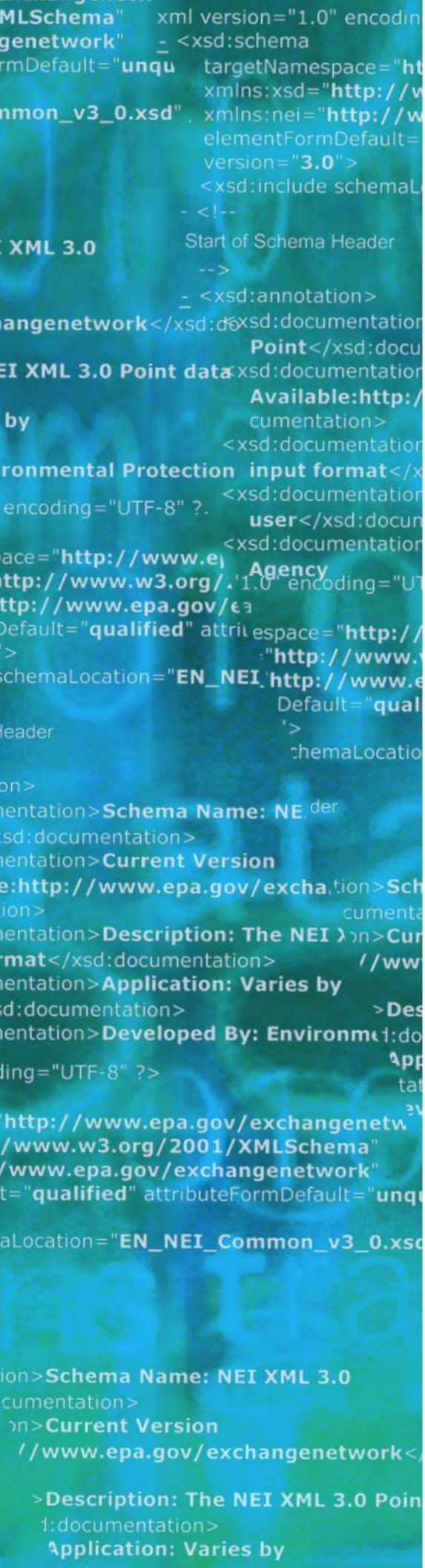


XML Design Rules and Conventions (DRC) for the Exchange Network

Version: 2.0

Revision Date: 01/12/2010

Environmental Information exchange Network

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1. Introduction

1.1. About XML Design Rules and Conventions (DRCs)

This guide establishes design rules and guidelines for the creation and use of the Extensible Markup Language (XML) for joint use by the U.S. Environmental Protection Agency (EPA), state and local governments, tribes, and territories exchanging data on the National Environmental Information Exchange Network (Network).

1.1.1. Audience

The audience for this guide includes Exchange Network policymakers, schema developers, XML instance authors, and XML application integrators. This guide applies to all Exchange Network organizations and their employees. It also applies to the facilities and personnel of agents (including contractors and grantees) who are involved in XML-related information resource activities.

1.1.2. Background

Published in 2003, the XML Design Rules and Conventions v1.0 document was one of the first formal published rules for development and management of data exchanges on the Network. Since that time, the Network has evolved greatly. The Network devised a comprehensive component versioning strategy, the XML namespace conventions were changed, and shared tools such as the Shared Schema Components and Exchange Network Header were developed, to name a few.

With the evolution of the Network, many discrete guidance documents were published. Over time, the volume of documentation swelled, making it difficult for schema and exchange developers to gather and comprehend all of the information needed to properly design an exchange. Complicating matters, information sometimes conflicted between documents.

The release of this document, XML Design Rules and Conventions v2.0 (DRC v2.0), along with the companion document Exchange Design Rules and Conventions v1.0 (EDRC v1.0), consolidates many of the past guidance papers into two comprehensive documents.

The following documents are superseded by the rules and guidelines in DRC v2.0 and EDRC v1.0:

- XML Design Rules and Conventions for the Environmental Information Exchange Network (September, 2003)
- XML Schema Design Rules and Conventions v1.1 – Interim Update for the Exchange Network (April, 2006)

- Principles, Rules, and Procedures for Change Management on the Exchange Network (December, 2006) including Appendix A and B
- Flow Documentation Checklist v2.0 (December, 2007)
- Namespace Organization, Naming and Schema File Location v1.11 (March, 2006)
- Shared Schema Components Technical Reference and Usage Guides v2.0 (May, 2006)
- Exchange Design Guidance and Best Practices for the Exchange Network v1.2 (August, 2006)

Additional discussion and background information on topics covered in the DRC v2.0 and EDRC v2.0 can be found in these documents if needed, however readers should be aware that some concepts presented in the historical documents may be out of date and in conflict with the latest rules and guidance.

1.2. Semantic Conventions used in this Document

Each rule or guideline has been distilled into a single, normative statement describing what a schema or exchange developer MUST, MUST NOT, SHOULD, SHOULD NOT, or MAY do. The specific meaning of these terms was originally defined in the Request for Comments 2119 issued by the Internet Engineering Task Force[1]. The terms are defined as follows:

Note that the force of these words is modified by the requirement level of the document in which they are used.

- **MUST**. This word, or the terms "REQUIRED" or "SHALL," means that the definition is an absolute requirement of the specification.
- **MUST NOT**. This phrase, or the phrase "SHALL NOT," means that the definition is an absolute prohibition of the specification.
- **SHOULD**. This word, or the adjective "RECOMMENDED," means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- **SHOULD NOT**. This phrase, or the phrase "NOT RECOMMENDED," means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be

[1] Internet Engineering Task Force, Request for Comments 2119, March 1997, www.ietf.org/rfc/rfc2119.txt?number=2119.

understood and the case carefully weighed before implementing any behavior described with this label.

- **MAY**. This word, or the adjective "OPTIONAL," means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor believes that it enhances the product, while another vendor may omit the same item. An implementation that does not include a particular option MUST be prepared to interoperate with another implementation that does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

1.3. Component Naming Syntax Conventions

Naming conventions for components such as schema files, namespaces, and documents are used throughout the design rule documentation. The following contentions are used:

Convention	Description
{ }	Placeholder for user-supplied word or term. Text inside the placeholder is the name for the term.
" "	Text included in quotes must be used verbatim in the component name.
[ ]	Items in brackets are optional.

For example, the naming convention for a component might be described as:

{ExchangeIdentifier}"_v"{MajorVersionNumber}[{RevisionIdentifier}]

Below are two examples of component names that properly implement the convention above:

- MyExchange_v2
- ABC_v3a

1.4. Exceptions to Design Rules

While the rules and guidelines in this document describe specific requirements for designing schema and data exchanges, Network governance recognizes that there will be scenarios where certain rules are not practical or cannot be met. In these cases, schema and exchange developers should document the discrepancies and the justification for deviating from the rules. Exchanges that deviate from rules will most often still be deemed acceptable for use on the Network and will be evaluated on a case-by-case basis during the Exchange Documentation Package review process.

2. XML Design Rules and Conventions

2.1. General XML Design

This section of the guide contains general, high-level XML design rules and guidelines that apply to all XML development efforts, rather than to a specific facet of XML technology described in following sections.

The general rules and guidelines, listed below, provide the common foundation for data and document development within the Network.

General XML Design

Rules and Guidelines	
Rule	**Description**
[GD1-1]	All Exchange Network schema MUST be based on the W3C suite of technical specifications that hold Recommendation status.
[GD1-2]	Only W3C technical specifications holding Recommendation, Proposed Recommendation, or Candidate Recommendation status shall be used for production activities.
[GD1-3]	W3C technical specifications holding Draft status MAY be used for prototyping. Such prototypes will not be put into production until the associated specifications reach a Recommendation, Proposed Recommendation, or Candidate Recommendation status.
[GD1-4]	All XML parsers, generators, validators, enabled applications, servers, databases, operating systems, and other software acquired or used by partners' activities shall be fully compliant with all W3C XML specifications that hold a Recommendation status.
[GD1-5]	The normative schema documents that implement the partner document types shall conform to XML Schema Part 1: Structures and XML Schema Part 2: Datatypes.
[GD1-6]	Each message MUST represent a single logical unit of information (such as facility permit compliance data) conveyed in the root element.
[GD1-7]	The business function of a message set MUST be unique and must not duplicate the business function of another message.
[GD1-8]	The name of the message set MUST be consistent with its definition.
[GD1-10]	Messages MUST use the UTF-8/UNICODE character set.

[GD1-11]	XML instance documents conforming to schemas SHOULD be readable and understandable, and should enable reasonably intuitive interactions.
[GD1-12]	Messages shall be modeled for the abstractions of the user, not the programmer.
[GD1-13]	Messages shall use markup to make data substructures explicit (that is, distinguish separate data items as separate elements and attributes).
[GD1-14]	Messages shall use well-known datatypes.
[GD1-15]	EPA messages shall reuse registered datatypes to the maximum extent practicable.
[GD1-16]	In a schema, information that expresses associations between data elements in different classification schemes (in other words, "mappings") MAY be regarded as metadata. This information should be accessible in the same manner as the rest of the information in the schema.

2.2. XML Tag Naming Conventions

The Exchange Network XML Tag Naming conventions are based on the ISO 11179 metadata standard. In addition to the rules and guidelines outlined in this section, the following guidelines also apply:

- All type, element, and attribute names should use American English. Type, element, and attribute names may use Oxford English. The use of Oxford English is encouraged for any message set that has the potential for international exchange.
- The content (or value) within tags, attributes, and other items may be in any language.

2.2.1. Tag Structure

The following defines rules for all *new* development of XML tag names. These rules are the "how" as opposed to the "what" for tag name formation.

Tag Structure

Rules and Guidelines	
Rule	**Description**
[GD3-1]	Element names MUST be in "Upper Camel Case" (UCC) convention, where UCC style capitalizes the first character of each word and compounds the name. Example: <UpperCamelCaseElement/>
[GD3-2]	Schema type names MUST be in UCC convention. Example: <DataType/>

[GD3-3]	Attribute names MUST be in "Lower Camel Case" (LCC) convention where LCC style capitalizes the first character of each word except the first word. Example: <UpperCamelCaseElement lowerCamelCaseAttribute="Whatever"/>
[GD3-4]	Acronyms SHOULD NOT be used, but in cases where they are used, the capitalization SHALL remain Example: <XMLSignature/>, and the acronym SHOULD be defined in the comments of the DTD or Schema or in a separate document noted in the DTD or Schema as providing a tag dictionary so that the meaning of the acronym is clear.
[GD3-5]	Abbreviations SHOULD NOT be used. In cases where they are used, they MUST be a major part of the federal or data standards vocabulary, and the abbreviation SHOULD be defined within the comments of the DTD or Schema or in a separate document (noted in the DTD or Schema) as providing a tag dictionary so that the meaning of the abbreviation is clear. An exception to this rule is when identifier is used as a representation term, ID SHOULD be used as part of the tag name.
[GD3-6]	Underscores (_), periods (.) and dashes (-) MUST NOT be used.
[GD3-7]	Verbosity in tag length SHOULD be limited to what is required to conform to the Tag Name Content recommendations. When tags will be used in database structures, a limit of 30 characters is recommended.

2.2.2. Tag Name Content (Semantic Guidelines)

The following tag naming conventions should be used in all Exchange Network XML schemas[2]. The guidance is the "what" as opposed to the "how" of tag name formation.

Tag Name Content

Rules and Guidelines	
Rule	**Description**
[GD3-8]	Element, attribute, and datatype tag names MUST be unique.
[GD3-10]	High-level parent element tag names SHOULD consist of a meaningful aggregate name followed by the term "Details". The aggregate name may consist of more than one word. Example: <SiteFacilityDetails/>
[GD3-11]	Tag names SHOULD be concise and MUST NOT contain consecutive redundant words.

[2] The list of rules is a modified version of the dictionary naming conventions from the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT), *Core Components Technical Specification, Part 1 (*Version 2.0.), August 11, 2003. This document was created as follow-on from the ebXML initiative and based on ISO 11179 Part 5, "Naming and Identification Principles for Data Elements."

[GD3-12]	Lowest level (it has no children) element tag name SHOULD consist of the Object Class, the name of a Property Term, and the name of a Representation Term. An Object Class identifies the primary concept of the element. It refers to an activity or object within a business context and may consist of more than one word. Example: <LocationSupplementalText/>
[GD3-13]	A Property Term identifies the characteristics of the object class. The name of a Property Term SHALL occur naturally in the tag definition and may consist of more than one word. A name of a Property Term shall be unique within the context of an Object Class but may be reused across different Object Classes. Example: <LocationZipCode/> and <MailingAddressZipCode/> may both exist.
[GD3-14]	If the name of the Property Term uses the same word as the Representation Term (or an equivalent word), this Property Term SHALL be removed from the tag name. In this case, only the Representation Term word will remain. *Examples:* If the Object Class is "*Goods*", the Property Term is "*Delivery Date*", and Representation Term is "*Date*", the tag name is <*GoodsDeliveryDate/*>
[GD3-15]	A Representation Term categorizes the format of the data element into broad types. A list of UN/CEFACT Representation Terms is included at the end of this list of rules, but the EPA and its partners may need to augment this list to accommodate the specific needs for environmental data. When possible the pre-defined UN/CEFACT list SHOULD be used. Proposed additions should be submitted to the TRG for consideration.
[GD3-16]	The name of the Representation Term MUST NOT be truncated in the tag name.
[GD3-17]	A tag name and all its components MUST be in singular form unless the concept itself is plural.
[GD3-18]	Non-letter characters MUST only be used if required by language rules.
[GD3-19]	Tag names MUST only contain verbs, nouns and adjectives (no words like "and", "of", "the").
[GD3-A]	All datatype names MUST end with either "Type" or "DataType".

The following table provides example tag names and the breakdown of tag names by object class, property term, and representation term.

Dictionary entry name	Definition	Object class	Property term	Representation term	Tag
Country. Details	Information about a country	Country	CountryDetails		
Country. Identification.Code	A nation with its own government	Country	Identification	Code	CountryIdentificationCode
Country Name	The name that represents a primary geopolitical unit	Country	Name	Name	CountryName

	of the world				
Location. Identification.Code	The identifier of a location	Location	Identifi-cation	Code	LocationIdentificationCode
Facility Registry Identifier	The identification number assigned by the EPA Facility Registry System to uniquely identify a facility site	Facility Registry	Identifier	Identifier	FacilityRegistryIdentifier
Organization. Details	An organized body, such as a business, government body, department, or charity	Organi-zation	OrganizationDetails		
Organization Data Universal Numbering System (DUNS) number	The DUNS number assigned by Dun and Bradstreet to identify unique business establishments	Organi-zation	DUNS	Identifier	OrganizationDUNSIdentifier
Organization. Name	The text used to identify an organization, the organization's name	Organi-zation	Name	Name	OrganizationName
Organization Formal Name	The legal, formal name of an organization that is affiliated with the facility site	Organi-zation	Formal	Name	OrganizationFormalName

The following table lists the acceptable representation terms to be used in tag names[3].

Term	**Definition**
Amount	A number of monetary units specified in a currency where the unit of currency is explicit or implied.
Binary Object	A set of finite-length sequences of binary octets. Secondary Representation Terms: Graphic, Picture, Sound, Video.
Code	A character string (letters, figures, or symbols) that, for brevity and/or language independence, may be used to represent or replace a definitive value or text of a Property.
Date Time	A particular point in the progression of time (ISO 8601). Secondary Representation Terms: Date, Time.
Identifier	A character string used to establish the identity of, and distinguish uniquely, one instance of an object within an identification scheme from all other objects within the same scheme.
Indicator	A list of two mutually exclusive Boolean values that express the only possible states of a Property. (Values typically indicate a condition such as on/off or true/false.)
Measure	A numeric value determined by measuring an object. Measures are specified with a unit of measure. The applicable unit of measure is taken from UN/ECE

[3] UN/CEFACT, *Core Components Technical Specification, Part 1* (Version 2.0), August 11, 2003.

	Rec. 20.
Numeric	Numeric information that is assigned or is determined by calculation, counting, or sequencing. It does not require a unit of quantity of a unit of measure. Secondary Representation Terms: Value, Rate, Percent.
Quantity	A counted number of nonmonetary units. Quantities need to be specified with a unit of quantity.
Text	A character string, (i.e., a finite set of characters) generally in the form of words of a language. Secondary Representation Terms: Name.

2.3. Datatypes

Datatypes represent the kind of information elements and attributes can hold—character strings or dates, for example. This section discusses datatypes and their use in schemas, and provides guidance for the Exchange Network's use of XML Schema.

2.3.1. Simple Datatypes

Simple datatypes include both built-in datatypes and user-defined datatypes.

Built-in Datatypes

Built-in datatypes are the datatypes that were defined by the W3C Schema team and included in the W3C Schema standard. The small number of built-in datatypes is believed to be so universal that they would need to be constantly redefined by most schema developers.

Built-in datatypes cannot be user defined. The following are examples of the W3C Schema standard simple datatypes:

- String
- anyURI—a standard Internet URI
- Boolean—a two-state true-or-false flag
- Decimal
- Date
- Integer
- negativeInteger—any integer with a value less than zero.

The following is an example of an element declaration that specifies a simple datatype:

```
<xsd:element name="SubmitterIdentificationCode" type="xsd:integer"/>
```

Any XML processor that complies with the W3C Schema standard will automatically validate built-in datatypes. That is to say, if an XML instance document contained a

string value instead of an integer value for the above element, an XML processor would generate an error.

User-defined Datatypes

One of the advantages of XML Schema is the ability to define your own datatypes. User-defined datatypes are based on the existing built-in datatypes and can also be further derived from existing user-defined datatypes. Datatypes can be derived in one of three ways:

- *By restriction*. Restraints are placed on the built-in datatype's limiting facets. The integer datatype could be restricted to allow only a range of integers.
- *By list*. The derived datatype is a list of values from the built-in datatype. The string datatype could be restricted to allow only county names from a single state.
- *By union*. The derived datatype is a combination of two or more built-in datatypes.

