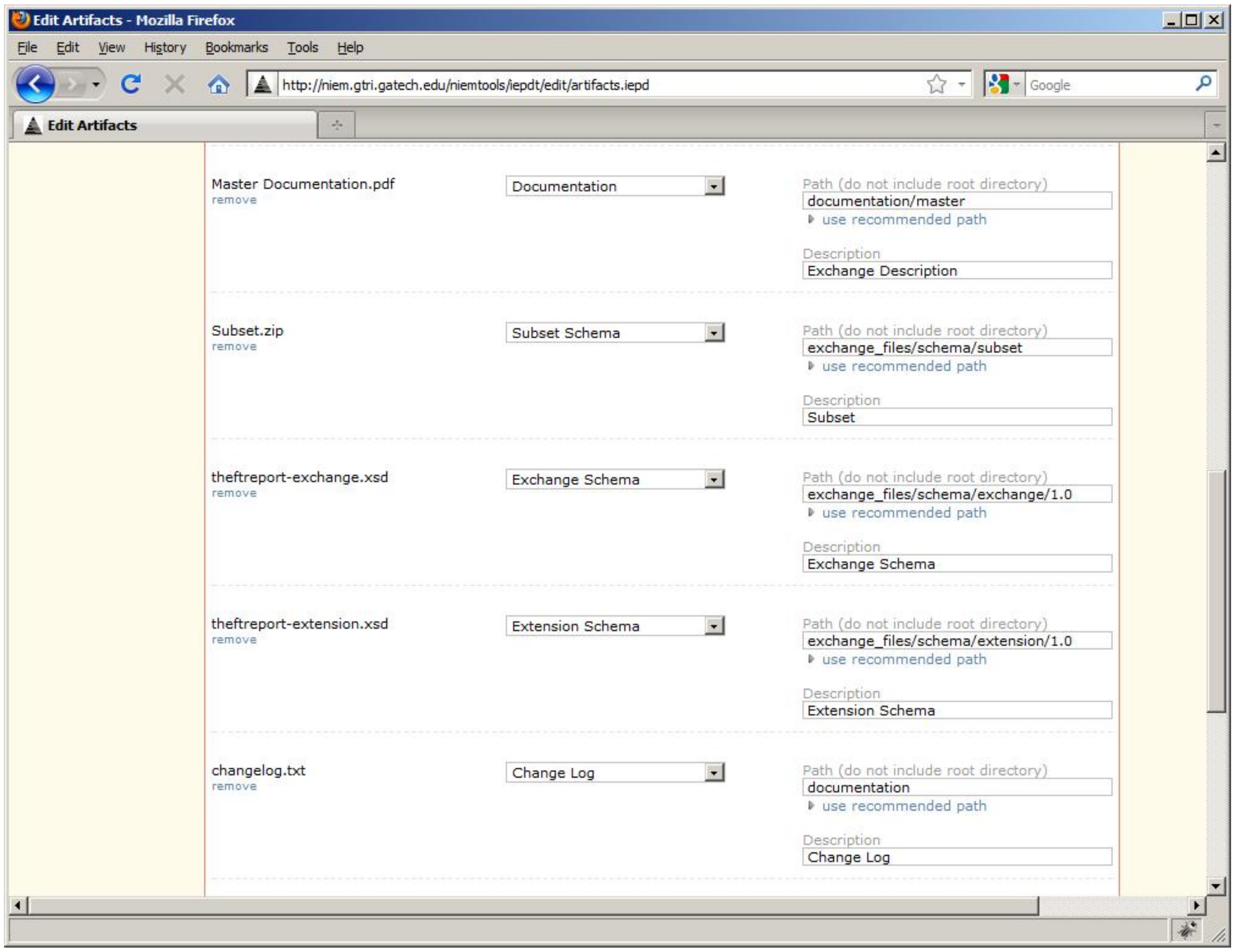
Add Artifact

Next Step: Enter metadata information on the next screen

You add the extension and exchange schemas, wantlist, subset, sample document, rendering instructions, CMT, UML model, change log, and master documentation using this page. Note that you should zip the subset schemas into one file; when the tool regenerates the IEPD, it will be unzipped.

For each artifact you add, you choose a type. Doing so is important, because it affects the way the catalog file is generated and the default directories used for the components. You can also choose a directory path on this page. For consistency, click **use recommended path** for each artifact, and the tool fills in the path. The only change I made to the recommended path was to include a `1.0` subdirectory for the exchange and extensions schemas, because I wanted the directory structure to mirror the namespace name. I also added a description for each artifact. Figure 19 shows the results.