User-defined Datatypes

Pros and Cons	
Advantages	Simple datatypes allow for specification of data requirements beyond what is possible with DTDs. Using simple datatypes increases interoperability between XML applications. Simple datatypes are validated by XML processors.
Disadvantages	A simple datatype may not always have the proper lexical format for use in a system. For instance, the *date* simple datatype is formatted *YYYY-MM-DD*, which may not be suitable for certain situations.
Rules and Guidelines	
Rule	**Description**
[SD2-1]	Exchange Network schemas MUST use simple datatypes to the maximum extent possible.
[SD2-A]	Schema developers SHOULD ensure that use of built-in data types in schema are appropriate to the data being represented.
Justification	

Simple datatypes are valuable because they allow stronger data validation capabilities than DTDs. Use of simple datatypes increases data quality among XML applications because all applications that use simple datatypes are subject to the same validations by XML processors.

When lexical format of a simple datatype is not suitable, schema developers can create their own datatypes using the W3C Schema Regular Expression syntax.

2.3.2. Complex Datatypes

Complex datatypes are user-defined datatypes that contain child elements or attributes. Complex datatypes can be defined as either global complex datatypes or local complex datatypes. Each is discussed below.

Global Complex Datatypes

Global complex datatypes are direct descendants of the root element of a schema. They can be associated with any element in a schema. Global complex datatypes are also known as named complex datatypes because they have an associated name. The following is an example of a global complex datatype:

```
<xsd:complexType name="FacilitySiteDetailsType">
  <xsd:sequence>
    <xsd:element name="FacilityIdentificationCode" type="xsd:string"/>
    <!—information removed for example purposes—>
  </xsd:sequence>
</xsd:complexType>
```

The following is an example of an element associated with the global complex datatype shown above:

```
<xsd:element name="FacilitySiteDetails" type="FacilitySiteDetailsType">
```

This declaration means that, in an XML instance document, the FacilitySiteDetails element will contain

- a subelement of FacilityIdentificationCode and
- any other elements declared within the FacilitySiteDetailsType global complex datatype.

The main advantage of global complex datatypes is that a change to a global complex datatype definition will propagate across all elements associated with that datatype in a schema. For example, if an element were added to the FacilitySiteDetailsType complex datatype definition, it would

- propagate to the declaration of the FacilitySiteDetails element and
- any other elements associated with FacilitySiteDetailsType datatype.

There may be situations when this is not desired. Continuing with the above example, there may be one place in a schema where the added element cannot appear. This may require two global complex datatypes—one that includes the new element and another that excludes it. The appropriate "version" of the datatype would then be used, as required.

Global Complex Datatypes

Pros and Cons	
Advantages	Global complex datatypes can be associated with any element in a schema. A change to a global complex datatype definition will propagate across all elements that are associated with that datatype in a schema. This allows far-reaching changes to be made
Disadvantages	A change to a global complex datatype definition may propagate across elements whose datatype should not be changed. Additional schema updates may be required in such cases, thereby increasing maintenance costs. Use of global complex datatypes places additional overhead on an XML processor to resolve all references.
Rules and Guidelines	
Rule	**Description**
[SD2-3]	Exchange Network schemas that employ complex datatypes MUST define the complex datatypes as global.
Justification	
Global complex datatypes are valuable because they can be associated with any element in a schema. This promotes high datatype visibility. Although there is a potential requirement for additional schema updates in a propagation scenario, the potential advantages for using global complex datatypes far outweigh the potential disadvantages. The potential overhead on an XML processor to resolve all references to global complex datatypes is not a high enough concern to warrant not recommending their use.	

Local Complex Datatypes

Local complex datatypes can appear anywhere in a schema. They are associated with a single element, and their definition cannot be associated with any other element in a schema. Local complex datatypes are also known as anonymous complex datatypes because they do not have a name associated with them. The following example is similar to the example given for global complex datatypes, only the FacilitySiteDetailsType datatype is now represented as a local complex datatype:

```
<xsd:element name="FacilitySiteDetails">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="FacilityIdentificationCode" type="xsd:string"/>
      <!—information removed for example purposes—>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Although the above declaration uses a local complex datatype, the result in an XML instance document will be the same as if the datatype were global; the FacilitySiteDetails element will contain a subelement of FacilityIdentificationCode and any other elements declared within the local complex datatype.

Because the complex datatype definition in the above declaration is associated with only the FacilitySiteDetails element, a change to its definition will affect only that element.

Local Complex Datatypes

Pros and Cons	
Advantages	A change to a local complex datatype definition will affect only the element with which it is associated, thereby allowing changes to be confined to a single location in a schema. This may be desirable in some situations.
Disadvantages	Local complex datatypes can be associated only with a single element in a schema. If the same local complex datatype definition is used in the declaration of multiple elements in a schema, and a change to the datatype is required, a change will need to be made where the local complex datatype definition exists. This can increase maintenance costs.
Rules and Guidelines	
Rule	**Description**
[SD2-5]	Exchange Network schemas SHOULD NOT use local complex datatypes.
Justification	
Use of local complex datatypes is discouraged because they result in low datatype visibility.	

2.4. Elements and Attributes

Elements are the basic building blocks of an XML instance document and are represented by tags. Attributes are W3C Schema constructs associated with elements that provide further information regarding elements. While elements can be thought of as containing data, attributes can be thought of as containing metadata. This

chapter discusses the element and attribute constructs and their potential uses, and provides guidance for the Exchange Network.

2.4.1. Elements

Elements are the basic building blocks of an XML document instance. An element may contain one or more sub-elements, as shown in the following XML instance document excerpt:

```
<FacilitySubmission>
  <FacilitySiteDetails>
    <FacilityIdentificationCode>15849</FacilityIdentificationCode>
    <FacilityAddressDetails>
      <!—information removed for example purposes—>
    </FacilityAddressDetails>
  </FacilitySiteDetails>
</FacilitySubmission>
```

In the above example, the FacilitySiteDetails element is a sub-element of the FacilitySubmission element, while the FacilityIdentificationCode and FacilityAddressDetails elements are sub-elements of the FacilitySiteDetails element. Elements can be extended as necessary (i.e., a schema developer can add sub-elements to an element if more information needs to be conveyed in an XML instance document than is currently conveyed).

Element order is enforced by XML processors. An error will result if the element order in an XML instance document is different than the declared order of the elements in the schema. Elements can be declared as either global elements or local elements. Each is discussed below.

Global Elements

Global elements are direct descendants of the root element of a schema. They can be referenced within any complex datatype definition in a schema through the use of a “ref” attribute. In the following example, the FacilityIdentificationCode element is a global element:

```
<?xml version=“1.0” encoding=“UTF-8”?>
<xsd:schema xmlns:xsd=“http://www.w3.org/2001/XMLSchema”>
  <xsd:element name=“FacilityIdentificationCode” type=“xsd:string”/>
  <!—information removed for example purposes—>
  <xsd:complexType name=“FacilitySiteDetailsType”>
    <xsd:sequence>
      <xsd:element ref=“FacilityIdentificationCode”>
      <!—information removed for example purposes—>
    </xsd:sequence>
  </xsd:complexType>
```

```
</xsd:schema>
```

Because the FacilityIdentificationCode element is a global element, a change to the FacilityIdentificationCode element declaration (such as a change in datatype) will propagate to the definition of the FacilitySiteDetailsType datatype, and all other complex datatype definitions where the element is referenced; however, there may be situations where this is not the desired result.

Continuing with the above example, there may be a reference to the FacilityIdentificationCode element in a schema not applicable for the new datatype. This may require the presence of two global elements—one associated with the new datatype, and the other associated with the original datatype. The appropriate "version" of the element would then be referenced, as needed.

A global element can serve as the root element of any XML instance document that conforms to a schema. In the following example, there are two global elements—AreaSubmission and EventSubmission. Therefore, an XML instance document that conforms to this schema can have either the AreaSubmission element or the EventSubmission element as its root:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="AreaSubmission" type="AreaSubmissionType"/>
  <xsd:element name="EventSubmission" type="EventSubmissionType"/>
  <!—information removed for example purposes—>
</xsd:schema>
```

In the above scenario, an XML instance document can have only one of the global elements in the schema as its root element. Therefore, it can include only that global element and its sub-elements.

Continuing with the same example, if an XML instance document has the AreaSubmission element as its root element, it can include the AreaSubmission element and its sub-elements (specifically, the elements contained within the AreaSubmissionType datatype). However, it cannot include the EventSubmission element or its sub-elements.

Global Elements

Pros and Cons

Advantages	Global elements can be referenced within any complex datatype definition in a schema. A change to a global element declaration will propagate to all complex datatype definitions where the element is referenced. This allows a far-reaching change to be made in a single location in a schema, thereby lowering maintenance costs. Global elements can serve as the root element of any XML instance document that conforms to a schema, thereby increasing schema versatility.
Disadvantages	A change to a global element declaration may propagate to elements for which the change should not apply. Additional schema updates may be required in such cases, thereby increasing maintenance costs. If a global element does not contain any mandatory sub-elements, it is possible to create an XML instance document with only a single empty element representing that global element; therefore, the XML instance document would contain no data. Use of global elements places additional overhead on an XML processor to resolve all references.
Rules and Guidelines	
Rule	**Description**
[SD3-1]	Exchange Network schemas MUST use global elements.
Justification	
Global elements are valuable because they can be referenced within any complex datatype definition in a schema. This promotes high element visibility. Although there is a potential requirement for additional schema updates in a propagation scenario, the potential advantages for using global elements far outweigh the potential disadvantages. Although an XML instance document can be created with only a single empty element representing a global element, the chances of this actually occurring in a real-world scenario are not high enough to warrant not recommending the use of global elements. The potential overhead on an XML processor to resolve all references to global elements is also not of high enough concern to warrant not recommending its use.	

Local Elements

Local elements are not direct descendants of the root element of a schema. Rather, they are nested inside the schema structure. Unlike global elements, local elements cannot be referenced outside of the complex datatype definition where they are declared. The following example is similar to the example shown above for global elements, but the FacilityIdentificationCode element is now a local element:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
  <!—information removed for example purposes—>
  <xsd:complexType name="FacilitySiteDetailsType">
    <xsd:sequence>
      <xsd:element name="FacilityIdentificationCode" type="xsd:string"/>
      <!—information removed for example purposes—>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Because the FacilityIdentificationCode element is now a local element, a change to the FacilityIdentificationCode element declaration will affect only the FacilitySiteDetailsType datatype.

It is possible to declare a local element in multiple places in a schema with a different datatype in each place. In the following example, the FacilityIdentificationCode element is declared as a local element within two different datatypes, but it has a different datatype in each declaration:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!—information removed for example purposes—>
  <xsd:complexType name="FacilitySiteDetailsType">
    <xsd:sequence>
      <xsd:element name="FacilityIdentificationCode" type="xsd:string"/>
      <!—information removed for example purposes—>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="StateReportingDetailsType">
    <xsd:sequence>
      <xsd:element name="FacilityIdentificationCode" type="xsd:integer"/>
      <!—information removed for example purposes—>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Local Elements

Pros and Cons	
Advantages	A change to a local element declaration will affect only that element, thereby allowing changes to be confined to a single location in a schema. This may be desirable in some situations.

Disadvantages	Local elements cannot be referenced within any complex datatype definition for a schema outside of the complex datatype definition where they are declared. If a local element is declared in multiple places in a schema with the same datatype, and a change to its datatype is required, a change will need to be made where the local element is declared. This can increase maintenance costs. Local elements cannot serve as the root element of any XML instance document that conforms to a schema.
Rules and Guidelines	
Rule	**Description**
[SD3-3]	Exchange Network schemas SHOULD NOT use local elements.
Justification	
Use of local elements is discouraged for Exchange Network schemas because they result in low element visibility.	

Cardinality of Elements

The term cardinality is defined as the number of elements in a set. When used in reference to W3C Schema, this term refers to the number of times an element may appear in a given content model in an XML instance document.

Cardinality is indicated in a schema using the minOccurs and maxOccurs constraints in an element declaration; these constraints are also known as occurrence indicators. Occurrence indicators can appear only on local element declarations or references to global elements. They cannot appear within global element declarations.

In the following example, the FacilitySiteDetails global element can occur a minimum of zero times (meaning it is optional) and a maximum of five times:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!—information removed for example purposes—>
  <xsd:complexType name="AreaSubmissionType">
    <xsd:sequence>
      <xsd:element ref="FacilitySiteDetails" minOccurs="0" maxOccurs="5"/>
      <!—information removed for example purposes—>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

It is possible to specify a different occurrence indicator value for a global element each time it is referenced in a schema. Therefore, the FacilitySiteDetails element in the

above example could be referenced in another global complex datatype in the schema with a minOccurs value of 2, thereby requiring that the element appear at least twice.

A maxOccurs value of "unbounded" can be used to indicate that an element can appear an unlimited number of times in a content model.

The default value for both occurrence indicators, minOccurs and maxOccurs, is 1. Therefore, in the following example, the FacilitySiteDetails global element may occur only once:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!—information removed for example purposes—>
  <xsd:complexType name="AreaSubmissionType">
    <xsd:sequence>
      <xsd:element ref="FacilitySiteDetails">
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Occurrence Indicators

Pros and Cons	
Advantages	Occurrence indicators allow an element to appear multiple times in a content model. It is possible to specify a different occurrence indicator value for a global element in each place in a schema where it is referenced.
Disadvantages	There are no disadvantages to this technique.
Rules and Guidelines	
Rule	**Description**
[SD3-5]	Exchange Network schemas SHOULD use occurrence indicators.
[SD3-6]	Exchange Network schemas SHOULD NOT use occurrence indicators when the required values are the default values.
Justification	
The ability to define specific cardinality for an element is very valuable. It is recommended that schema developers not specify default values for occurrence indicators (i.e., minOccurs="1"; maxOccurs="1") because doing so can unnecessarily clutter a schema.	

2.4.2. Attributes

Attributes are W3C Schema constructs associated with elements that provide further information regarding elements. While elements can be thought of as containing data,

attributes can be thought of as containing metadata. Unlike elements, attributes cannot be nested within each other—there are no "subattributes." Therefore, attributes cannot be extended as elements can. The following is an example of an attribute in an XML instance document:

```
<FacilitySiteDetails informationFormatIndicator="A">
```

Attribute order is not enforced by XML processors—that is, if the attribute order in an XML instance document is different than the order in which the attributes are declared in the schema to which the XML instance document conforms, no error will result. As with elements, attributes can be declared as either global attributes or local attributes.

General guidance on attributes is given below, followed by a discussion of global attributes and local attributes.

Attributes (General)

Pros and Cons	
Advantages	Attributes are useful for conveying metadata for elements.
Disadvantages	Unlike elements, attributes cannot be extended. Unlike elements, attribute order is not enforced by XML processors.
Rules and Guidelines	
Rule	**Description**
[SD3-9]	Exchange Network schemas MUST NOT use attributes in place of data elements.
[SD3-10]	Exchange Network schemas MAY use attributes for metadata.
Justification	
The use of attributes is prohibited because XML instance documents contain data exclusively, as opposed to metadata. The fact that attributes cannot contain other attributes and cannot be extended makes their usefulness very limited as well.	

Global Attributes

Global attributes are direct descendants of the root schema element. As with global elements, global attributes can be referenced within any complex datatype definition in a schema through the use of a "ref" attribute. In the following example, the InformationFormatIndicator attribute is a global attribute:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:attribute name="InformationFormatIndicator" type="xsd:string"/>
  <!—information removed for example purposes—>
  <xsd:complexType name="FacilitySiteDetailsType">
    <xsd:sequence>
      <xsd:element ref="FacilityIdentificationCode">
      <!—information removed for example purposes—>
    </xsd:sequence>
    <xsd:attribute ref="InformationFormatIndicator"/>
  </xsd:complexType>
</xsd:schema>
```

Because the InformationFormatIndicator attribute is a global attribute, a change to the InformationFormatIndicator attribute declaration (such as a change in datatype) will propagate to the definition of the FacilitySiteDetailsType datatype and to all other complex datatype definitions where the attribute is referenced. As with global elements, there may be situations where this is not the desired result.

Global Attributes

Pros and Cons	
Advantages	Global attributes can be referenced within any complex datatype definition in a schema. A change to a global attribute declaration will propagate to all complex datatype definitions where the attribute is referenced. This allows a broad-reaching change to be made in a single location in a schema, thereby lowering maintenance costs.
Disadvantages	A change to a global attribute declaration may propagate to complex datatype definitions for which the change should not apply. Additional schema updates may be required in such cases, thereby increasing maintenance costs. Use of global attributes places additional overhead on an XML processor to resolve all references.
Rules and Guidelines	
Rule	**Description**
[SD3-12]	Exchange Network schemas MUST NOT use global attributes in place of data elements
[SD3-13]	Exchange Network schemas MAY use global attributes for metadata.
Justification	

Allowing the use of global attributes in Exchange Network schema provide more flexibility to schema developers. There are no significant drawbacks to using global attributes.

Local Attributes

Local attributes are not direct descendants of the root element of a schema. Rather, they are nested inside the schema structure. Unlike global attributes, local attributes cannot be referenced within any complex datatype definition in a schema outside of the complex datatype definition where they are declared. The following example is similar to the example shown above for global elements, but the informationFormatIndicator attribute is now a local attribute:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!—information removed for example purposes—>
  <xsd:complexType name="FacilitySiteDetailsType">
    <xsd:sequence>
      <xsd:element ref="FacilityIdentificationCode">
      <!—information removed for example purposes—>
    </xsd:sequence>
    <xsd:attribute name="informationFormatIndicator" type="xsd:string"/>
  </xsd:complexType>
</xsd:schema>
```

Because the informationFormatIndicator attribute is now a local attribute, a change to the informationFormatIndicator attribute declaration will affect only the FacilitySiteDetailsType datatype.

As with local elements, it is possible to declare a local attribute in multiple places within a schema, with a different datatype in each place. This technique may be desirable in some situations.

Local Attributes

Pros and Cons	
Advantages	A change to a local attribute declaration will affect only that attribute, thereby allowing changes to be confined to a single location in a schema. This may be desirable in some situations.
Disadvantages	Local attributes cannot be referenced within any complex datatype definition in a schema outside of the complex datatype definition where they are declared. If a local attribute is declared in multiple places in a schema with the same datatype and a change to its datatype is required, a change will need to be made wherever the local attribute is declared. This can increase maintenance costs.

Rules and Guidelines	
Rule	**Description**
[SD3-15]	Exchange Network schemas MUST NOT use local attributes in place of data elements.
[SD3-16]	Exchange Network schemas MAY use local attributes for metadata.
Justification	
Allowing the use of local attributes in Exchange Network schema provide more flexibility to schema developers. There are no significant drawbacks to using local attributes.	

Cardinality of Attributes

Cardinality for attributes differs from cardinality for elements; an attribute cannot occur more than once on a given element. Therefore, there are no minOccurs or maxOccurs occurrence indicators for attributes. Instead, a "use" indicator can be specified for an attribute with one of the following values:

- Required. The attribute must appear in an XML instance document. For example:

```
<xsd:attribute name="informationFormatIndicator" type="xsd:string" use="required"/>
```

- Optional. The attribute may or may not appear in an XML instance document. This is the default value. For example:

```
<xsd:attribute name="informationFormatIndicator" type="xsd:string" use="optional"/>
```

- Prohibited. The attribute must not appear in an XML instance document. For example:

```
<xsd:attribute name="informationFormatIndicator" type="xsd:string" use="prohibited"/>
```

A "use" indicator can appear only on local attribute declarations or references to global attributes, not on global attribute declarations. It is also possible to specify a different "use" indicator value for a global attribute each place within a schema where it is referenced.

For example, a "prohibited" value may be used if an attribute is added to a schema, but the schema developer wants to prohibit the use of the attribute until a later time because of some system dependency (perhaps a database field with which the attribute is associated does not yet exist).

Cardinality of Attributes

Pros and Cons	
Advantages	The “use” indicator allows the appearance of an attribute to be enforced.
Disadvantages	There are no disadvantages to this technique.
Rules and Guidelines	
Rule	**Description**
[SD3-18]	Exchange Network schemas SHOULD use the "use" indicator.
[SD3-19]	Exchange Network schemas SHOULD NOT use the "use" indicator when the required value is the default value.
Justification	
The ability to enforce the appearance of an attribute is very valuable. It is recommended that schema developers not specify a default value for a “use” indicator (i.e., use=“optional”) because doing so can unnecessarily clutter a schema. Use of the “prohibited” value can unnecessarily complicate a schema. It is preferable to use change control techniques for situations such as the example above.	

2.4.3. Element and Attribute Grouping

The W3C Schema standard has various methods for grouping elements and attributes together. This section discusses each of these methods.

Compositors

Compositors are W3C Schema constructs that group element declarations together. There are three types of compositors in the W3C Schema standard:

- Sequence
- Choice
- All.

“Sequence” Compositor

The “sequence” compositor has been used in several examples in this document. It indicates that the elements declared inside it must appear in an XML instance document in the order declared. For example:

```
<xsd:complexType name=“FacilitySiteDetailsType”>
  <xsd:sequence>
    <xsd:element ref=“FacilityIdentificationCode”>
    <xsd:element ref=“FacilityName”>
    <xsd:element ref=“FacilityAddressDetails”>
```

```
    <!—information removed for example purposes—>
  </xsd:sequence>
</xsd:complexType>
```

If the above elements appear under the FacilitySiteDetailsType element in an XML instance document in an order other than that shown above, an XML processor will generate an error.

"Sequence" Compositor

Pros and Cons	
Advantages	The "sequence" compositor allows element order enforcement.
Disadvantages	There are no disadvantages to this technique.
Rules and Guidelines	
Rule	**Description**
[SD3-22]	Exchange Network schemas SHOULD use the "sequence" compositor.
Justification	
The ability to enforce element order is very valuable.	

"Choice" Compositor

The "choice" compositor indicates that only one of the elements declared within it can appear in an XML instance document. For example:

```
<xsd:complexType name="FacilitySiteDetailsType">
   <xsd:choice>
     <xsd:element ref="FacilityIdentificationCode">
     <xsd:element ref="FacilityName">
     <xsd:element ref="FacilityAddressDetails">
     <!—information removed for example purposes—>
   </xsd:choice>
</xsd:complexType>
```

If more than one of the above elements appear under the FacilitySiteDetailsType element in an XML instance document, an XML processor will generate an error.

"Choice" Compositor

Pros and Cons	
Advantages	The "choice" compositor allows single element choices to be enforced.
Disadvantages	There are no disadvantages to this technique.

Rules and Guidelines	
Rule	**Description**
[SD3-24]	Exchange Network schemas SHOULD use the "choice" compositor.
Justification	
As its name implies, the “choice” compositor is very useful in scenarios where only one choice can be made among a list of elements—for instance, elements that represent a series of menu choices.	

“All” Compositor

The “all” compositor indicates that the elements declared within it can appear in an XML instance document, in any order. For example:

```
<xsd:complexType name=“FacilitySiteDetailsType”>
   <xsd:all>
     <xsd:element ref=“FacilityIdentificationCode”>
     <xsd:element ref=“FacilityName”>
     <xsd:element ref=“FacilityAddressDetails”>
     <!—information removed for example purposes—>
   </xsd:all>
</xsd:complexType>
```

The above elements can appear under the FacilitySiteDetailsType element in any order, and an XML processor will not generate an error. However, with the all compositor, no element within it can appear more than once. It is therefore illegal to specify a minOccurs or maxOccurs value greater than one for any element declared within an “all” compositor.

“All” Compositor

Pros and Cons	
Advantages	The “all” compositor allows for flexible element ordering.
Disadvantages	No element within an “all” compositor can appear more than once.
Rules and Guidelines	
Rule	**Description**
[SD3-26]	Exchange Network schemas MUST NOT use the "all" compositor.
Justification	

It is important that element order be enforced, therefore the use of the “all” compositor is prohibited.

Although no element within an “all” compositor can appear more than once, the potential advantages for using the “all” compositor far outweigh the potential disadvantages.

Model Groups

Up to this point, all element groupings have used compositors in this document. There is another type of element grouping—a model group—that allows elements to be referenced within multiple complex datatypes using a single name.

Model groups must be globally defined with a group element, as shown in the following example:

```
<xsd:group name=“LocationCodes”>
  <xsd:sequence>
    <xsd:element name=“LocationCode1” type=“xsd:string”>
    <xsd:element name=“LocationCode2” type=“xsd:string”>
    <xsd:element name=“LocationCode3” type=“xsd:string”>
  </xsd:sequence>
</xsd:group>
```

As with global elements, the three elements grouped together in the above example can be referenced within any complex datatype definition within a schema using a “ref” attribute. For example:

```
<?xml version=“1.0” encoding=“UTF-8”?>
<xsd:schema xmlns:xsd=“http://www.w3.org/2001/XMLSchema”>
  <xsd:element name=“FacilitySiteDetails” type=“FacilitySiteDetailsType”/>
  <xsd:element name=“SampleLocationDetails” type=“SampleLocationDetailsType”/>
  <xsd:complexType name=“FacilitySiteDetailsType”>
    <xsd:sequence>
      <xsd:element ref=“FacilityIdentificationCode”>
      <xsd:group ref=“LocationCodes”>
      <!—information removed for example purposes—>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name=“SampleLocationDetailsType”>
    <xsd:sequence>
      <xsd:element ref=“SampleIdentificationCode”>
      <xsd:group ref=“LocationCodes”>
      <!—information removed for example purposes—>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

As noted in an earlier chapter, the global complex datatypes are advantageous mostly because a change to a global complex datatype definition will propagate to all

elements associated with that datatype within a schema. Model groups are similarly advantageous because a change to a model group declaration will propagate to all global complex datatype definitions where the model group is referenced—which in turn will propagate to all elements that are associated with those global complex datatypes.

Continuing with the above example, if an element named LocationCode4 were added to the LocationCodes model group, it would be reflected within both the FacilitySiteDetails and SampleSiteDetails content models because the LocationCode4 element would now become a subelement of both the FacilitySiteDetails and SampleSiteDetails elements.

Cardinality can also be indicated for model groups using the minOccurs and maxOccurs constraints in the same way they are used with global element references.

Model Groups

Pros and Cons	
Advantages	Model groups can be referenced in any complex datatype definition within a schema. A change to a model group declaration will propagate to all complex datatype definitions where the model group is referenced, which in turn propagate to elements. This allows a far-reaching change to be made in a single location in a schema, thereby lowering maintenance costs.
Disadvantages	As with global elements, a change to model group declaration may propagate to datatypes and elements for which the change should not apply. Additional schema updates may be required in such cases, thereby increasing maintenance costs. Use of model groups places additional overhead on an XML processor to resolve all references.
Rules and Guidelines	
Rule	**Description**
[SD3-28]	Exchange Network schemas MAY use model groups.
Justification	
Model groups allow elements to be referenced within multiple complex datatypes using a single name. Although there is a potential requirement for additional schema updates in the propagation scenario discussed above, the potential advantages for using model groups far outweigh the potential disadvantages. The potential overhead on an XML processor to resolve all references to model groups is also not of high enough concern to warrant not recommending their use.	

Attribute Groups

In the same way that model groups allow grouping of elements, attribute groups allow grouping of attributes. Attribute groups are useful when the same set of attributes is associated with multiple elements in a schema. Attribute groups must be globally defined with an attributeGroup element, as shown in the following example:

```
<xsd:attributeGroup name="sourceInformation">
<xsd:attribute name="authorName" type="xsd:string">
  <xsd:attribute name="creationDate" type="xsd:date">
  <xsd:attribute name="lastModificationDate" type="xsd:date">
</xsd:attributeGroup>
```

The attribute group in the above example can be associated with any element in a schema, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="BrochureInformation" type="BrochureInformationType"/>
  <xsd:element name="NewletterInformation" type="NewsletterInformationType"/>
  <!—information removed for example purposes—>
  <xsd:complexType name="BrochureInformationType">
    <xsd:sequence>
      <xsd:element ref="BrochureTitle">
      <!—information removed for example purposes—>
    </xsd:sequence>
    <xsd:attributeGroup ref="sourceInformation"/>
  </xsd:complexType>
  <xsd:complexType name="NewsletterInformationType">
    <xsd:sequence>
      <xsd:element ref="NewsletterTitle">
      <!—information removed for example purposes—>
    </xsd:sequence>
    <xsd:attributeGroup ref="sourceInformation"/>
  </xsd:complexType>
</xsd:schema>
```

As with model groups, the main advantage of attribute groups is that a change to an attribute group declaration will propagate to all elements with which the attribute group is associated. Continuing with the above example, if an attribute named authorEMailAddress were added to the sourceInformation attribute group, it would be reflected within both the BrochureInformation and NewsletterInformation content models because the authorEMailAddress attribute would now become associated with both the BrochureInformation and NewsletterInformation elements.

Attribute Groups

Pros and Cons	
Advantages	Attribute groups can be associated within any element in a schema. A change to an attribute group declaration will propagate to all elements with which the attribute group is associated. This allows a far-reaching change to be made in a single location in a schema, thereby lowering maintenance costs.
Disadvantages	A change to an attribute group declaration may propagate to elements for which the change should not apply. Additional schema updates may be required in such cases, thereby increasing maintenance costs. Use of attribute groups places additional overhead on an XML processor to resolve all references.
Rules and Guidelines	
Rule	**Description**
[SD3-30]	Exchange Network schemas MUST NOT use attribute groups in place of data elements.
[SD3-31]	Exchange Network schemas MAY use attribute groups for metadata.
Justification	
Model groups allow elements to be referenced within multiple complex datatypes using a single name. Although there is a potential requirement for additional schema updates in the propagation scenario discussed above, the potential advantages for using model groups far outweigh the potential disadvantages. The potential overhead on an XML processor to resolve all references to model groups is also not of high enough concern to warrant not recommending their use.	

2.5. Namespaces

Namespaces associate schema constructs with a conceptual space that defines a markup vocabulary. This chapter discusses namespaces, namespace usage, and provides rules and guidance for use of namespaces in Exchange Network schema and instance files. It is divided into two sections:

- Namespaces and how they are used within Exchange Network schemas
- Namespaces and how they are used within XML instance documents.

2.5.1. Namespace Declaration and Qualification

A namespace is declared in the root element of a schema using a namespace identifier. Schema constructs are associated with a namespace identifier through a user-defined namespace prefix, making the constructs “namespace qualified.”

In the following example, the namespace identifier is "http://www.exchangenetwork.net/schema/TRI/2" and the namespace prefix is "TRI":

```
<xsd:schema xmlns:TRI="http://www.exchangenetwork.net/schema/TRI/4">
```

This means that any construct in the schema with a name prefix of "TRI" belongs to the TRI namespace, as in the following example:

```
<element name="ChemicalIdentification" type="TRI:ChemicalIdentificationDataType">
```

Per W3C specifications, a namespace identifier must be a uniform resource identifier (URI). There are two kinds of URIs: a uniform resource locator (URL) and a uniform resource name (URN). Therefore, a namespace identifier must be either a URL or a URN. If a namespace identifier is a URL, it is not required to be a resolvable World Wide Web address.

Namespaces allow constructs with the same name but from different markup vocabularies to be used in the same schema with no adverse effects. In the following example, two ChemicalIdentityDataType datatypes are used in the same schema, but they are associated with two different namespaces. In this example, the schema declares two elements, each using the ChemicalIdentityDataType from a different namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:TRI="http://www.exchangenetwork.net/schema/TRI/4"
    xmlns:SC="urn:us:net:exchangenetwork:sc:1:0">
  <xsd:element name="TRIChemicalIdentity" type="TRI:ChemicalIdentityDataType" />
  <xsd:element name="BasicChemicalIdentity" type=SC:ChemicalIdentityDataType" />
  <!—information removed for example purposes—>
</xsd:schema>
```

If the state elements declared above were not in separate namespaces, an XML processor would generate an error. This condition is known as name collision.

2.5.2. Namespaces in Exchange Network Schema

Because namespace names are an important part of the Exchange Network schema versioning strategy, it is imperative that schema developers follow the namespace naming rules when designing schema.

Exchange network namespaces are URL formatted. The rationale for choosing a URL-formatted namespace (over a URN-formatted namespace) is as follows:

- URL formatted namespaces are simple, and intuitive to internet users and developers
- Meet namespace requirements set out by the W3C
  - By their nature URLs refer to a unique internet location
  - Within the scope of the EN, URLs pointing to `http://www.exchangenetwork.net/schema`, will be persistent
- Namespaces will be unique to flows determined by the inherent structure of the repository
- URL namespace resolution is universally supported by all web applications
  - Secondary registration is not required with IANA – reducing administrative overhead
  - On-the-spot resolution prevents naming errors – users get instantaneous feedback if they misspell a namespace by accessing the URL as a hyperlink
- The ability to resolve namespaces to a network location within the EN repository supports other EN priority initiatives including:
  - Schema management and versioning
  - Improving the Schema Repository
- Standardized implementation of URL formatted namespaces supports future development initiatives based upon resolution of namespaces (e.g. RDDL[4])

Examples of Exchange Network Namespaces are as follows:

```
http://www.exchangenetwork.net/schema/AQS/3
http://www.exchangenetwork.net/schema/RCRA/Handler/1
```

In the following example, a properly-formed Exchange Network namespace is shown:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:TRI="http://www.exchangenetwork.net/schema/TRI/4"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- information removed for example purposes -->
</xsd:schema>
```

Please note that regardless of the minor version number of the given schema, the namespace must only contain the major version number. For more information on this decision, please refer to the Schema Versioning section of this document.

Namespaces in Exchange Network Schema

Pros and Cons

[4] See http://www.rddl.org/

Advantages	The Network namespace strategy ensures consistency of namespace implementation for all Network schemas and provides consistency and clarity among the schema and exchange documentation for a given version of an exchange.
Disadvantages	There are no disadvantages to using this technique.
Rules and Guidelines	
Rule	**Description**
[SD4-1]	Exchange Network schemas MUST use namespaces.
[SD4-2]	Exchange Network schemas MUST use namespace qualification for all schema constructs.
[SD4-A]	The schema namespace name MUST be URL-formatted as "http://www.exchangenetwork.net/schema/"{ExchangeIdentifier}["/"{Category}]"/"{Version}.
[SD4-B]	Exchange Network namespaces MUST contain the Exchange Identifier term that clearly and uniquely defines the type of data being exchanged.
[SD4-C]	Exchange Network namespaces MAY contain a category term which further divides the data exchange into smaller components.
[SD4-D]	Exchange Network namespaces MUST contain a major version number as the last part of the namespace name.
[SD4-E]	Exchange Network namespaces MUST NOT contain a minor version number in the namespace name.
Justification	
The namespace requirements have been formally established by Network governance.	

Target Namespaces

Declaration of a target namespace in a schema indicates that the schema is acting as a "collector" of constructs declared within it. Declaring a target namespace in a schema ensures that all constructs within the schema will be associated with a namespace. Conversely, without a target namespace, constructs declared in the schema would not belong to any namespace. While a schema may have more than one declared namespace, only one namespace can be designated as the target namespace.

In the following example, the schema declares a target namespace which matches the declared namespace:

```
<xsd:schema xmlns:TRI="http://www.exchangenetwork.net/schema/TRI/4"
    targetNamespace="http://www.exchangenetwork.net/schema/TRI/4"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
  <!-- information removed for example purposes -->
</xsd:schema>
```

Target Namespaces

Pros and Cons	
Advantages	Declaration of a target namespace in a schema ensures that all constructs within the schema will be associated with a namespace.
Disadvantages	There are no disadvantages to this technique.
Rules and Guidelines	
Rule	**Description**
[SD4-13]	Exchange Network schemas MUST use target namespaces.
Justification	
Target namespaces are valuable because they allow a set of schema constructs to be collected into a single conceptual space. This allows the constructs to be identified as a single set of constructs.	