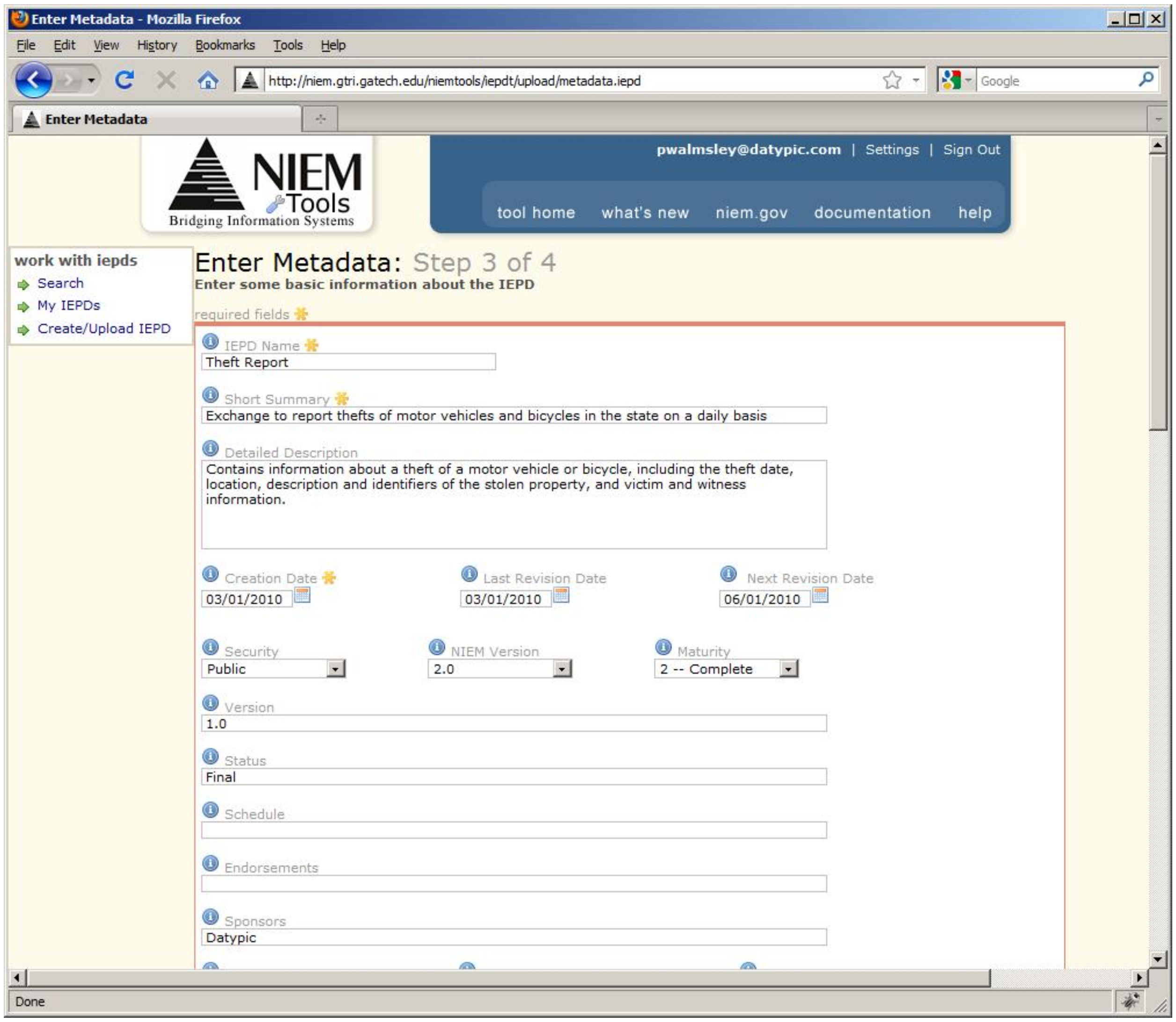
Figure 19. Added artifacts


Edit Artifacts - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://niem.gtri.gatech.edu/niemtools/iepdt/edit/artifacts.iepd
Google
Edit Artifacts

Master Documentation.pdf remove	Documentation	Path (do not include root directory): documentation/master; use recommended path; Description: Exchange Description
Subset.zip remove	Subset Schema	Path (do not include root directory): exchange_files/schema/subset; use recommended path; Description: Subset
theftreport-exchange.xsd remove	Exchange Schema	Path (do not include root directory): exchange_files/schema/exchange/1.0; use recommended path; Description: Exchange Schema
theftreport-extension.xsd remove	Extension Schema	Path (do not include root directory): exchange_files/schema/extension/1.0; use recommended path; Description: Extension Schema
changelog.txt remove	Change Log	Path (do not include root directory): documentation; use recommended path; Description: Change Log

When you are done adding artifacts, click **Next** to go to the **Enter Metadata** page in Figure 20. Here, you provide metadata about the IEPD, such as name, description, version, organization, and point of contact. The tool uses this information to generate an XML document called metadata.xml that is included with the IEPD.

Figure 20. Enter Metadata page


Enter Metadata - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://niem.gtri.gatech.edu/niemtools/iepdt/upload/metadata.iepd
Google
Enter Metadata
NIEM Tools
Bridging Information Systems
pwalmsley@datypic.com | Settings | Sign Out
tool home what's new niem.gov documentation help
work with iepds
Search
My IEPDs
Create/Upload IEPD
Done

Enter Metadata: Step 3 of 4

Enter some basic information about the IEPD

required fields

IEPD Name: Theft Report

Short Summary: Exchange to report thefts of motor vehicles and bicycles in the state on a daily basis

Detailed Description: Contains information about a theft of a motor vehicle or bicycle, including the theft date, location, description and identifiers of the stolen property, and victim and witness information.

Creation Date	Last Revision Date	Next Revision Date
03/01/2010	03/01/2010	06/01/2010

Security	NIEM Version	Maturity
Public	2.0	2 -- Complete

Version: 1.0

Status: Final

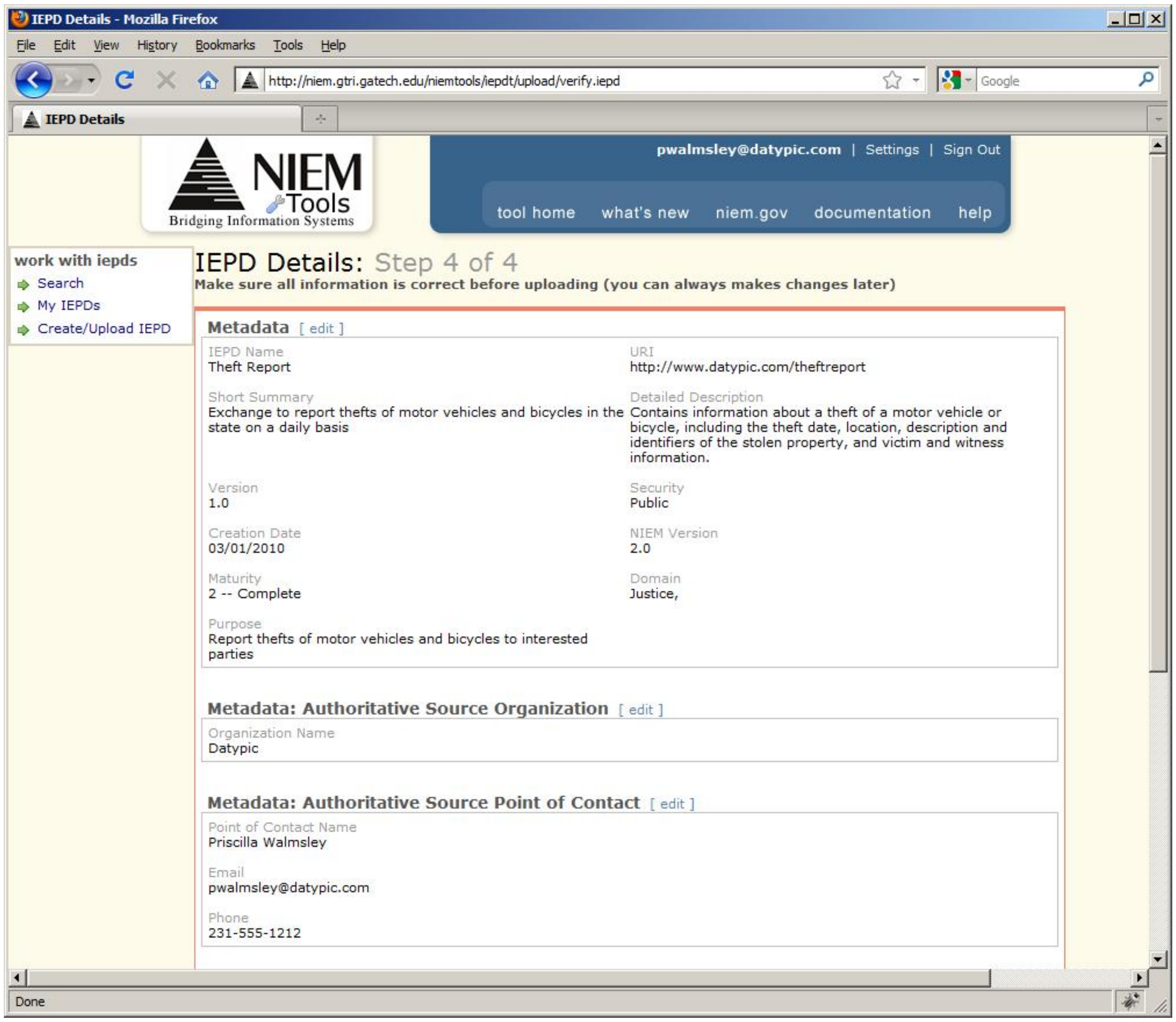
Schedule:

Endorsements:

Sponsors: Datypic

One thing that is slightly confusing about this page is that the fields required for NIEM conformance are not all marked with yellow asterisks. Later, when you validate your IEPD, you will be informed if you missed any required metadata. In general, it's best to fill in as many as are relevant. When you're done with the metadata, click **Next** to go to the summary page in Figure 21.

Figure 21. IEPD summary page


IEPD Details - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://niem.gtri.gatech.edu/niemtools/iepdt/upload/verify.iepd
IEPD Details
NIEM Tools
Bridging Information Systems
pwalmsley@datypic.com | Settings | Sign Out
tool home what's new niem.gov documentation help
work with iepds
Search
My IEPDs
Create/Upload IEPD
Done

IEPD Details: Step 4 of 4
Make sure all information is correct before uploading (you can always makes changes later)

Metadata [edit]

IEPD Name: Theft Report	URI: http://www.datypic.com/theftreport
Short Summary: Exchange to report thefts of motor vehicles and bicycles in the state on a daily basis	Detailed Description: Contains information about a theft of a motor vehicle or bicycle, including the theft date, location, description and identifiers of the stolen property, and victim and witness information.
Version: 1.0	Security: Public
Creation Date: 03/01/2010	NIEM Version: 2.0
Maturity: 2 -- Complete	Domain: Justice,
Purpose: Report thefts of motor vehicles and bicycles to interested parties	

Metadata: Authoritative Source Organization [edit]

Organization Name
Datypic

Metadata: Authoritative Source Point of Contact [edit]

Point of Contact Name
Priscilla Walmsley

Email
pwalmsley@datypic.com

Phone
231-555-1212

From here, you can click **Validate IEPD** to determine whether your IEPD contains all the necessary metadata. When it is validated, click **Upload IEPD** to save the IEPD in your profile so that you can retrieve it under **My IEPDs** later. Note that uploading your IEPD does not automatically make it visible to other users of the Web site.