The W3C Schema Namespaces

The W3C Schema standard has three namespaces that contain W3C Schema constructs. Two of these namespaces contain constructs used in schemas, while the third contains constructs used in XML instance documents. The two schema construct namespaces are discussed below, and the third is discussed in a later section.

W3C Schema Namespace

The W3C Schema standard has its own namespace that contains all W3C Schema constructs used in schemas. This namespace is referred to as the W3C Schema namespace. To use W3C Schema constructs in a schema, the W3C Schema namespace must be declared in the root element using the namespace identifier "http://www.w3.org/2001/XMLSchema," as follows:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

This namespace declaration indicates to an XML processor that any construct in a schema with a namespace prefix "xsd" is a W3C Schema construct, as in the following example:

```
<xsd:element name="FacilityIdentificationCode" type="xsd:string"/>
```

Although user-defined, the prefix “xs” or “xsd” is most often used in W3C Schema literature and references as the namespace prefix for W3C Schema constructs.

W3C Schema Namespace

Pros and Cons	
Advantages	Declaring the W3C Schema namespace in a schema allows use of W3C Schema constructs.
Disadvantages	There are no disadvantages to this technique.
Rules and Guidelines	
Rule	**Description**
[SD4-5]	Exchange Network schemas MUST declare the W3C Schema namespace.
[SD4-6]	Exchange Network schemas MUST use namespace qualification for all W3C Schema constructs.
Justification	
The W3C Schema namespace must be declared in order to use W3C Schema constructs. Although namespace qualification of W3C Schema constructs can increase verbosity, the ability to easily differentiate between a W3C Schema construct and a user-defined schema construct is very valuable. Consistent use of a single namespace prefix makes it easy to identify a W3C Schema construct when viewing a schema, and promotes a common look and feel of schemas across the Exchange Network.	

W3C Schema Datatype Namespace

In addition to the W3C Schema namespace, there is a separate namespace—the W3C Schema Datatypes namespace—that contains only the W3C Schema built-in datatypes (i.e., it is a subset of the W3C Schema namespace). Its required namespace identifier is “http://www.w3.org/2001/XMLSchema-datatypes.”

If the W3C Schema Datatypes namespace is declared (but not the W3C Schema namespace), that schema can include only W3C Schema built-in datatypes and no other W3C Schema constructs.

The W3C Schema Datatypes namespace gives product developers an opportunity to include W3C Schema datatypes in their product without supporting the full range of the W3C Schema markup vocabulary (e.g., Schematron & Relax NG).

W3C Schema Datatype Namespace

Pros and Cons

Advantages	The W3C Schema Datatypes namespace gives product developers an opportunity to include W3C Schema datatypes in their product without requiring them to support the full range of the W3C Schema markup vocabulary.
Disadvantages	There are no disadvantages to this technique.
Rules and Guidelines	
Rule	**Description**
[SD4-11]	Exchange Network schemas SHOULD NOT declare the W3C Schema Datatypes namespace.
Justification	
Because the W3C Schema Datatypes namespace is a subset of the W3C Schema namespace, there is no need to declare the W3C Schema Datatypes namespace in a schema.	

External Schema References

It is possible to reference one or more additional schemas from within a schema, thereby creating a modular schema configuration. This technique is valuable because it allows constructs to be used in schemas other than the schema in which they are declared. In this section, the term “including schema” refers to the schema that includes an external schema, while the term “included schema” refers to the external schema.

Two W3C Schema constructs are used for external schema references:

- Include
- Import.

The “include” construct must be used when the including and included schemas have the same target namespace. In the following example, the target namespace of both the schema shown (the including schema) and the TRI_ChemicalIdentification_v4.0.xsd schema (the included schema) is the “http://www.exchangenetwork.net/schema/TRI/4” namespace:

```
<?xml version=“1.0” encoding=“UTF-8”?>
<xsd:schema xmlns:xsd=“http://www.w3.org/2001/XMLSchema”
    xmlns:TRI=“http://www.exchangenetwork.net/schema/TRI/4”
    targetNamespace=“http://www.exchangenetwork.net/schema/TRI/4”>
  <xsd:include schemaLocation=“TRI_ChemicalIdentification_v4.0.xsd”/>
<!—information removed for example purposes—>
</xsd:schema>
```

This means any constructs within the TRI_ChemicalIdentification_v4.0.xsd schema can be used in the above schema.

The "import" construct must be used when the including and included schemas have different target namespaces. In the following example, the target namespace of the SC_ SC_AgencyIdentity_v1.0.xsd schema (the included schema) is "urn:us:net:exchangenetwork:sc:1:0":

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:TRI="http://www.exchangenetwork.net/schema/TRI/4"
    xmlns:SC="urn:us:net:exchangenetwork:sc:1:0"
    targetNamespace="http://www.exchangenetwork.net/schema/TRI/4">
  <xsd:import schemaLocation="SC_AgencyIdentity_v1.0.xsd"
    namespace="urn:us:net:exchangenetwork:sc:1:0" />
  <!—information removed for example purposes—>
</xsd:schema>
```

Any constructs within the SC_AgencyIdentity_v1.0.xsd schema can be used in the above schema; however, because these schemas have different target namespaces, the target namespace of the SC_AgencyIdentity_v1.0.xsd schema must be declared in the root element of the including schema.

External Schema References

Pros and Cons	
Advantages	External schema references enable the creation of a modular schema configuration.
Disadvantages	An XML processor may have a limit on the number of schemas that can be externally referenced within a single schema. Therefore, there is a risk that the number of externally referenced schemas may exceed that limit.
Rules and Guidelines	
Rule	**Description**
[SD4-15]	Exchange Network schemas SHOULD reference external schemas.
[SD4-16]	Exchange Network schemas MAY use the include construct.
[SD4-17]	Exchange Network schemas MAY use the import construct.
Justification	
The rules regarding external schema references are consistent with the guidance provided in the section on schema configuration and documentation. It is not anticipated that the number of externally referenced schemas used by the Exchange Network will exceed the limit for any given XML processor.	

Single or Multiple Namespaces

In some cases, it is appropriate to use multiple namespaces in Exchange Network schema. Some acceptable scenarios where multiple namespaces may occur in schema include situations when:

- the Shared Schema Components v1.0 are referenced in a target schema,
- portions of schemas developed outside the Exchange Network are used (such as GML schema),
- The target schema implements multiple "modules", each in a separate namespace as described in the Schema Modularization rules and guidelines.

Unless consistent with the Network namespace guidance, schemas developed specifically for the Exchange Network should not implement multiple namespaces within the same exchange.

Single and Multiple Namespaces

Pros and Cons – Single Namespace Configuration	
Advantages	A single-namespace configuration is very simple. A single-namespace configuration ensures consistent use of the include construct for external schema references.
Disadvantages	A single-namespace configuration increases the risk of name collision. This requires more work on the part of schema developers to ensure this does not occur.
Pros and Cons – Multiple Namespace Configuration	
Advantages	A multiple-namespace configuration decreases the risk of name collision. A multiple-namespace configuration allows the structure or organization of a markup vocabulary to be easily represented.
Disadvantages	A multiple-namespace configuration can be very complex depending on the number of namespaces used. A multiple-namespace configuration requires use of both the include and import constructs for external schema references. There is a risk that the wrong construct may be used in an external schema reference, thereby generating an XML processor error.
Rules and Guidelines	
Rule	**Description**
[SD4-27]	Exchange Network schemas MAY use multiple namespaces.
Justification	

The potential advantages gained from the use of multiple namespaces outweigh the potential complexities. Use of multiple namespaces allows the flexibility to address media, functional, and jurisdictional areas. This will allow namespace managers to develop their own constructs that are specific to their area, while still utilizing the higher level namespaces when necessary. This is discussed further in the section on schema configuration and documentation.

Default Namespaces

Declaration of a default namespace in a schema provides an efficient way to associate schema constructs with a specific namespace without the need for adding an explicit namespace prefix to each construct. While a schema may have more than one declared namespace, only one namespace can be designated as the default namespace. The W3C XML Schema specification does not require that a default namespace be declared in a schema.

A default namespace is declared simply by omitting the namespace prefix in a namespace declaration, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:TRI="http://www.exchangenetwork.net/schema/TRI/4"
    xmlns="http://www.exchangenetwork.net/schema/TRI/4"
    targetNamespace="http://www.exchangenetwork.net/schema/TRI/4">
  <!—information removed for example purposes—>
</xsd:schema>
```

In the following example, the FacilityIdentificationCode element belongs to the default namespace ("http://www.exchangenetwork.net/TRI/4") because it is not namespace qualified:

```
<xsd:element name="FacilityIdentificationCode" type="xsd:string"/>
```

"Namespace coercion" is a condition that occurs when all of the following conditions are true:

- A schema that has no target namespace is included in a schema that has a target namespace.
- Constructs in the including schema that belong to the schema's target namespace are not namespace qualified.

This condition is known as namespace coercion because the constructs in the included schema are "coerced" to become part of the including schema's namespace. In the following example, the default namespace is the target namespace. Therefore, if the FacilityIdentification.xsd schema does not have a target namespace, all constructs included within it would become part of the "urn:us:net:exchangenetwork" namespace by way of namespace coercion:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.exchangenetwork.net/schema/TRI/4"
    targetNamespace="http://www.exchangenetwork.net/schema/TRI/4">
  <xsd:include schemaLocation="FacilityIdentification.xsd"/>
  <!—information removed for example purposes—>
</xsd:schema>
```

With namespace coercion, it is impossible to visually discern the origin of schema constructs by examining the including schema (further external research would be required). For example, if multiple schemas were externally referenced in the above example (meaning each had a target namespace of "http://www.exchangenetwork.net/schema/TRI/4" or no target namespace), it would not be possible to differentiate between a construct that came from an included schema that had a target namespace or one that had no target namespace. All such constructs would not be namespace qualified.

Default Namespaces

Pros and Cons	
Advantages	A default namespace reduces verbosity in a schema.
Disadvantages	Declaration of a default namespace in a schema increases ambiguity because the omission of namespace prefixes makes it more difficult to identify the namespace where a construct belongs. Use of default namespaces can cause namespace coercion.
Rules and Guidelines	
Rule	**Description**
[SD4-23]	Exchange Network schemas MUST NOT use default namespaces.
Justification	
The potential advantages gained from the use of multiple namespaces outweigh the potential complexities. Use of multiple namespaces allows the flexibility to address media, functional, and jurisdictional areas. This will allow namespace managers to develop their own constructs that are specific to their area, while still utilizing the higher level namespaces when necessary. This is discussed further in the section on schema configuration and documentation.	

2.5.3. Namespaces in XML Instance Documents

All discussion up to now has focused on the declaration and use of namespaces in schemas. Namespaces are also declared and used in XML instance documents. The following concepts are covered in this section:

- **XML instance document validation**—methods for validating an XML instance document against a schema
- **The W3C Schema Instance namespace**—a set of namespaces specific to the W3C Schema standard
- **Namespace schemaLocation Attribute** —the declaration of namespaces in XML instance documents and designation of constructs belonging to those namespaces
- **Namespace scope**—the range of applicability of namespaces within XML instance documents and methods for altering this range.

XML Instance Document Validation

There are two principle ways in which an XML instance document can be associated with a schema for validation purposes:

1. The validating system selects the schema based on its exact location as specified in the XML instance document.
2. The validating system selects the appropriate schema based on some external configuration

In the first approach, the schema location (which may be a URL or file path) can be listed in the root element of the XML instance document, as follows:

```
<?xml version="1.0"?>
<TRI:TRI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.exchangenetwork.net/schema/TRI/4
  TRI_TRI_v4.0.xsd">
```

The schemaLocation attribute is a W3C Schema construct that associates an XML instance document with a schema. It is used only when a schema has a target namespace. The "http://www.exchangenetwork.net/schema/TRI/4" namespace identifier is the target namespace of the TRI_TRI_v4.0.xsd schema.

If a schema does not have a target namespace, the noNamespaceSchemaLocation construct must be used:

```
<?xml version="1.0"?>
<TRI:TRI xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="TRI_TRI_v4.0.xsd">
```

An XML processor is not required by the W3C Schema standard to recognize the schemaLocation and noNamespaceSchemaLocation constructs. Therefore, an XML processor can ignore the schema listed in an XML instance document and validate the

XML instance document against an entirely different schema while still conforming to the W3C Schema standard.

In Exchange Network schema, the value specified in the schemaLocation attribute must match the namespace URL. The following example shows an example of proper use of the schema location attribute:

```
<?xml version="1.0" encoding="UTF-8"?>
<FacilityContact xmlns="http://www.exchangenetwork.net/schema/NPDES/1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.exchangenetwork.net/schema/NPDES/1
    http://www.exchangenetwork.net/schema/NPDES/1">
  <FirstName>Joe</FirstName>
  <LastName>Smith</LastName>
  <!-- information removed for example purposes —>
</FacilityContact>
```

Note that the schemaLocation attribute is formatted to repeat the namespace name and the schema location, separated by a space character. This is the proper format according to W3C specifications. In this example, the first string defines the namespace name while the second string identifies the resolvable location where the schema can be retrieved. The Exchange Network repository will be configured to return the Default File (index.xsd) for the namespace which will enable the parser to validate the instance document against the Message Schema(s) in the given namespace.

This rule has a basis in historical problems whereby instance documents contained the local path to the schema on the sender's system (i.e. c:\myschema\index.xsd) in the path portion of the attribute value. The receiver cannot locate the schema at the specified address, causing schema validation to fail.

XML Instance Document Validation

Pros and Cons	
Advantages	Validation of an XML instance document ensures that its contents satisfy all requirements within the schema to which it validates.
Disadvantages	Validation of an XML instance document introduces an additional level of complexity to a process flow. If the location of the schema to which an XML instance document validates is listed in the root element of the XML instance document and the location of the schema changes, all XML instance documents that validate to that schema must be updated if they are to be processed in the future. An XML processor is not required by the W3C Schema standard to recognize the schemaLocation and noNamespaceSchemaLocation constructs. In some situations, this may prevent an XML instance document from being validated against a schema.

Rules and Guidelines	
Rule	**Description**
[SD4-35]	Exchange Network XML instance documents MUST be validated against a schema during processing.
[SD4-36]	Exchange Network XML instance documents SHOULD list the storage location of the schema where the XML instance document validates in the root element.
[SD4-H]	If a schemaLocation is specified in an XML instance document, the location MUST match the namespace URL address.
[SD4-38]	Exchange Network XML instance documents MUST NOT use the noNamespaceSchemaLocation construct when listing the storage location of the schema to which the XML instance document validates.
Justification	
Validation of XML instance documents will help ensure data integrity within process flows and data storage. The potential advantages gained from validation of XML instance documents outweigh the potential increase in complexity. Including the storage location of the schema to which an XML instance document validates in the root element of the XML instance document makes the XML instance document and the schema "tightly coupled." Although this may be desirable in some situations (e.g., if the schema location is not expected to change), it may be undesirable in others (e.g., if the schema location may change). Because schema construct visibility is not as important for document-centric schemas as for Exchange Network schemas, data integrity (and, therefore, XML instance document validation) is not as critical for document-centric schemas. The rules regarding the schemaLocation and noNamespaceSchemaLocation constructs are consistent with the guidance provided in the section of this chapter on target namespaces of this chapter.	

Namespace Declaration and Qualification

Elements and attributes in XML instance documents can be namespace qualified. As with schemas, a namespace is declared in the root element of an XML instance document schema through the use of a namespace identifier along with a user-defined namespace prefix. In the following example, all elements and attributes that belong to the target namespace of the index.xsd schema have a namespace prefix of "TRI" in the XML instance document:

```
<?xml version="1.0" encoding="UTF-8"?>
<TRI:TRI xsi:schemaLocation="http://www.exchangenetwork.net/schema/TRI/4 index.xsd"
    xmlns:TRI="http://www.exchangenetwork.net/schema/TRI/4"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
  <TRI:Submission>
    <TRI:TRISubmissionIdentifier>123</TRI:TRISubmissionIdentifier>
    <TRI:Facility>
      <!-- information removed for example purposes —>
    <TRI:Facility>
  </TRI:Submission>
</TRI:TRI>
```

It should be noted that

- the *namespace identifier* in an XML instance document must be the same as the namespace identifier for the target namespace in the schema, and
- the *namespace prefix* in an XML instance document does not need to be the same as the namespace prefix for the target namespace in the schema.

Elements and attributes in XML instance documents can be namespace qualified only if they belong to the target namespace of the schema that validates the XML instance document. Therefore, all global elements and attributes must be namespace qualified. However, the requirement for local elements and attributes that belong to the target namespace of the schema depends on the setting of a "switch mechanism" in the schema that uses the following two indicators:

- elementFormDefault
- attributeFormDefault.

The elementFormDefault indicator controls the namespace qualification of local elements, while the attributeFormDefault indicator controls the namespace qualification of local attributes. Both of these indicators appear as attributes of the root element of a schema, and each can have a value of "qualified" or "unqualified" (default). For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.exchangenetwork.net/schema/TRI/4"
    elementFormDefault="qualified" attributeFormDefault="qualified">
```

These declarations require that all local elements and attributes in the target namespace of the schema be namespace qualified in an XML instance document.

Namespace Declaration and Qualification

Pros and Cons

Advantages	Namespace qualification of elements and attributes in XML instance documents identifies the namespace where they belong. Namespace qualification of elements and attributes in XML instance documents allows elements or attributes with the same name but from different markup vocabularies to be used in the same XML instance document with no adverse effects.
Disadvantages	Namespace qualification of elements and attributes can increase verbosity in an XML instance document.
Rules and Guidelines	
Rule	**Description**
[SD4-43]	Exchange Network XML instance documents MUST use namespace qualification for all elements.
Justification	
The potential advantages gained from the use of multiple namespaces outweigh the potential complexities. Use of multiple namespaces allows the flexibility to address media, functional, and jurisdictional areas. This will allow namespace managers to develop their own constructs that are specific to their area, while still utilizing the higher level namespaces when necessary. This is discussed further in the section on schema configuration and documentation.	

The W3C Schema Instance Namespace

The W3C Schema standard has its own namespace, referred to as the W3C Schema Instance namespace, which contains all W3C Schema constructs used in XML instance documents (schemaLocation, noNamespaceSchemaLocation, type, and nil). To use such constructs, the W3C Schema Instance namespace must be declared in the root element of an XML instance document using the namespace identifier "http://www.w3.org/2001/XMLSchema-instance":

```
<?xml version="1.0" encoding="UTF-8"?>
<WQX:WQX xmlns:WQX="http://www.exchangenetwork.net/schema/wqx/2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.exchangenetwork.net/schema/wqx/2
    http://www.exchangenetwork.net/schema/wqx/2/0/WQX_WQX_v2.0.xsd">
  <!-- information removed for example purposes —>
</WQX:WQX>
```

Although user-defined, the prefix "xsi" is most often used in W3C Schema literature and references as the namespace prefix for W3C Schema Instance constructs.

W3C Schema Instance Namespace

Pros and Cons

Advantages	Declaring the W3C Schema Instance namespace in an XML instance document allows the use of W3C Schema Instance constructs.
Disadvantages	There are no disadvantages to this technique.
Rules and Guidelines	
Rule	**Description**
[SD4-45]	Exchange Network XML instance documents MUST declare the W3C Schema Instance namespace when W3C Schema Instance constructs are used.
[SD4-46]	Exchange Network XML instance documents SHOULD use "xsi" as a namespace prefix for all W3C Schema Instance constructs.
Justification	
The W3C Schema Instance namespace must be declared in an XML instance document in order to use W3C Schema Instance constructs. Consistent use of a single namespace prefix makes it easy to identify a W3C Schema construct when viewing an XML instance document, and promotes a common look and feel of XML instance documents across the Exchange Network.	

Namespace Scope

As with variables in programming languages, namespaces in XML instance documents have a scope of applicability in an XML instance document. The scope of a namespace applies to the declared element (which may be the root element) and all content within that element. In the following example, the scope of the WQX namespace is the entire XML instance document:

```
<?xml version="1.0" encoding="UTF-8"?>
<EN:MyRootElement xmlns:EN="http://www.exchangenetwork.net/schema/EN/1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.exchangenetwork.net/schema/EN/1
    http://www.exchangenetwork.net/schema/EN/1/0/EN_EN_v1.0.xsd">
  <!-- information removed for example purposes —>
</EN:MyRootElement>
```

A namespace can also be declared on an element other than the root element; this is known as a local namespace declaration. In the following example, the namespace identifier “http://www.state.va.us/xml” represents a local namespace identification:

```
<?xml version="1.0" encoding="UTF-8"?>
<EN:MyRootElement xmlns:EN="http://www.exchangenetwork.net/schema/EN/1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.exchangenetwork.net/schema/EN/1
    http://www.exchangenetwork.net/schema/EN/1/0/EN_EN_v1.0.xsd">
```

```
  <VA:MyChildElement
      xmlns:VA:"http://www.state.va.us/xml">Value</VA:MyChildElement>
</EN:MyRootElement>
```

In this example, the scope of the "http://www.state.va.us/xml" namespace is the VA:MyChildElement element (along with its attributes and sub-elements, if it contained any).

This arrangement is necessary when using the Exchange Network Header as a XML message wrapper. Since the Header declares its own namespace, it is necessary to localize the namespace declaration in the root element of the embedded XML message. See the documentation on the Exchange Network Header for more information.

Local Namespaces

Pros and Cons	
Advantages	Local namespace declarations confine namespace declarations to the smallest area possible in an XML instance document. This may translate into more efficient processing by an XML processor.
Disadvantages	Local namespace declarations make it more difficult to visually identify all namespaces declared in an XML instance document because the namespace declarations are scattered throughout the XML instance document.
Rules and Guidelines	
Rule	**Description**
[SD4-49]	Exchange Network XML instance documents MUST NOT use local namespace declarations.
Justification	
The W3C Schema Instance namespace must be declared in an XML instance document in order to use W3C Schema Instance constructs. Consistent use of a single namespace prefix makes it easy to identify a W3C Schema construct when viewing an XML instance document, and promotes a common look and feel of XML instance documents across the Exchange Network.	

2.6. Schema Configuration and Documentation

This chapter discusses the Exchange Network schema configuration architecture, nested includes and documentation within Exchange Network schemas.

2.6.1. Schema Modularization

Schema modularization refers to the way schema files are segmented to maximize readability and manageability, and to simplify reuse. The modularization approach is

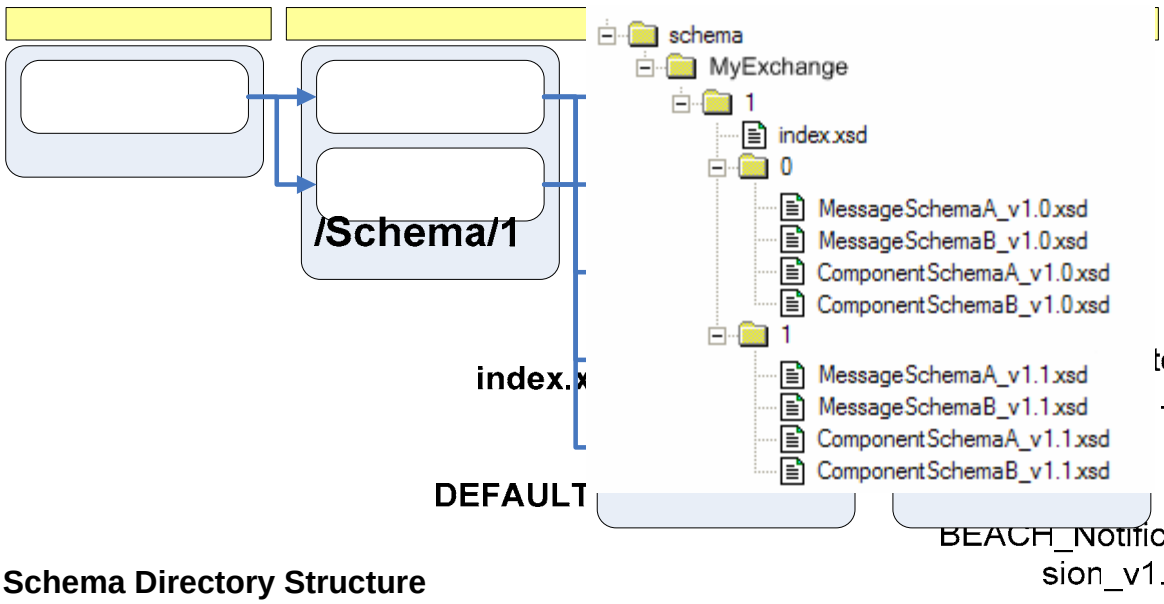
integral with the schema versioning and schema file naming rules as discussed in section 2.7.

The Exchange Network defines five types of schema files based on content and usage:

Schema Type	Definition
Default Schema	A schema with the name "index.xsd". This schema points to the root schema for the given namespace using the W3C *include* construct. This file allows the recipient to load all definitions for a namespace without needing to know the filename(s) of the root schema(s). Additionally, it allows viewers of the Exchange Network repository to easily locate the root schema by reading the name of the file(s) referenced in the schema's *include* construct. The Exchange Network schema repository is configured to return this document when the namespace URL is requested.
Message Schema	A schema which is used to convey a payload in an Exchange Network transaction. Each namespace must have at least one message schema but may have more than one. A Message Schema references (includes) one or more component schema.
Component Schema	A schema which defines a data structure. One or more component schemas are referenced in one or more message schema to build a complex representation of data. Component schema may reference (include) other component schema.
Local Shared Schema	A schema which defines a list of generic simple or complex datatypes which are referenced within one or more component schema in the namespace. Local shared schemas are typically simple in nature and do not "include" any other schema. They are the lowest level in the Network schema hierarchy. In some instances, a namespace may not have any local shared schema. There should only be one local shared schema in a given namespace.
EN Shared Schema Components (SSCs)	Schema components that have been standardized by the Exchange Network for use in all registered schema that need to define a common feature such as an address, geographic coordinate or facility. All schema developers must evaluate the suitability of existing SSCs for inclusion into new or modified schema. SSCs reduce effort by allowing developers to consume standard XML structures into their schema without having to reinvent them. SSCs are the product of the Exchange Network Core Reference Model (CRM) project[5].

The following diagram illustrates the relationship between each schema type:

[5] http://www.exchangenetwork.net/dev_schema/crm.htm



Schema Directory Structure

Network schema must be organized into a specific directory structure. The structure requires that the default schema (index.xsd) be placed in a folder with a name that matches the major version number. A subfolder, named after the minor version number of the schema then contains the remainder of the schema files. This arrangement is integral to the Network schema versioning strategy.

The diagram below illustrates the arrangement of schema within the Network repository when multiple minor versions have been published:

In this figure, the directory structure located at http://www.exchangenetwork.net is shown. An example Exchange (named 'MyExchange') has a major version number of 1, with two minor versions. The major version folder contains the index.xsd, which includes the root schema file from the latest minor version, which is version 1.1. Each minor version root schema includes the message and component schema files contained within the same minor version folder.

The target namespace for this example would be:

http://www.exchangenetwork.net/schema/MyExchange/1

while the locations of the minor version subfolders in the repository would be:

http://www.exchangenetwork.net/schema/MyExchange/1/0
http://www.exchangenetwork.net/schema/MyExchange/1/1

Default Delivery of Root Schema

To support delivery of the root schema, the default document served by the web server for all exchanges located within the http://www.exchangenetwork.net/schema/ directory will be index.xsd. The web server delivers the default schema (index.xsd) in response to all http requests that are not fully specified (i.e., do not end with a file name).

This arrangement ensures that an XML instance file may use a schemaLocation that matches the namespace name. If each new minor version follows backward compatibility guidelines, instance documents based on older minor versions will continue to validate against the latest schema returned by the Network repository.

Schema Modularization

Pros and Cons	
Advantages	This structure is compatible with the architecture of the Exchange Network schema repository and versioning strategy.
Disadvantages	There are no disadvantages to this technique
Rules and Guidelines	
Rule	**Description**
[SD5-1]	Message-level Schemas SHOULD be used.
[GD1-D]	Message-level root elements MUST be defined in the Message Schema.
[SD5-R]	Exchange Network schema MUST be modularized into default, message, component, and shared schema as described in schema guidelines.
[SD5-S]	The exchange default schema (index.xsd) MUST be stored in a directory with a name that matches the exchange major version number and the message, component, and shared schema files MUST be stored in a subdirectory matching the exchange minor version number.
Justification	
The segmentation of schema into message, component, and shared schema files ensures that they are easy to read and maintain.	

2.6.2. Multiple References to Global Complex Elements

Design Rules and Conventions dictate that all elements must be declared as global (SD3-1). These elements must then be referenced by other elements in order to create complex data structures. For example, a schema developer may create a global, complex element named PermittedFeature which describes some physical object which is subject to environmental regulation. In the following example, the PermittedFeature element is a global element:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema >
  <xsd:element name="PermittedFeature">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="PermittedFeatureIdentifier"/>
```

```
        <xsd:element ref="PermittedFeatureName"/>
        <xsd:element ref="PermittedFeatureDescription"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="PermittedFeatureIdentifier" type="xsd:string"/>
  <xsd:element name="PermittedFeatureName" type="xsd:string"/>
  <xsd:element name="PermittedFeatureDescription" type="xsd:string"/>
</xsd:schema>
```

The developer may then wish to implement the PermittedFeature element as a child of both a Facility element and a Permit element within the schema. Assume that the developer wishes to allow the PermittedFeature element to occur more than once in both instances. In the following example, both the Facility element and the Permit element reference the global PermittedFeature element:

```
<xsd:element name="Permit">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="PermittedFeature" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="Facility">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="PermittedFeature" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

While this scenario is quite practical and does not violate W3C standards, it may create problems for exchange implementers. The source of the problem is that two complex elements with repeating content (PermittedFeature in this case) share the same name.

Many developer tools attempt to represent each of these objects as “tables” in the relational database paradigm (or more accurately, two classes in the same namespace). This structure would result in the tool attempting to create two “tables” with the same name; one as a child of Permit and one as a child of Facility. This results in a name collision which cannot be resolved by the tool. Two approaches to resolving this situation are provided below.

Approach #1 – Create separate elements for each reference

Instead of creating a global element named PermittedFeature, create a global complex type named PermittedFeatureDataType. Then create two global elements with different names

which reference the newly created type. The elements could be named PermitPermittedFeature and FacilityPermittedFeature. This would enable the same structure to be reused but given different names based upon its implementation. In the following example, the developer uses the approach outlined above:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema>
  <!-- complex global types -->
  <xsd:complexType name="PermittedFeatureDataType">
    <xsd:sequence>
      <xsd:element ref="PermittedFeatureIdentifier"/>
      <xsd:element ref="PermittedFeatureName"/>
      <xsd:element ref="PermittedFeatureDescription"/>
    </xsd:sequence>
  </xsd:complexType>

  <!-- simple global elements -->
  <xsd:element name="PermittedFeatureIdentifier" type="xsd:string"/>
  <xsd:element name="PermittedFeatureName" type="xsd:string"/>
  <xsd:element name="PermittedFeatureDescription" type="xsd:string"/>

  <!-- complex global elements -->
  <xsd:element name="PermitPermittedFeature" type="PermittedFeatureDataType"/>
  <xsd:element name="FacilityPermittedFeature" type="PermittedFeatureDataType"/>

  <xsd:element name="Permit">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="PermitPermittedFeature" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="Facility">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="FacilityPermittedFeature" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

This is a good approach when the same data structure exists for two separate types of things. For example, a given schema might have a Facility Address and a Contact Address. Both have the same structure, but have different meaning based on the implementation.

Approach #2 – Use KEY and KEYREF to create multiple references to a single element list

The second option is to define a single list of referenced items within the schema. Items within the list can then be referenced using the W3C schema KEY and KEYREF constructs. This approach is very similar to what might be done in a relational database when a given table is a child to multiple other tables.

In the following example, the schema defines unbounded Facility and Permit elements. Assume that both the Facility and Permit may have a contact (specified in FacilityContactIdentifier and PermitContactIdentifier respectively). Also assume that the contact may be the same person for both the permit and facility. The developer has chosen to use the KEY/KEYREF technique to link a permit and facility contact to an item in the ContactList element. Note that this example uses a local ContactList element for improved readability although global elements are required for Exchange Network schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema>
  <xsd:element name="NPDES">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Facility" maxOccurs="unbounded"/>
        <xsd:element ref="Permit" maxOccurs="unbounded"/>
        <xsd:element name="ContactList">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Contact" maxOccurs="unbounded">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="ContactIdentifier"/>
                    <xsd:element name="ContactName"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:keyref name="FacilityContactForeignKey" refer="ContactKey">
      <xsd:selector xpath="Facility"/>
      <xsd:field xpath="FacilityContactIdentifier"/>
    </xsd:keyref>
    <xsd:keyref name="PermitContactForeignKey" refer="ContactKey">
      <xsd:selector xpath="Permit"/>
      <xsd:field xpath="PermitContactIdentifier"/>
    </xsd:keyref>
    <xsd:key name="ContactKey">
      <xsd:selector xpath="ContactList/Contact"/>
```

```
      <xsd:field xpath="ContactIdentifier"/>
    </xsd:key>
  </xsd:element>
  <xsd:element name="Facility">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="FacilityIdentifier" type="xsd:integer"/>
        <xsd:element name="FacilityName" type="xsd:string"/>
        <xsd:element name="FacilityContactIdentifier"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Permit">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="PermitIdentifier" type="xsd:integer"/>
        <xsd:element name="PermitNumber" type="xsd:string"/>
        <xsd:element name="PermitType" type="xsd:string"/>
        <xsd:element name="PermitContactIdentifier"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The following example shows an instance file in which a single contact is linked to both a facility and a permit:

```
<?xml version="1.0" encoding="UTF-8"?>
<NPDES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Facility>
    <FacilityIdentifier>111</FacilityIdentifier>
    <FacilityName>ACME Industries, Inc.</FacilityName>
    <FacilityContactIdentifier>123</FacilityContactIdentifier>
  </Facility>
  <Permit>
    <PermitIdentifier>1111</PermitIdentifier>
    <PermitNumber>WA0000123</PermitNumber>
    <PermitType>Individual Permit</PermitType>
    <PermitContactIdentifier>123</PermitContactIdentifier>
  </Permit>
  <ContactList>
    <Contact>
      <ContactIdentifier>123</ContactIdentifier>
      <ContactName>Joe Smith</ContactName>
    </Contact>
  </ContactList>
</NPDES>
```

If the instance document contained a FacilityContactIdentifier or PermitContactIdentifier other than "123", the document would not pass schema validation. Note that working with KEY and KEYREF can be challenging because the developer must be familiar with XPath expressions. The KEY and KEYREF constructs also have important scope limitations which must be understood by the developer. Furthermore, validation of XML documents that use KEY and KEYREF takes significantly more time than those do not. This is due to the fact that the parser must scan the entire document to ensure the uniqueness of every KEY field. For this reason, developers may consider using Approach #1, especially when it is anticipated that large messages are expected using the schema.

Either solution presented here is an acceptable method of handling multiple references to a complex global structure. The developer will need to determine which method is most suitable for the given situation.