Finally, to actually generate the IEPD, click **Download** from the **IEPD Successfully Uploaded** page. You can download, modify, or delete any of your IEPDs at any time by clicking **My IEPDs** from the menu on the left side of the tool pages.

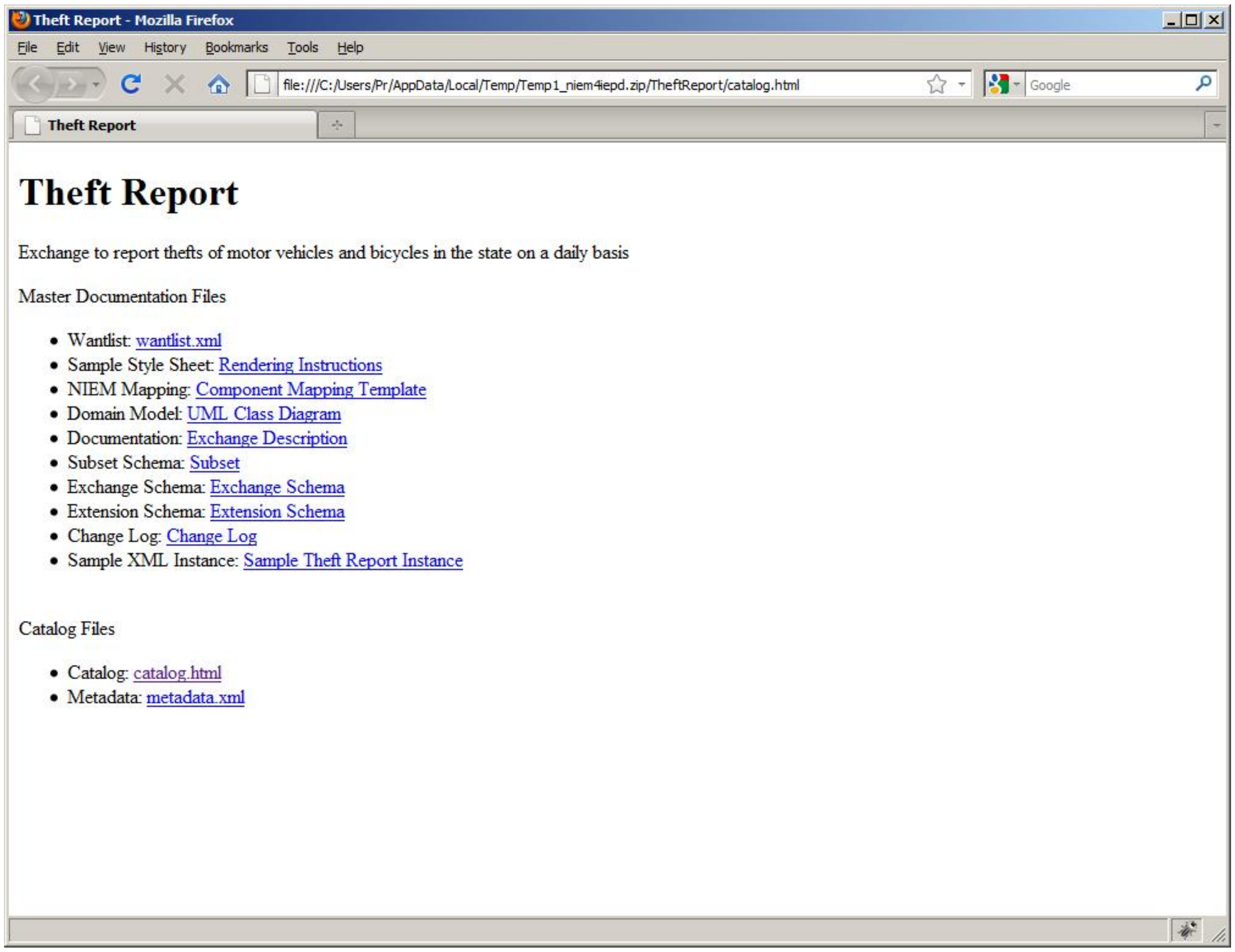
Looking at the generated IEPD, you can see a copy of all of the artifacts that you uploaded plus two new generated files. The `metadata.xml` document, in Example 15, provides a standardized cross-IEPD format for information about the exchange.

Example 15. The metadata document

```
<Metadata>
 <URI>http://www.datypic.com/theftreport</URI>
 <Name>Theft Report</Name>
 <Summary>Exchange to report thefts of motor vehicles and bicycles on a daily basis</Summary>
 <Description>Contains information about a theft of a motor vehicle or bicycle,
                including the theft date, location, description and identifiers of
                the stolen property, and victim and witness information.</Description>
 <Version>1.0</Version>
 <URL>http://www.datypic.com/theftreport</URL>
 <CreationDate>03/01/2010</CreationDate>
 <LastRevisionDate>03/01/2010</LastRevisionDate>
 <NextRevisionDate>06/01/2010</NextRevisionDate>
 <NIEMVersion>2.0</NIEMVersion>
 <Security>Public</Security>
 <Maturity>2</Maturity>
 <Status>Final</Status>
 <Schedule/>
 <Lineage/>
 <Relationships/>
 <Keywords/>
 <Domain>Justice, </Domain>
 <ExchangePartners/>
 <Process/>
 <TriggeringEvent/>
 <Conditions/>
 <Endorsements/>
 <Sponsors>Datypic</Sponsors>
 <Purpose>Report thefts of motor vehicles and bicycles to interested parties</Purpose>
 <MessageExchangePatterns>publish/subscribe</MessageExchangePatterns>
 <CommunicationsEnvironment/>
 <ExchangePartnerCategories>Law Enforcement, DMV,
                               Department of Revenue</ExchangePartnerCategories>
 <AuthoritativeSource>
  <Category>none</Category>
  <Organization>
   <Name>Datypic</Name>
   <Address1/>
   <Address2/>
   <City/>
   <State/>
   <Zip/>
   <Country/>
   <URL>http://www.datypic.com</URL>
  </Organization>
  <PointOfContact>
   <Name>Priscilla Walmsley</Name>
   <Address1/>
   <Address2/>
   <City/>
   <State/>
   <Zip/>
   <Country/>
   <Phone>231-555-1212</Phone>
   <Fax/>
   <Email>pwalmsley@datypic.com</Email>
  </PointOfContact>
 </AuthoritativeSource>
</Metadata>
```