Multiple References to Global Complex Elements

Pros and Cons	
Advantages	Using either technique may reduce the effort needed by exchange developers to map data between a relational database and XML.
Disadvantages	This technique may add complexity to schema.
Rules and Guidelines	
Rule	**Description**
[GD1-A]	Elements with recurring, complex content SHOULD NOT be referenced more than once within the same root document, even if the element exists in different schema files. When the same structure needs to be reused in multiple areas, either create a new element with a different tag name utilizing a common datatype or use the KEY and KEYREF construct to represent the element in a single list.
Justification	
While additional effort is required to implement either technique in XML schema, the probability of reduced effort for exchange developers and implementers is expected to outweigh the added schema development costs.	

2.6.3. Shared Schema Components (SSCs)

The Exchange Network Shared Schema Components (SSCs) are a product of the Exchange Network's Core Reference Model (CRM), which provides groupings of related data elements and data blocks into what are referred to as Major Data Groups. These Major Data Groups more fully describe business areas, functions, and entities where EPA and its Partners have an environmental interest.

SSCs are reusable XML schemas that organize related data elements common to multiple environmental data flows. Shared schema components:

- Incorporate Environmental Data Standards Council (EDSC) data standards for data element grouping, data element names, and definitions
- Facilitate the creation of XML schema for environmental data flows
- Improve the quality of exchanged data

The Network Operations Board (NOB) released a decision memorandum in October, 2005 mandating the usage of SSC in schema where possible. As is made clear by the memorandum, it is imperative that schema developers thoroughly examine the elements and datatypes available in the SSCs and evaluate their fitness for use in the target schema.

There are three levels in which SSCs may be integrated into a target schema. All options begin by including one or more SSC schemas in the target schema. Each option is listed below:

High Integration Instances where SSC elements or data types with complex content are directly integrated into the target schema without modification.

Medium Integration Instances where SSC elements or data types with complex content are modified through the process of XML extension and/or restriction before being included into the target schema.

Low Integration Instances where elements or data types with simple content are integrated into the target schema.

In the following example, the SSC v2.0 FacilitySiteName and LocationAddress schemas are integrated into the definition of a Facility element in the example schema:

```
<xsd:schema xmlns:TRI="http://www.exchangenetwork.net/schema/TRI/2"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="TRI_MailingAddress_v2.0.xsd"/>
  <xsd:include schemaLocation="TRI_GeographicLocationDescription_v2.0.xsd"/>
  <xsd:include schemaLocation="http://www.en.net/schema/SC/SC_SimpleContent_v2.0.xsd"/>
  <xsd:include schemaLocation="http://www.en.net/schema/SC/SC_LocationAddress_v2.0.xsd"/>
  <xsd:element name="Facility" type="TRI:FacilityDataType"/>
  <xsd:complexType name="FacilityDataType">
    <xsd:sequence>
      <xsd:element ref="TRI:FacilityIdentifier" minOccurs="0"/>
      <xsd:element ref="TRI:FacilitySiteName" minOccurs="0"/>
      <xsd:element ref="TRI:LocationAddress" minOccurs="0"/>
      <xsd:element ref="TRI:MailingFacilitySiteName" minOccurs="0"/>
      <xsd:element ref="TRI:MailingAddress" minOccurs="0"/>
      <xsd:element ref="TRI:FacilitySICDetails" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="TRI:GeographicLocationDescription" minOccurs="0"/>
      <xsd:element ref="TRI:ParentCompanyNameNAIndicator" minOccurs="0"/>
```

```
      <xsd:element ref="TRI:ParentCompanyNameText" minOccurs="0"/>
      <xsd:element ref="TRI:ParentDunBradstreetCode" minOccurs="0"/>
      <xsd:element ref="TRI:FacilityDunBradstreetCodeDetails" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <!--information removed for example purposes -->
</xsd:schema>
```

Note that FacilitySiteName is an element declared in SC_SimpleContent_v2.0.xsd. The LocationAddress element is defined in the SC_LocationAddress_v2.0.xsd schema. Note that because SSC 2.0 do not have a declared namespace, they assume the namespace of the target schema.

Shared Schema Components

Pros and Cons	
Advantages	Usage of Shared Schema Components promotes consistency between exchanges. Implementing Shared Schema components can help ensure conformance with environmental data standards. Shared components can simplify development of new exchanges, especially when the exchange does not target a specific data system. Using predefined data blocks saves schema developers from creating structures from scratch.
Disadvantages	Shared Schema Components may differ substantially from the design of a specific target data system, complicating data mapping. Shared Schema Components may not implement needed data elements or specific element restrictions required for an exchange. The process of examining the library of shared components for possible matches will add effort on the part of a schema developer.
Rules and Guidelines	
Rule	**Description**
[SD5-A]	Developers MUST use SSC in schema when they are an appropriate fit for the targeted business process.
[SD5-B]	Developers SHOULD create custom datatypes and elements only after determining that no existing SSC adequately describes the given construct.
[SD5-C]	Existing schema SHOULD be evaluated for SSC compatibility and subsequently updated to include SSC elements and/or datatypes at the time of the next revision.
[SD5-D]	Developers SHOULD use XML datatype extension and restriction to modify SSC types.

Justification
Shared schema components promote consistency, embody data standards, and can simplify development of new schemas.

2.6.4. Reuse of Externally-developed Schemas

Most often, schemas used for Network transactions are typically designed and developed specifically for the Network using Network rules and guidelines. In some cases, schemas or portions of schema developed by external groups are used for Network exchanges. This section provides guidelines for using schema or schema components developed by external groups.

Voluntary Standards Body Schemas

Voluntary standards body schemas include schema constructs from a markup vocabulary defined by a voluntary standards body. There are various voluntary standards body efforts currently defining open libraries of schema constructs. Examples of horizontal (cross-industry) efforts include

- OASIS Universal Business Language (UBL), and
- American National Standards Institute (ANSI) X12.

Examples of vertical (inter-industry) efforts include

- Petroleum Industry Data Exchange (PIDX), and
- Geography Markup Language (GML).

Exchange developers are encouraged to use voluntary standards body schemas (or portions thereof) from horizontal efforts, although it is not required.

Pros and Cons	
Advantages	Use of voluntary standards body schemas enables Exchange Network to use open libraries of schema constructs. Use of a voluntary standards body's markup vocabulary in XML efforts can increase interoperability between Exchange Network and industry trading partners.
Disadvantages	Because various voluntary standards body efforts are taking place, it is probable that the many different emerging markup vocabularies will create a proliferation of semantically synonymous constructs. This can make data harmonization more difficult.
Rules and Guidelines	
Rule	**Description**

[SD5-7]	Appropriate Voluntary Standard Body Schemas SHOULD be adopted, when appropriate.
Justification	
Data standards support the efficient and accurate exchange of data and help secondary users to understand, interpret, and use data appropriately[6]. The use of Voluntary Standard Body Schemas is in accordance with OMB Circular A-119[7].	

Other Externally-developed Schemas

Schemas may be developed by agencies or organizations initially for use outside the Exchange Network, but later wish to be used for Network exchanges. Although it is recommended that constructs in shared Exchange Network schemas be used to the greatest extent possible, there are times when it is more appropriate to use externally-targeted schemas or schema constructs because

- the constructs represent information specific to the agency or organization,

  or

- it may be unduly burdensome to harmonize schema constructs for the Exchange Network.

Externally-developed schemas will likely not conform to Network schema design rules and guidelines and consequently, will not conform to the Network versioning and supporting document version alignment strategy, namespaces, and so on. Schema developers and exchange implementers are encouraged to weigh the benefits of reusing existing, externally-developed schema for Network exchanges with the challenges of managing schema that do not conform to other Network schemas.

Pros and Cons	
Advantages	Reuse of externally-developed schemas can be implemented without any changes to the schema. Can potentially increase interoperability between the Exchange Network and existing systems.
Disadvantages	Externally-developed schemas will likely not conform to Network schema rules and guidelines. Externally-developed schemas may be more difficult to manage due to inconsistency with Network guidelines for namespaces, hosting, and component versioning.

[6] Interim Network Steering Group, Network Blueprint Amendment, February 12, 2002, p. 16.

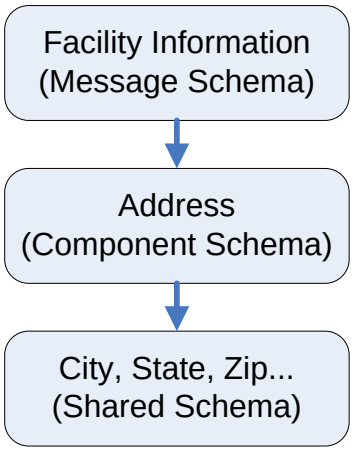
[7] Office of Management and Budget, Circular A-119: Federal Participation in the Development and Use of Voluntary Consensus Standards and in Conformity Assessment Activities, February 10, 1998.

Rules and Guidelines	
Rule	**Description**
[SD5-12]	Schemas developed outside of the Exchange Network MAY be used if they are consistent with the guidelines for schema development as set forth in Exchange Network rules and guidance.
Justification	
While it would be preferable to use only shared Exchange Network schemas, use of functional area schemas is necessary for the Exchange Network Schema Configuration Architecture. This approach allows functional areas to create their own schema constructs that are not available elsewhere. While duplication of schema constructs is a risk in allowing such flexibility, standard operating procedures can help mitigate this risk.	

2.6.5. Nested Includes

Constructs are shared by schemas in the same namespace through the use of inclusion. Inclusion allows one or more schemas to inherit the characteristics of a construct by referencing the schema that contains the construct directly ("single include") or indirectly ("nested includes"). The result is similar to cutting and pasting the construct into the calling schema, but it does not allow the construct to be changed or overridden. It also provides the added benefit of inheritance.

A nested include is illustrated in Figure 2.5-3. This example shows an example of a message schema referencing an address schema that in turn, references a shared schema that defines simple elements such as a city, state, and zip code.



When two or more schemas are nested into the message schema, the schema at the top of the nest is usually referenced to retrieve the characteristics of the construct in the message schema at the bottom of the nest. Note that as the number of schemas involved in nesting increases, so does the complexity involved in tracking and maintaining the schemas and their references.

Nested Includes

Pros and Cons	
Advantages	The structure for storing commonly used constructs is highly modular.
Disadvantages	It is difficult to keep track of and maintain constructs or schemas. The use of nested includes introduces potential conflict between XML construct names across the Exchange Network. Configuration does not enable the highest degree of interoperability and increases the complexity of Exchange Network schemas. Procedural guidance must be provided for proper implementation and use. Strict versioning and notification guidance are required.
Rules and Guidelines	
Rule	**Description**
[SD5-14]	Exchange Network schemas SHOULD group like constructs into one schema.
[SD5-15]	Message-level schemas SHOULD maintain a reasonable number of nested includes.
Justification	
As with any solution that offers a high degree of modularity, inclusion is highly subjective and requires common sense to avoid overly cumbersome implementations. Storage of similar shared complex or simpleType constructs in one shared schema is a more logical choice for developing schemas than accessing constructs through multiple layers of schemas.	

2.6.6. Documentation within Schema

This section discusses documentation within Exchange Network schemas. The following concepts are covered:

- *Schema construct documentation*—schema constructs within Exchange Network schemas
- *Schema header documentation*—schema headers within Exchange Network schemas.

Schema Construct Documentation

It is possible to enter HTML comments in schema. These comments will be ignored by the XML processor. In the following example, an HTML comment is shown

```
<!—This is a comment -->
```

This type of comment does not allow for machine processing of comments. Alternatively, the W3C documentation is very useful when documenting schemas. Following is an example of the documentation element:

```
<xsd:documentation>Facility Identification Schema</xsd:documentation>
```

Although the W3C Schema standard still supports HTML-style comments, use of the documentation element in a schema enables machine processing of comments because it is an XML element; therefore, comments can be processed by an application (for example, an XSL stylesheet) to create such documents as a user manual.

The documentation element can be used anywhere within a schema. In the following example, it is a sub-element of the W3C Schema annotation element and is used to document the function of a specific element:

```
<xsd:element name="FACID:FacilityIdentificationCode" type="xsd:string"/>
  <xsd:annotation>
    <xsd:documentation>Description goes here</xsd:documentation>
  <xsd:annotation>
</xsd:element>
```

Schema Construct Documentation

Pros and Cons	
Advantages	Schema construct documentation adds clarity to a schema and therefore enables efficient reuse of constructs. The W3C Schema documentation element enables machine processing of comments in a schema.
Disadvantages	Schema construct documentation can increase verbosity in a schema.
Rules and Guidelines	
Rule	**Description**
[SD5-28]	All schema SHOULD include schema construct documentation using the W3C documentation element. Documentation SHOULD only describe the element or datatype and SHOULD NOT contain lists of acceptable codes, implementation details or other information not directly related to the meaning of the construct.
[SD5-29]	Exchange Network schemas SHOULD use the documentation element for schema construct documentation.
[SD5-30]	HTML-style comments (<!--comment -->) SHOULD NOT be used in schema.
Justification	

HTML-style comments may be used to document the structure of a schema, as in the following example: <!—THESE ARE THE GLOBAL TYPES—!>. Such comments are not critical for machine processing because their meaning is irrelevant outside the schema.

Although the documentation element of schema constructs can increase verbosity, it adds clarity and enables efficient reuse of constructs by conveying their purpose. The advantages of documentation, therefore, outweigh the potential disadvantages.

Schema Header Documentation

Just as schema construct documentation adds clarity to a schema, schema header documentation enables information—the purpose, use, and contents of a schema—to be concentrated in a single place within a schema. As with schema construct documentation, the W3C Schema documentation element should be used for schema header documentation. Table 2.5-1 lists the items that should be included in the header section of all Exchange Network schemas.

Header Item Name	Description
Schema Name	The schema file name
Current Version Available At	The URL where the schema is located, if it is URL accessible
Description	Plain text description of the information described by the schema
Developed By	Agency/Company name that developed the schema. Both names should be included if a separate entity developed the schema on behalf of another entity.
Point of Contact	Person to contact with questions about the Schema

The header item name should precede the actual contents of the header item, as in the following example:

```
<xsd:annotation>
  <xsd:documentation>Schema Name : FACID_FacilityIndex_v3.0</xsd:documentation>
  <xsd:documentation>Current Version Available At : http://www.exchangenetwork.net/
    </xsd:documentation>
  <xsd:documentation>Description : This is a root element for the FacilityID exchange. This
    component contains a list of abbreviated facility summary information. This element is
    optimized for exchanging a list of high-level facility information based on an ad hoc user query.
    </xsd:documentation>
  <xsd:documentation>Developed by : US Environmental Protection Agency</xsd:documentation>
  <xsd:documentation>Point of Contact : Jane Smith</xsd:documentation>
</xsd:annotation>
```

Schema Header Documentation

Pros and Cons

Advantages	Schema header documentation concentrates the information about the purpose, use, and contents of a schema in a single place within a schema.
Disadvantages	There are no disadvantages to this technique.
Rules and Guidelines	
Rule	**Description**
[SD5-34]	Exchange Network schemas MUST include schema header documentation.
Justification	
Schema header documentation allows a schema developer to easily discern the purpose, use, and contents of a schema. This information is also very helpful when a schema developer needs to select a schema to be used as a template in the creation of another schema.	

2.7. Schema Versioning

This section describes the Exchange Network approach to schema versioning. Several components work together to define the versioning strategy, described in the following sub-sections:

- Major changes, minor changes and revisions,
- namespace versioning,
- schema file name versioning,
- The W3C Schema version attribute,
- The user-defined version attribute, and
- configuration of the Network schema repository

Each component is described in the following sections

2.7.1. Major Changes, Minor Changes, and Revisions

Changes to schema are defined as major changes, minor changes, and revisions. Each type of change is defined by different characteristics, described below.

Major changes to schema are those that are not backward compatible with previous versions of schema whereas minor changes are backward compatible. Backward compatibility exists if an XML instance document that validates against a previous minor version will continue to validate against the new minor version of the schema. Generally, minor changes may only add optional new components and/or reduce restrictiveness.

Specifically, in order for a schema change to be backward compatible, the following must be true:

- New elements and/or attributes may be added, but they must be optional in the new schema,
- existing required elements are made optional,
- allow additional occurrences of an element,
- add a "choice" compositor where an element existed previously, ensuring that one of the choice definitions is identical to the previous version element, or
- loosen restrictions such as expanding or eliminating maxLength values or adding more enumerations.

Revisions are allowed for schema while the schema is draft. Draft schemas must be denoted as such in the schema file name using the following naming convention:

TRI_ChemicalActivity_v1.2_**draft3**.xsd

The naming convention used in the example above is based on the conventions described in section **Error! Reference source not found.** that details the naming convention to be used for all exchange components.

It is recommended that developers put the word "draft" and a revision letter, number, or date in filename of the zip archive used to distribute schema packages. Developers are strongly encouraged to include a README file that contains the date, revision letter, and detailed change history. This allows others to easily differentiate between different schema versions.

Pros and Cons	
Advantages	The schema versioning strategy ensures consistency and predictability between schema versions The schema versioning strategy ensures that minor version changes are backward compatible with other minor versions of the same schema. The naming conventions for draft schema help reduce the likelihood that a draft schema will be misinterpreted as a production schema by an exchange implementer.
Disadvantages	Schema versioning introduces rigor that will require special attention on the part of a schema developer.
Rules and Guidelines	
Rule	**Description**
[SD5-E]	New minor versions of schema MUST be able to validate instance documents created with preceding minor versions of that schema. However, instance documents should not be expected to validate against versions of schema preceding the one they were created with.

[SD5-F]	New minor versions of a schema MUST only add new optional elements and/or attributes to prior minor versions.
[SD5-G]	New minor versions of a schema MUST NOT remove elements and/or attributes from prior minor versions.
[SD5-H]	New minor versions MUST utilize the exact same namespace as prior minor versions.
[SD5-I]	All existing instance documents in the same namespace MUST validate against the new minor version.
[SD5-J]	The schema major version MUST be incremented if any elements or attributes are removed or if new mandatory elements or attributes are added.
[SD5-K]	The schema file name, XSD version attribute, header documentation and namespace MUST all contain matching version information.
[SD5-L]	When any schema construct is altered in a given namespace, all schema in the namespace MUST undergo a version increment.
Justification	
The versioning strategy provides a consistent, structured approach to schema versioning, without which it would be very difficult to monitor or control the evolution of Network exchanges.	