The `catalog.html` file is an XHTML document that serves as a table of contents of the artifacts in the IEPD. It is shown in its rendered human-readable form in Figure 22. The catalog is also machine-readable, thanks to `rddl:purpose` attributes embedded in the XHTML.

Figure 22. Rendered IEPD catalog file


Theft Report - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Users/Pr/AppData/Local/Temp/Temp1_niem4iepd.zip/TheftReport/catalog.html
Google
Theft Report

Theft Report

Exchange to report thefts of motor vehicles and bicycles in the state on a daily basis

Master Documentation Files

- Wantlist: wantlist.xml
- Sample Style Sheet: Rendering Instructions
- NIEM Mapping: Component Mapping Template
- Domain Model: UML Class Diagram
- Documentation: Exchange Description
- Subset Schema: Subset
- Exchange Schema: Exchange Schema
- Extension Schema: Extension Schema
- Change Log: Change Log
- Sample XML Instance: Sample Theft Report Instance

Catalog Files

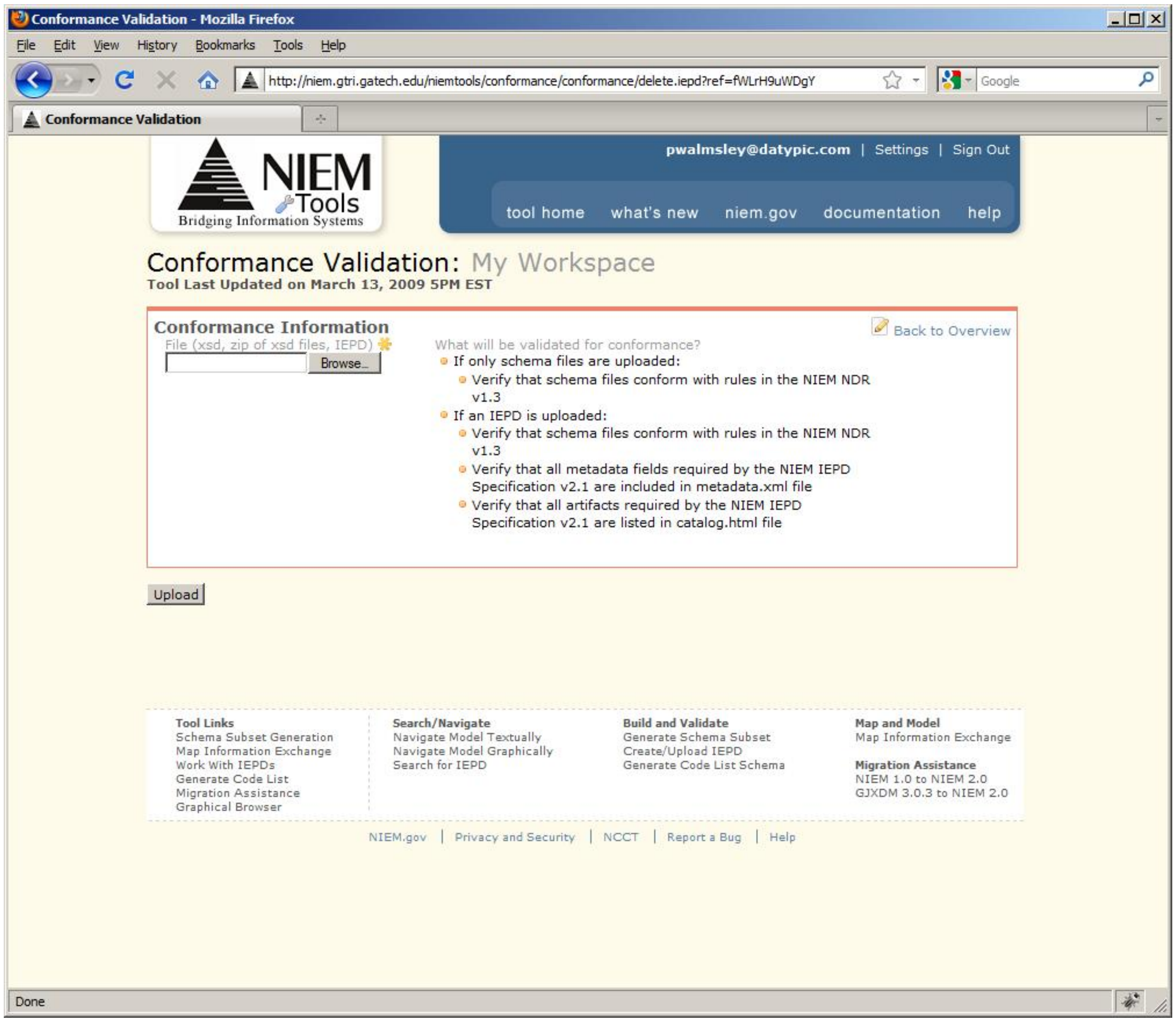
- Catalog: catalog.html
- Metadata: metadata.xml

Validating the IEPD

Once you create an IEPD, you can test it for conformance using the NIEM Conformance Validation Tool. This process ensures that all required artifacts are present in the IEPD. It also tests the schemas against some of the rules for NIEM-conformant schemas defined in the NIEM Naming and Design Rules (NDR) document.

From the Conformance Validation Tool main page, click **Begin** to bring up the page shown in Figure 23. It prompts you to upload a file, which will be your entire IEPD as a zip file. The tool uses the catalog file to determine the location and type of all artifacts. After you click through the page that asks you to verify the purpose of your artifacts, a new section of this page, called **My Validations**, appears with a conformance report for your IEPD.

Figure 23. Conformance tool


Conformance Validation - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://niem.gtri.gatech.edu/niemtools/conformance/conformance/delete.iepd?ref=fWLrH9uWDgY
Conformance Validation
NIEM Tools
Bridging Information Systems
pwalmsley@datypic.com | Settings | Sign Out
tool home what's new niem.gov documentation help

Conformance Validation: My Workspace
Tool Last Updated on March 13, 2009 5PM EST

Conformance Information
File (xsd, zip of xsd files, IEPD)

Back to Overview

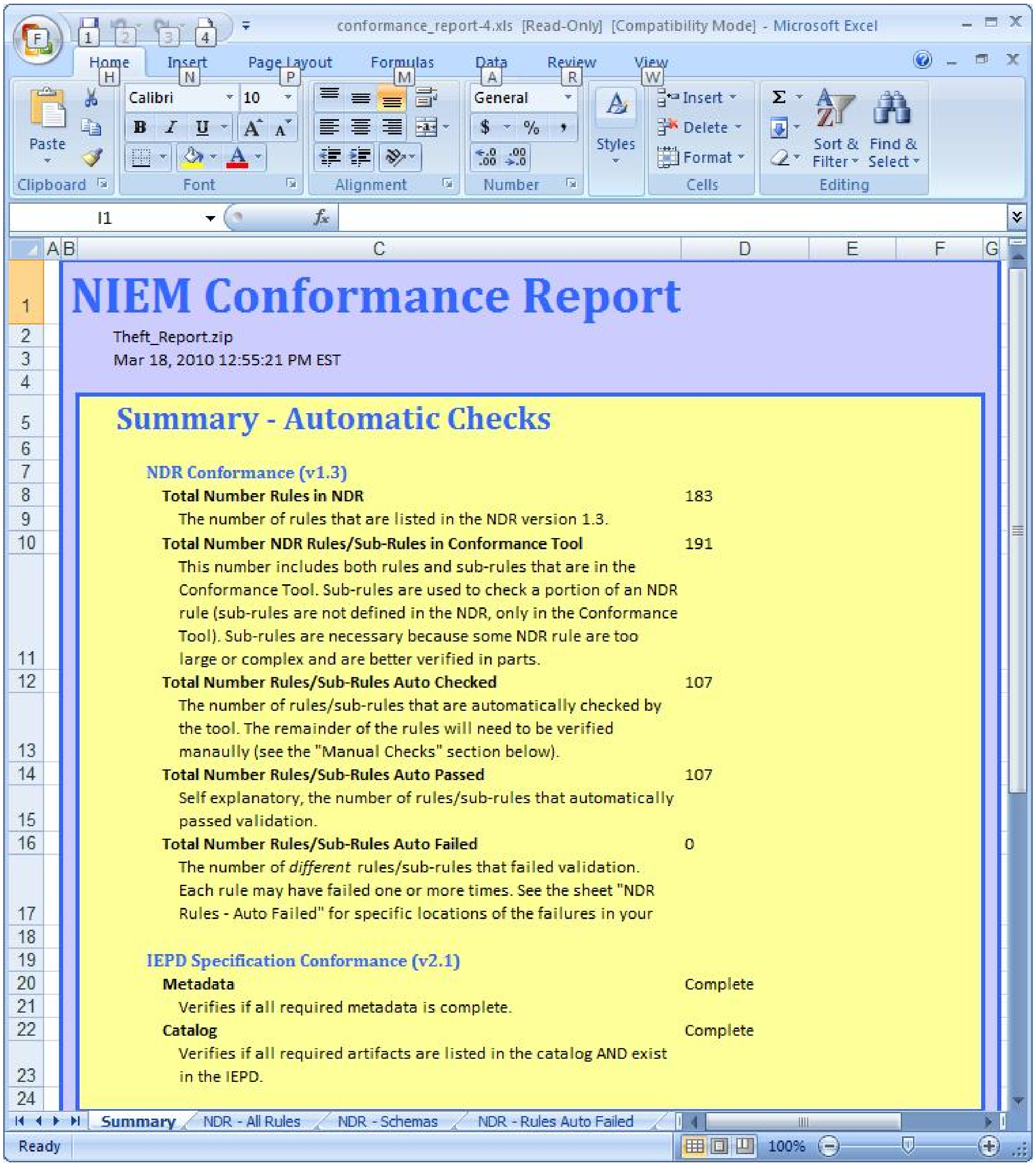
What will be validated for conformance?

- If only schema files are uploaded:
  - Verify that schema files conform with rules in the NIEM NDR v1.3
- If an IEPD is uploaded:
  - Verify that schema files conform with rules in the NIEM NDR v1.3
  - Verify that all metadata fields required by the NIEM IEPD Specification v2.1 are included in metadata.xml file
  - Verify that all artifacts required by the NIEM IEPD Specification v2.1 are listed in catalog.html file

Figure 24 shows the summary page of the conformance report, which is in Microsoft® Office Excel® format. It has several worksheets:

- **Summary** is the basic statistical information shown in Figure 24.
- **NDR - All Rules** provides a list of all the rules in the NDR and, for those that can be automatically checked, whether the schemas in the IEPD passed. For each rule that cannot be checked automatically, a drop-down list allows you to indicate that you checked it manually.
- **NDR - Schemas** is a summary of all of the schema documents and whether they are NIEM conformant.
- **NDR - Rules Auto Failed** is a list of each instance of an NDR rule violation.
- **IEPD - Metadata** is a list of all of the metadata fields, signaling an error if they are required but absent.
- **IEPD - Catalog** is a list of all of the artifact types, signaling an error if they are required but absent.

Figure 24. Conformance report summary



NIEM Conformance Report

Theft_Report.zip

Mar 18, 2010 12:55:21 PM EST

Summary - Automatic Checks

NDR Conformance (v1.3)

Total Number Rules in NDR	183
The number of rules that are listed in the NDR version 1.3.	
Total Number NDR Rules/Sub-Rules in Conformance Tool	191
This number includes both rules and sub-rules that are in the Conformance Tool. Sub-rules are used to check a portion of an NDR rule (sub-rules are not defined in the NDR, only in the Conformance Tool). Sub-rules are necessary because some NDR rule are too large or complex and are better verified in parts.	
Total Number Rules/Sub-Rules Auto Checked	107
The number of rules/sub-rules that are automatically checked by the tool. The remainder of the rules will need to be verified manaully (see the "Manual Checks" section below).	
Total Number Rules/Sub-Rules Auto Passed	107
Self explanatory, the number of rules/sub-rules that automatically passed validation.	
Total Number Rules/Sub-Rules Auto Failed	0
The number of *different* rules/sub-rules that failed validation. Each rule may have failed one or more times. See the sheet "NDR Rules - Auto Failed" for specific locations of the failures in your	

IEPD Specification Conformance (v2.1)

Metadata	Complete
Verifies if all required metadata is complete.	
Catalog	Complete
Verifies if all required artifacts are listed in the catalog AND exist in the IEPD.	

A complete copy of the conformance report for the Theft Report IEPD is available from the Artifacts page.

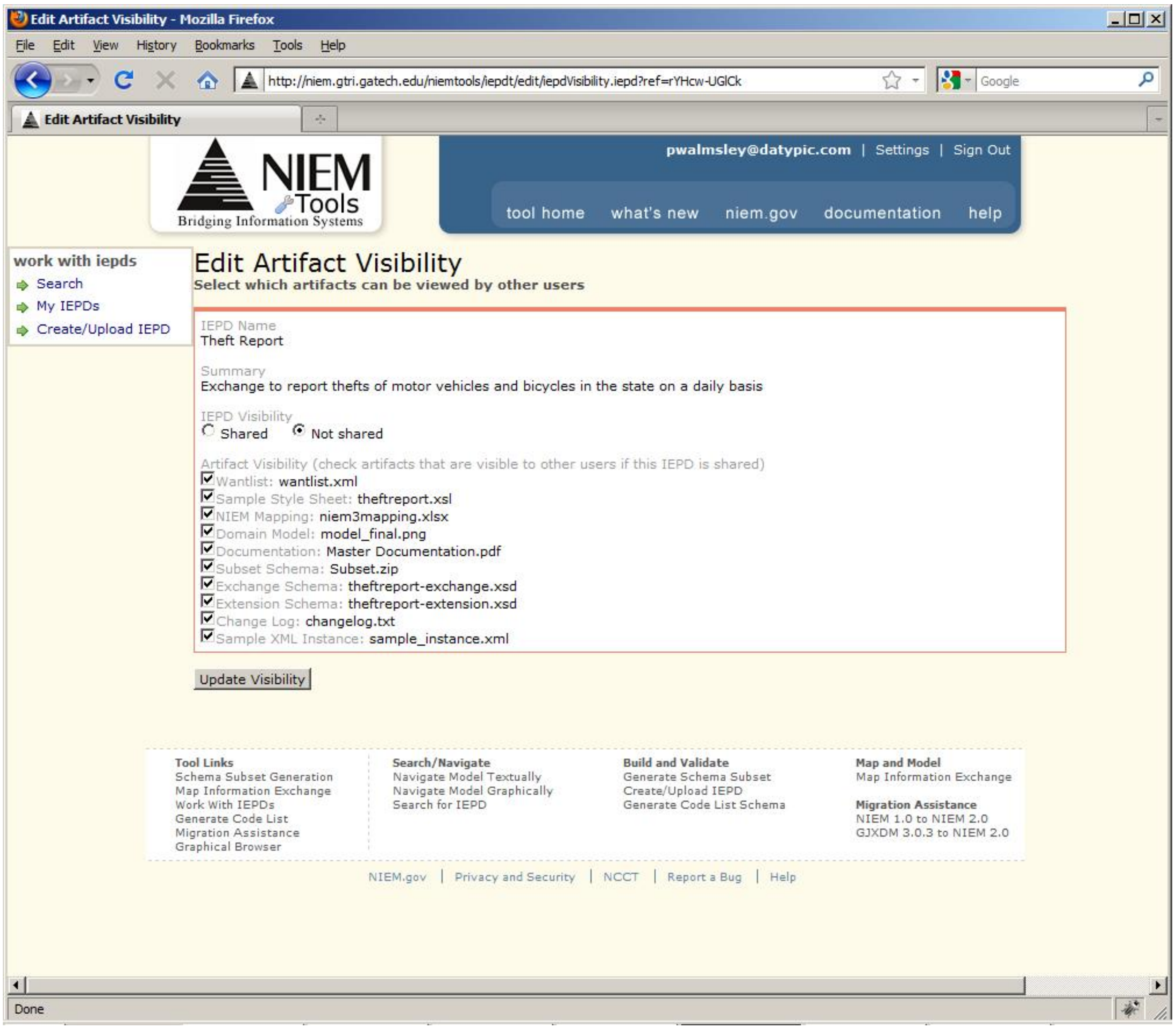
Submitting an IEPD to public repositories

IEPDs are meant to be shared. Although NIEM helps interoperability by providing common definitions and techniques, XML documents from two IEPDs that use different subsets of NIEM are not interchangeable. Therefore, reuse of IEPDs is critical to achieving interoperability.

For others to use your IEPD, it has to be available publicly for others to discover. Two major Web sites provide directories of existing IEPDs, and submitting your IEPD to these sites is easy.

First, you can list your IEPD on the NIEM Web site itself to make it available under the Search function of the Work with IEPDs tool. To do this within the Work with IEPDs tool, bring up your IEPD summary page. Click the **Edit** link in the top right, then click **Edit Visibility/Sharing** to bring up the Edit Artifact Visibility page in Figure 25.

Figure 25. Editing IEPD visibility



Edit Artifact Visibility

Select which artifacts can be viewed by other users

IEPD Name
Theft Report

Summary
Exchange to report thefts of motor vehicles and bicycles in the state on a daily basis

IEPD Visibility
Shared [✓ Not shared]

Artifact Visibility (check artifacts that are visible to other users if this IEPD is shared)

- ✓ Wantlist: wantlist.xml
- ✓ Sample Style Sheet: theftreport.xsl
- ✓ NIEM Mapping: niem3mapping.xlsx
- ✓ Domain Model: model_final.png
- ✓ Documentation: Master Documentation.pdf
- ✓ Subset Schema: Subset.zip
- ✓ Exchange Schema: theftreport-exchange.xsd
- ✓ Extension Schema: theftreport-extension.xsd
- ✓ Change Log: changelog.txt
- ✓ Sample XML Instance: sample_instance.xml

From here, select **Shared** to make the IEPD visible. You can also hide individual artifacts (by clearing the check box beside them) even if the IEPD itself is visible. This allows users to be aware of the IEPD even if individual artifacts are classified. Click **Update Visibility** when you are done.

The second place to submit your IEPD is to the Office of Justice Programs (OJP) IEPD Clearinghouse. To do this, visit the clearinghouse Web site, and click **Submit IEPD Information** to get a form on which you can upload your IEPD.

LEXS: An alternatives to creating an entire IEPD

Now that you understand the process of developing a NIEM IEPD, consider again whether this is a task you should undertake. At the beginning of this article, I suggested that you consider reusing an existing IEPD before you create your own from scratch. Not only will you improve your interoperability with other applications, but you will also save yourself a lot of work.

However, it is sometimes difficult to find an IEPD that meets all of your requirements. In these cases, a good solution is Logical Entity Exchange Specification (LEXS). LEXS is a NIEM IEPD framework that balances the competing goals of interoperability and flexibility by separating documents into a digest and a structured payload. The digest, which contains the most commonly used NIEM components, has a fixed structure and is interoperable across all LEXS-based IEPDs. The structured payload allows individual LEXS-based IEPDs to extend and customize the LEXS base model. LEXS also provides solutions for handling message exchange, search, subscriptions, attachments, and rendering.

Conclusion

Throughout this article, you have seen how to create a NIEM IEPD, including all steps in the process: modeling an exchange, creating an appropriate NIEM subset for the model, writing your own extensions of NIEM, and assembling all the artifacts into an IEPD. Following these guidelines for a NIEM-conformant exchange will help you to capitalize on the promise of NIEM: to facilitate information sharing among public and private sector organizations.

About the author

Priscilla Walmsley serves as Managing Director and Senior Developer at Datypic. She specializes in XML technologies, architecture and implementation. She has implemented NIEM exchanges for the U.S. Departments of Justice, Defense, and Health and Human Services.

She is the author of *Definitive XML Schema* (Prentice Hall, 2012), and *XQuery* (O'Reilly Media, 2015). In addition, she co-authored *Web Service Contract Design and Versioning for SOA* (Prentice Hall, 2008).

About Datypic

Datypic provides development services and training, specializing in XML, content management and electronic publishing. We are experts in XML-related technologies such as XML Schema, XSLT and XQuery, and have extensive experience with software development and implementation.

We participate in projects ranging from one day to many months, anywhere in the world. We can arrange to work remotely or at your site, whichever you prefer.

For more information, please read about our services at datypic.com.

About this article

A prior version of this article was first published by IBM developerWorks. Its current version number is 2.0 and it was last updated on April 30, 2014.