2.7.2. Namespace Versioning

In January 2006, the Network Technology Group (NTG) released final guidance on the use of namespaces in Exchange Network Schema[8]. The guidelines presented in this section are based upon the recommendations in the final paper.

Namespaces in Network schema are URL formatted and are constructed using the following format:

```
http://www.exchangenetwork.net/schema/{ExchangeIdentifier}[/{module}]/{MajorVersion}
```

The Network namespace consists of the following components:

- *Namespace root* – the namespace root consists of the URL stub, matching the URL of the Network schema repository. The namespace root is required and cannot be changed.
- *Exchange Identifier* – a unique name or acronym used to identify the exchange. The identifier is used consistently across many exchange components.

[8] Namespace Organization, Naming, and Schema File Location v1.11 (March 6, 2006), http://www.exchangenetwork.net/dev_schema/Network_Namespace_v1.11.pdf

- *Module* – An optional subdivision of the exchange often used when an exchange has many schema or multiple categories of payloads representing different business areas that may be versioned separately.
- *Major Version* – the major version number for the exchange

The following example shows two, well-formed namespaces:

```
http://www.exchangenetwork.net/schema/AQS/1
http://www.exchangenetwork.net/schema/RCRA/Handler/3
```

Namespace Versioning

Pros and Cons	
Advantages	URL formatted namespaces are simple, and intuitive to internet users and developers. Namespaces will be unique to exchanges determined by the inherent structure of the repository The ability to resolve namespaces to a network location within the Network schema repository supports Network schema management and versioning goals. Standardized implementation of URL formatted namespaces supports future development initiatives based upon resolution of namespaces (e.g. RDDL[9])
Disadvantages	There are no disadvantages to this technique
Rules and Guidelines	
Rule	**Description**
[SD4-A]	The schema namespace name MUST be URL-formatted as “http://www.exchangenetwork.net/schema/”{ExchangeIdentifier}[“/”{Category}]”/”{Version}.
[SD4-B]	Exchange Network namespaces MUST contain the Exchange Identifier term that clearly and uniquely defines the type of data being exchanged.
[SD4-C]	Exchange Network namespaces MAY contain a module name which further divides the data exchange into smaller components.
[SD4-D]	Exchange Network namespaces MUST contain a major version number as the last part of the namespace name.
[SD4-E]	Exchange Network namespaces MUST NOT contain a minor version number in the namespace name.

[9] Resource Directory Description Language, http://rddl.org/

Justification
The Network namespace rules and guidelines were developed and approved by Exchange Network governance as part of the overarching versioning and repository architecture strategy.

2.7.3. Schema File Name

File naming conventions for Exchange Network schema are closely tied to the Schema Versioning strategy as well the approach for schema modularization as described in section 2.6.1.

The Network has developed specific guidelines for naming XML schema files.

Schema file names must be implemented consistently to ensure the Exchange Network registry and users can differentiate between schema versions. Rules in this section are closely related to the rules in the *Schema Versioning* section of this document.

Naming rules vary depending on the schema type, as described above. The following list provides examples of the proper naming convention for each schema type:

Schema Type	Convention	Examples
Default File	"index.xsd"	index.xsd
Message Schema	*ExchangeIdentifier* [+ "_" + *FlowCategoryName*] + "_" + *MessageName* + "_v" + *Version* + ".xsd".	NPDES_Permit_v1.1.xsd NPDES_Inspection_v1.1.xsd
Component Schema	*ExchangeIdentifier* [+ "_" + *FlowCategoryName*] + "_" + *ComponentName* + "_v" + *Version* + ".xsd".	NPDES_MonitoringPointDetails_v1.1.xsd
Local Shared Schema	*ExchangeIdentifier* [+ "_" + *FlowCategoryName*] + "_Shared" + "_v" + *Version* + ".xsd".	NPDES_Shared_v1.1.xsd

Each component of the name is defined as follows:

Component Name	Description/Usage
ExchangeIdentifier	A unique identifier for the data exchange (commonly an acronym).

ExchangeCategoryName	An optional naming component to be used only if the data exchange is subdivided into smaller parts. This naming component should only be used if the namespace contains an exchange category "subdirectory" in the namespace name (i.e. http://www.exchangenetwork.net/schema/RCRA/Handler/2).
MessageName	A term describing the data carried by the schema (i.e. inspection).
ComponentName	A term describing the structure contained within the schema.
Version	The major and minor version number of the schema file starting with "v" and separated by a period.

Ensure that case (capitalization) is used consistently when naming schema files and referencing schema using include and import statements. Some Web servers are case-sensitive and will not return a document if "index.XSD" is requested, but the actual file is named "index.xsd".

Schema File Names

Rules and Guidelines	
Rule	**Description**
[GD2-2]	File names MUST NOT use abbreviations unless their meaning is beyond question (EPA, GSA, FBI).
[GD2-A]	Each namespace must contain a default schema named "index.xsd" which contains only an include construct referencing the root schema.
[GD2-C]	Message schema MUST utilize the following naming format: {ExchangeIdentifier}["_"{ExchangeCategoryName}]"_"{MessageName} "_v"{Version}".xsd".
[GD2-D]	Component Message schema MUST utilize the following naming format: {ExchangeIdentifier}["_"{ExchangeCategoryName}]"_"{ComponentName} "_v"{Version}".xsd".
[GD2-E]	Local shared schema MUST utilize the following naming format: {ExchangeIdentifier}["_"{ExchangeCategoryName}]"_Shared_v"{Version}".xsd" .
[SD5-22]	Exchange Network schemas MUST include a major and minor version number in their filename.

2.7.4. W3C Schema Version Attribute

The W3C Schema standard contains a "version" attribute that can be included in the root element of a schema. Following is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:TRI="http://www.exchangenetwork.net/schema/TRI/4"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.exchangenetwork.net/schema/TRI/4" version="4.0">
```

W3C Schema Version Attribute

Pros and Cons	
Advantages	The W3C Schema version attribute takes advantage of a built-in feature of W3C Schema. Instance documents would not need to change if they remain valid with the new version of the schema. The schema contains information that informs applications that it has changed. An application could interrogate the version attribute, recognize that this is a new version of the schema, and take appropriate action.
Disadvantages	The W3C Schema version attributes are not inherently enforced by an XML validator.
Rules and Guidelines	
Rule	**Description**
[SD5-20]	Exchange Network schemas MUST include a version number using the W3C Schema version attribute.
[SD5-21]	The W3C Schema version attribute MUST include both a major version component and a minor version component.
Justification	
Version numbers must be included in schemas for configuration management. Use of a W3C Schema construct for the version number (as opposed to a user-defined schema construct) ensures that the version can be consistently located by an XSLT stylesheet or XML-capable processing application.	

2.7.5. User-defined Version Attribute on Message Schema Root Element

A user-defined version attribute on an element used in the instance root is useful for tracking minor updates. In the schema, it would look like this:

```
<xsd:complexType name="FacilityIndexDataType">
  <xsd:sequence>
    <xsd:element ref="facid:FacilitySummaryList" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="schemaVersion" type=”xsd:decimal” use="required"/>
</xsd:complexType>
```

In an XML instance document, the schemaVersion attribute would appear as follows:

```
<FACID:FacilityIndex schemaVersion="3.0">
  <!-- information removed for example purposes —>
</ FACID:FacilityIndex >
```

User-Defined Version Attribute

Pros and Cons	
Advantages	The schema version attribute is enforceable through schema validation. Instances would not validate without matching the constraints defined for the attribute. Instance documents may not need to change if they remain valid with the new schema version as long as the constraints were built to accommodate this. An application would receive an indication that the schema has changed because the instance would carry the version number.
Disadvantages	If the instance root element versioning is not implemented properly, any time a schema is updated, the instance documents based on that schema will need to be updated.
Rules and Guidelines	
Rule	**Description**
[SD5-26]	Exchange Network schemas MUST define a required attribute named "schemaVersion" in the root element of all message schema.
Justification	
Version numbers must be included in schemas for configuration management. Use of a W3C Schema construct for the version number (as opposed to a user-defined schema construct) ensures that the version can be consistently located by an XSLT stylesheet or XML-capable processing application.	

2.8. Information Association and Uniqueness

Information association allows XML processors or processing applications to link information items. Uniqueness involves the non-duplication of certain information within either an entire XML instance document or a section of an XML instance document. This chapter discusses techniques that can be used for information association and uniqueness within XML instance documents, and it provides guidance for the Exchange Network.

2.8.1. Information Association

For processing, it is sometimes necessary to associate information in an XML instance document. For example, consider an XML instance document with information about purchase orders and customers. It may be beneficial for a processing application that stores this information in a relational database to associate the purchase orders in the

XML instance document with their corresponding customers so that these associations can be recorded in the database.

Several techniques can be used to make information associations in XML instance documents:

- ID/IDREF technique, which uses the W3C Schema ID and IDREF built-in datatypes to associate information in an XML instance document
- KEY/KEYREF technique, which uses W3C Schema KEY and KEYREF constructs to associate information in an XML instance document
- XLink/XPointer technique, whichuses two relatively new W3C standards to associate information in an XML instance document through the addition of declarations to an XML instance document.

These techniques are discussed in the following subsections.

2.8.2. ID/IDREF Technique

With the ID/IDREF technique, information that must be included in an association is contained in an attribute of datatype ID or IDREF. In the following example, submitters are associated with their facilities through the inclusion of a facilityIdentificationCode attribute for each submitter and facility:

```
<FacilityList>
  <FacilitySiteDetails facilityIdentificationCode="A15849">
    <FacilityAddressDetails>
      <!—information removed for example purposes—>
    </FacilityAddressDetails>
  </FacilitySiteDetails>
</FacilityList>

<PermitList>
  <PermitDetails permitFacilityIdentificationCode="A15849">
      <!—information removed for example purposes—>
   </PermitDetails>
</PermitList>
```

In the above example, the facilityIdentificationCode attribute is of datatype "ID," while the permitFacilityIdentificationCode attribute is of datatype "IDREF." An XML processor will validate that there is a corresponding ID-type attribute value in an XML instance document for each IDREF-type attribute value—there is a corresponding facilityIdentificationCode attribute value for each permitFacilityIdentificationCode attribute value. Therefore, in the above example, if there were no ID-type attribute in the XML instance document with a value of A15849, an XML processor would generate an error.

It should be noted that an XML processor cannot confirm that matching ID/IDREF values perform the intended associations. The ID/IDREF construct is akin to an anchor

tag in HTML. It is only intended to associate an IDREF in one portion of the document with an ID in some other portion of the document, but the location of the ID is not enforced. In the example above, if the ID "A15849" appeared on any other element, the document would validate.

ID/IDREF Technique

Pros and Cons	
Advantages	With the ID/IDREF technique, an XML processor will validate a corresponding ID-type attribute value in an XML instance document for each IDREF-type attribute value.
Disadvantages	With the ID/IDREF technique, an XML processor cannot confirm that matching ID/IDREF values perform the intended associations. The technique uses attributes, which are prohibited for Exchange Network schemas. An ID- or IDREF-type attribute value must be unique in an XML instance document. An ID- or IDREF-type attribute value cannot begin with a number.
Rules and Guidelines	
Rule	**Description**
[SD6-1]	Exchange Network schemas MUST NOT use the ID/IDREF technique for information association.
Justification	
The many disadvantages of this technique render it virtually unusable. The ID/IDREF technique has been superseded within the W3C Schema standard by the new KEY/KEYREF technique, which is discussed below.	

2.8.3. KEY/KEYREF Technique

The KEY/KEYREF technique has the following improvements over the ID/IDREF technique:

- Use of attributes is not required (elements can be used for associations).
- Constructs used in associations can be of any datatype (they do not have to be ID-type or IDREF-type attributes).
- An XML processor will confirm that matching values perform the intended associations.
- Values for constructs used in associations can be duplicated in XML instance documents because the KEY/KEYREF technique allows the specification of a range within an XML instance document for which the values must be unique.

The KEY/KEYREF technique uses separate W3C constructs to associate information in an XML instance document. The first construct—the KEY construct—declares a "primary key" for an information association, while the second construct—the KEYREF construct—declares a "foreign key."

Continuing with the above example, the following KEY construct declares a primary key for the facilityIdentificationCode attribute in an XML instance document:

```
<xsd:keyref name="SubmitterKey" refer="FacilityKey">
  <xsd:selector xpath="Submitters/SubmitterDetails"/>
  <xsd:field xpath="@submitterFacilityIdentificationCode"/>
</xsd:keyref>
```

As with the KEY declaration, the above KEYREF declaration stipulates that the value of the permitFacilityIdentificationCode attribute must be unique within all PermitDetails elements appearing under the Submitters element in the XML instance document. In addition, the refer attribute ensures that an XML processor will confirm there is a corresponding FacilityIdentificationCode value for each permitFacilityIdentificationCode value in an XML instance document.

KEY/KEYREF Technique

Pros and Cons	
Advantages	Use of attributes is not required (unlike ID/IDREF). Constructs used in associations can be of any datatype. An XML processor will confirm that matching values perform the intended associations. Values for constructs used in associations can be duplicated in XML instance documents.
Disadvantages	KEY and KEYREF declaration names must be unique within a schema and across externally referenced schemas, regardless of namespace.
Rules and Guidelines	
Rule	**Description**
[SD6-3]	Exchange Network schemas SHOULD use the KEY/KEYREF technique for information association.
[SD6-4]	Extreme caution SHOULD be applied when writing an XPath expression in a selector element to ensure it specifies the intended range.
[SD6-5]	Special attention SHOULD be paid to the restrictions on KEY and KEYREF declaration names given above.
Justification	

The vast improvements of the KEY/KEYREF technique over the ID/IDREF technique make it very valuable. The fact that this technique does not require the use of attributes also makes it beneficial, because Exchange Network schemas prohibit attributes.

2.8.4. KEY Technique

A KEY declaration can be used without a corresponding KEYREF declaration to enforce uniqueness. With the KEY technique, the construct that enforces uniqueness must appear in an XML instance document. Continuing with the above example, the following KEY construct enforces uniqueness for the facilityIdentificationCode attribute within a specified range in an XML instance document:

```
<xsd:key name="FacilityKey">
  <xsd:selector xpath="FacilityList/FacilitySiteDetails"/>
  <xsd:field xpath="@facilityIdentificationCode"/>
</xsd:key>
```

It is required, however, that the facilityIdentificationCode attribute always appear within the specified range.

KEY Technique

Pros and Cons	
Advantages	The KEY technique enforces uniqueness of values within a specified range in an XML instance document, while requiring their constructs to appear within that range.
Disadvantages	KEY declaration names must be unique within a schema and across externally referenced schemas, regardless of namespace. An XML processor may not detect an incorrect XPath expression in a KEY declaration. This can cause a duplication of a value in an XML instance document to be undetected.
Rules and Guidelines	
Rule	**Description**
[SD6-9]	Exchange Network schemas SHOULD use the KEY technique to enforce uniqueness of values in an XML instance document when their constructs are required to appear within the specified range.
[SD6-10]	Extreme caution SHOULD be applied when writing an XPath expression in a selector element to ensure it specifies the intended range.
[SD6-11]	Special attention SHOULD be paid to the restrictions on KEY declaration names given above.
Justification	

The KEY technique is extremely useful for enforcing uniqueness in an XML instance document, and its use is therefore recommended.

2.8.5. XLink/XPointer Technique

The XLink/XPointer technique is used to create hyperlinks in XML documents. With this technique the declarations for the information association are placed in an XML instance document rather than in a schema. Two types of links are discussed:

- Simple links, which declare associations between two items within a single XML instance document
- Extended links, which declare associations between any number of items both within and across XML instance documents.

Simple Links

Simple links are unidirectional links, much like the HTML "A" element. In the following example, submitters are associated with their facilities through the inclusion of a simple link within each PermitDetails element:

```
<FacilityList>
  <FacilitySiteDetails facilityIdentificationCode="A15849">
    <FacilityAddressDetails>
      <!—information removed for example purposes—>
    </FacilityAddressDetails>
  </FacilitySiteDetails>
</FacilityList>

<PermitList>
  <PermitDetails>
    <PermitFacilityIdentificationCode xlink:type="simple" xlink:href="#A15849"/>
    <!—information removed for example purposes—>
  </PermitDetails>
</PermitList>
```

The notation used in the xlink:href construct above ("#" followed by the actual value) is called an XPointer Bare Names notation. The following should be noted regarding this technique:

- Information that requires inclusion in an association must be contained in an attribute of datatype ID.
- An XLink-aware processor is not required to confirm there is a corresponding ID-type attribute value in an XML instance document for each href attribute value.
- An XLink-aware processor is not required to confirm that matching values perform the intended associations in an XML instance document.

Extended Links

Unlike simple links, extended links do not need to be declared in the same XML instance document that contains the items being associated. Therefore, they are useful when associations are required between items in one or more XML instance documents, but the XML instance documents cannot be updated to indicate the associations. Also unlike simple links, extended links can declare associations between more than two items.

Continuing with the previous example, the following extended link declaration associates permits with their facilities:

```
<xlink:extended role="Link Permits to Facilities" title="Links">
  <xlink:locator href="#9187234"role="Permit" label="Permit 9187234">
  <xlink:locator href="#147341" role="Permit" label="Permit 147341">
  <xlink:locator href="#A15849" role="Facility" label="Facility A15849">
  <xlink:arc from="Permit 9187234" to="Facility A54346" arcrole="Permit Issued To">
  <xlink:arc from="Permit 147341" to="Facility A54346" arcrole="Permit Issued To">
</xlink:extended>
```

The locator elements in the above example specify the elements that participate in the extended link. There is one locator element for each submitter identification code and facility identification code. The role attributes simply describe the function of the location where they appear. The arc elements specify the actual associations between the submitters and facilities using each submitter and facility's label attribute.

The following similarities with simple links should be noted:

- Information that requires inclusion in an association must be contained in an attribute of datatype ID.
- An XLink-aware processor is not required to verify that there is a corresponding ID-type attribute value in an XML instance document for each href attribute value.
- An XLink-aware processor is not required to confirm that matching values perform the intended associations in an XML instance document.

XLink/XPointer Technique

Pros and Cons	
Advantages	The XLink/XPointer technique allows declarations for information association to be placed in an XML instance document rather than a schema. This can be useful when a schema cannot be updated. Extended links allow declarations for information association to be placed in a separate XML instance document. This can be useful when an XML instance document cannot be updated. Extended links allow associations to be declared between more than two items.

Disadvantages	With the XLink/XPointer technique, information that must be included in an association must be contained in an attribute of datatype ID. An XLink-aware processor is not required to confirm that there is a corresponding ID-type attribute value in an XML instance document for each href attribute value in a simple or extended link. An XLink-aware processor is not required to confirm that matching values perform the intended associations in an XML instance document.
Rules and Guidelines	
Rule	**Description**
[SD6-15]	Exchange Network schemas MUST NOT use the XLink/XPointer technique for information association.
Justification	
The XLink and XPointer standards are not yet mature; therefore, there is very little XML processor support for them. In addition, the XLink/XPointer technique requires the use of ID-type attributes, which have multiple disadvantages.	

2.8.6. UNIQUE Technique

For processing, it is sometimes necessary to ensure that information is not duplicated in an XML instance document. Two techniques can be used to enforce uniqueness in XML instance documents:

- KEY technique, which, as stated earlier in this chapter, uses the W3C Schema KEY construct to specify a range within an XML instance document for which values must be unique, while requiring their constructs to appear within that range
- UNIQUE technique, which uses the W3C Schema UNIQUE construct to specify a range within an XML instance document for which values must be unique, without requiring their constructs to appear within that range.

With the UNIQUE technique, the construct for which uniqueness is enforced does not need to appear in an XML instance document; however, if it does appear, its value must be unique within a specified range within the XML instance document. This range is specified using a technique that is similar to the KEY technique.

Continuing with the above example, the following UNIQUE construct enforces uniqueness for the facilityIdentificationCode attribute within a specified range in an XML instance document:

```
<xsd:unique name="FacilityKey">
  <xsd:selector xpath="FacilityList/FacilitySiteDetails"/>
  <xsd:field xpath="@facilityIdentificationCode"/>
```

```
</xsd:key>
```

It is not required, however, that the facilityIdentificationCode attribute always appear within the specified range.

Unique Technique

Pros and Cons	
Advantages	The UNIQUE technique enforces uniqueness of values within a specified range in an XML instance document without requiring their constructs to appear within that range.
Disadvantages	UNIQUE declaration names must be unique within a schema and across externally referenced schemas, regardless of namespace. An XML processor may not detect an incorrect XPath expression in a UNIQUE declaration. This can cause a duplication of a value in an XML instance document to be undetected.
Rules and Guidelines	
Rule	**Description**
[SD6-17]	Exchange Network schemas SHOULD use the UNIQUE technique to enforce uniqueness of values in an XML instance document when their constructs are not required to appear within the specified range.
[SD6-18]	Extreme caution SHOULD be applied when writing an XPath expression in a selector element to ensure it specifies the intended range.
[SD6-19]	Special attention SHOULD be paid to the restrictions on UNIQUE declaration names given above.
Justification	
The XLink and XPointer standards are not yet mature; therefore, there is very little XML processor support for them. In addition, the XLink/XPointer technique requires the use of ID-type attributes, which have multiple disadvantages.	

2.9. Advanced W3C Concepts

This chapter discusses the following W3C Schema advanced concepts and provides recommendations for their use on the Exchange Network:

- *Datatype derivation*—derivation of new datatypes from existing datatypes in a schema.
- *Variable content models*—schema constructs that allow the structure of information within an XML instance document to vary greatly without requiring schema updates.

- *Default and fixed element and attribute values*—the new W3C Schema features of default and fixed element values, as well as default and fixed attribute values.
- *Nillible Attribute*—A built-in XSD attribute that allows an element to be empty.
- *Substitution groups*—a W3C Schema feature that allows an element to replace another element in an XML instance document without requiring schema updates.
- *Supplemental instructions*—the use of supplemental instructions in a schema to pass them to a processing application.

2.9.1. Datatype Derivation

Datatype derivation can be applied to both simple and complex datatypes. Each is discussed below.

Simple Datatype Derivation

Simple datatypes can be derived using the following techniques:

- Simple datatype restriction, in which the properties of a simple datatype are used for the basis of a new simple datatype and further restricted
- *List technique,* in which a space-separated list of values is created from a base datatype
- *Union technique,* in which a range of possible values for a simple datatype is restricted through the union of two or more simple datatypes.

The three techniques for deriving simple datatypes are discussed in the following subsections. In the discussion, two concepts are key:

- A base datatype is the existing datatype that serves as the basis for a derived datatype.
- A facet is a W3C Schema construct that specifies various datatype properties. The following are examples of W3C Schema facets:
    - minInclusive—the minimum permissible value for a range
    - maxInclusive—the maximum permissible value for a range
    - minLength—the minimum permissible length for a datatype
    - maxLength—the maximum permissible length for a datatype
    - enumeration—a set of allowed values for a datatype.

Simple Datatype Restriction

With simple datatype restriction, a base datatype is restricted to a range of allowed values using facets. The base datatype may be a W3C Schema built-in datatype or a user-defined

datatype. In the following example, a derived simple datatype is defined to restrict the base datatype "integer" (a W3C Schema built-in datatype) to a range (1–10) of allowed values:

```
<xsd:element name="WaterQualityRatingCode"type="xsd:RangeOneToTenDataType"/>
<xsd:simpleType name="RangeOneToTenType">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1">
    <xsd:maxInclusive value="10">
  </xsd:restriction>
</xsd:simpleType>
```

This derived datatype can also be used as a base datatype for other restrictions. In the following example, a new datatype is defined based on the above datatype, but for the range of 3–10:

```
<xsd:simpleType name="RangeThreeToTenDataType">
  <xsd:restriction base="xsd: RangeOneToTenDataType">
    <xsd:minInclusive value="3">
  </xsd:restriction>
</xsd:simpleType>
```

Only the "minInclusive" facet was required above because the "maxInclusive" value of 10 carried from the base datatype.

As with complex datatypes, derived simple datatypes can be named and, therefore, globally declared.

Simple Datatype Restriction

Pros and Cons	
Advantages	Simple datatype restriction allows global simple datatypes to be created. Global simple datatypes can be associated with any element in a schema. Simple datatype restriction also allows the use of existing simple datatypes to define new datatypes, thereby decreasing complexity in a schema. A change to a user-defined datatype that is used as a base datatype for other simple datatypes will propagate to those datatypes. This allows a far-reaching change to be made in a single location within a schema, thereby lowering maintenance costs.
Disadvantages	A change to a user-defined datatype that is used as a base datatype for other simple datatypes will propagate to those datatypes. Additional schema updates may be required in such cases, thereby increasing maintenance costs. Restriction makes schema more difficult to read since a reader must trace back to a one or more predecessor types to fully comprehend the structure of the derived type.
Rules and Guidelines	

Rule	Description
[SD7-1]	Exchange Network schemas SHOULD NOT use simple datatype restriction when a data standard or an approved schema exists.
[SD7-2]	Exchange Network schemas MUST use global simple datatypes.
[SD7-3]	Exchange Network schemas MUST NOT use local simple datatypes.
[SD7-A]	Exchange Network schemas SHOULD define facets such as string lengths, patterns, and numeric ranges when the schema targets a specific data system.
Justification	
Simple datatype restriction is a very useful W3C Schema feature. Global simple datatypes are valuable because they can be associated with any element in a schema. This promotes high datatype visibility. Although there is a potential requirement for additional schema updates in a propagation scenario, the potential advantages of using simple datatype restriction far outweigh the potential disadvantages.	

List Technique

With the list technique, a datatype for a whitespace-delimited list of values is defined from a base datatype. In the following example, a StateCodeListDataType is defined based on a custom StringMin2Max2DataType datatype:

```
<xsd:element name="StateCodeList" type="StateCodeListDataType"/>

<xsd:simpleType name="StateCodeListDataType">
  <xsd:list itemType="StringMin2Max2DataType"/>
</xsd:simpleType>

<xsd:simpleType name="StringMin2Max2DataType">
  <xsd:restriction base="xsd:string">
    <xsd:minLength value="2"/>
    <xsd:maxLength value="2"/>
  </xsd:restriction>
</xsd:simpleType>
```

The following is an XML instance document excerpt that uses the above declaration:

```
<StateCodeList>MI OR WA LA</StateCodeList>
```

In the example above, an XML parser would require that state codes were two characters long, but would not ensure that the state code was valid.

List Technique

Pros and Cons

Advantages	The list technique allows datatypes to be defined to represent a whitespace-delimited list of values.
Disadvantages	The list technique is not appropriate for values that may contain whitespace The list technique does not allow for complex structures. If a list item needs to be expanded to hold additional data, the list technique cannot be used. There is no way to represent nil values using the list technique. Individual list values cannot be accessed using XPath or XSLT.
Rules and Guidelines	
Rule	**Description**
[SD7-7]	Exchange Network schemas MAY use the list technique.
[SD7-8]	Exchange Network schemas MUST NOT use the list technique if the values within the list may contain spaces themselves(e.g., a person's first and last name).
Justification	
The usefulness of this technique is limited because of the disadvantages stated above, but it is a convenient way to represent lists of values in a concise manner.	

Union Technique

With the union technique, a datatype that represents a range of allowed values is defined through the union of two or more existing datatypes. In the following example, a simple datatype, StateOrRegionType, is defined as the union of two list datatypes: one that holds state codes (StateCodesType), and another that holds region codes (RegionCodesType). The StateOrRegion element can therefore contain any value that is a state or a region code:

```
<xsd:element name="StateOrRegion" type= "StateOrRegionType"/>

<xsd:simpleType name="StateOrRegionType">
  <xsd:union memberTypes="StateListType RegionListType"/>
</xsd:simpleType>

<xsd:simpleType name="StateListType">
  <xsd:list itemType="StateCodesType"/>
</xsd:simpleType>

<xsd:simpleType name="RegionListType">
  <xsd:list itemType="RegionCodesType"/>
</xsd:simpleType>

<xsd:simpleType name="StateCodesType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="AK"/>
    <xsd:enumeration value="AL"/>
    <xsd:enumeration value="AR"/>
    <!—information removed for example purposes—>\
  </xsd:restriction>
```

```
</xsd:simpleType>

<xsd:simpleType name="RegionCodesType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="01"/>
    <xsd:enumeration value="02"/>
    <xsd:enumeration value="03"/>
    <!—information removed for example purposes—>
  </xsd:restriction>
</xsd:simpleType>
```

Therefore, either of the following two elements would be valid in an XML instance document:

```
<StateOrRegion>VA</StateOrRegion>
<StateOrRegion>03</StateOrRegion>
```

Union Technique

Pros and Cons	
Advantages	The union technique allows an element to contain a range of values that is merged from two or more datatypes. A change to any user-defined datatype used in a union will propagate to the union datatype. This allows a far-reaching change to be made in a single location within a schema, thereby lowering maintenance costs.
Disadvantages	A change to any user-defined datatype used in a union will propagate to the union datatype. Additional schema updates may be required in such cases, thereby increasing maintenance costs. Use of the union technique may add an additional level of complexity to a schema.
Rules and Guidelines	
Rule	**Description**
[SD7-11]	Exchange Network schemas MAY use the union technique.
Justification	
The union technique is a very useful W3C Schema feature. Although there is a potential for additional schema updates to be required in a propagation scenario, the potential advantages for using the union technique far outweigh the potential disadvantages.	

Complex Datatype Derivation

Complex datatypes can be derived using the following techniques:

- Complex datatype restriction, in which the definition of a complex datatype serves as the basis of a new complex datatype, which is further restricted through the removal or modification of constructs

- Complex datatype extension, in which the definition of a complex datatype serves as the basis of a new complex datatype, which is further expanded through the addition of constructs.

Complex Datatype Restriction

With complex datatype restriction, a base datatype is restricted through the removal or modification of constructs. In the following example, the SSC v2.0 PermitIdentityDataType will be restricted to create a new derived type named NPDESPermitIdentity. The area below displays the PermitIdentityDataType as defined in the SSC v2.0:

```
<xsd:complexType name="PermitIdentityDataType">
  <xsd:sequence>
    <xsd:element ref="PermitIdentifier" minOccurs="0"/>
    <xsd:element ref="PermitName" minOccurs="0"/>
    <xsd:element ref="OtherPermitIdentifier" minOccurs="0"/>
    <xsd:element ref="ProgramName" minOccurs="0"/>
    <xsd:element ref="PermitType" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

The area below displays the new definition for the derived NPDESPermitIdentityDataType. Note that all constructs from the base datatype must be listed in the restriction:

```
<xsd:include schemaLocation="SC_IndividualIdentity_v2.0.xsd"/>

<xsd:complexType name="NPDESPermitIdentityDataType">
  <xsd:complexContent>
    <xsd:restriction base="PermitIdentityDataType">
      <xsd:sequence>
        <xsd:element ref="PermitIdentifier" minOccurs="0" />
        <xsd:element ref="PermitName" minOccurs="0"/>
        <xsd:element ref="OtherPermitIdentifier" minOccurs="0"/>
        <xsd:element ref="ProgramName" minOccurs="0"/>
        <xsd:element ref="PermitType" minOccurs="0"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

By removing the minOccurs="0" from the PermitIdentifier element, it will now be required in the XML instance document. (Recall that the default occurrence for an element is minOccurs="1" and maxOccurs="1" if not specified explicitly). Also, since the OtherPermitIdentifier element has been removed, it may not appear in an XML instance document for a NPDESPermitIdentity element. Note that elements can only be removed when they are declared as optional in the base datatype.

Complex Datatype Restriction

Pros and Cons

Advantages	Complex datatype restriction allows new complex datatypes to be defined based on existing complex datatypes, thereby decreasing complexity in a schema. A change to a complex datatype that is used as a base datatype for other complex datatypes will propagate to those datatypes. This allows a far-reaching change to be made in a single location in a schema, thereby lowering maintenance costs.
Disadvantages	A change to a complex datatype that is used as a base datatype for other complex datatypes will propagate to those datatypes. Additional schema updates may be required in such cases, thereby increasing maintenance costs.
Rules and Guidelines	
Rule	**Description**
[SD7-13]	Exchange Network schemas MAY use complex datatype restriction.
Justification	
Complex datatype restriction is a very useful W3C Schema feature. Although there is a potential requirement for additional schema updates in a propagation scenario, the potential advantages for using complex datatype restriction far outweigh the potential disadvantages.	

Complex Datatype Extension

With complex datatype extension, a base datatype is extended through the addition of constructs. In the following example, the SSC v2.0 PermitIdentityDataType is extended to include an additional element named ReissuanceNumber:

```
<xsd:complexType name="NPDESPermitIdentityExtendedDataType">
  <xsd:complexContent>
    <xsd:extension base="PermitIdentityDataType">
      <xsd:sequence>
        <xsd:element ref="ReissuanceNumber"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Elements that are added through extension are appended to the end of the sequence defined in the base type.

Any element in an XML instance file that is expected to conform to the base type may be substituted with one of the derived types. When a derived type is substituted for a base type in an instance document the xsi:type W3C attribute must be used to indicate to the XML processor which datatype (base or derived) applies:

```
<PermitIdentity xsi:type="NPDESPermitIdentityDataType">
  <PermitIdentifier>WA1234567</PermitIdentifier>
```

```
  <!—information removed for example purposes—>
</PermitIdentity>
```

Complex Datatype Extension

Pros and Cons	
Advantages	Complex datatype extension allows the definition of new complex datatypes based on existing complex datatypes, thereby decreasing complexity in a schema. A change to a complex datatype that serves as a base datatype for other complex datatypes will propagate to those datatypes. This allows a far-reaching change to be made in a single location in a schema, thereby lowering maintenance costs.
Disadvantages	A change to a complex datatype that serves as a base datatype for other complex datatypes will propagate to those datatypes. Additional schema updates may be required in such cases, thereby increasing maintenance costs.`
Rules and Guidelines	
Rule	**Description**
[SD7-15]	Exchange Network schemas MAY use complex datatype extension.
Justification	
Complex datatype extension is a very useful W3C Schema feature. Although there is a potential requirement for additional schema updates in a propagation scenario, the potential advantages for using complex datatype extension far outweigh the potential disadvantages.	

Prohibiting Complex Datatype Derivation

It is possible to indicate in a schema that a given complex datatype cannot be restricted or extended. This may be useful when a schema developer believes a datatype definition is final and, therefore, should not be restricted or extended by other schema developers—or when the base datatype of a derived datatype may change in the future. This is accomplished by a final attribute, which is placed on the complex datatype definition as follows:

```
<xsd:complexType name=“FacilityAddressDetailsType” final=“restriction”>
  <xsd:sequence>
    <!—information removed for example purposes—>
  </xsd:sequence>
</xsd:complexType>
```

The value of the final attribute shown above can also be “extension” (to prohibit complex datatype extension) or “all” (to prohibit both restriction and extension).

Prohibiting Complex Datatype Derivation

Pros and Cons	
Advantages	The ability to prohibit complex datatype derivation can be useful when a schema developer believes a datatype definition is final and, therefore, should not be restricted or extended by other schema developers, or when it is anticipated that the base datatype of a derived datatype may change in the future.
Disadvantages	Prohibiting complex datatype derivation may lead to the unnecessary creation of new complex datatypes, causing unnecessary duplication of schema constructs.
Rules and Guidelines	
Rule	**Description**
[SD7-17]	Exchange Network schemas MAY use the final attribute derivation.
Justification	
The potential for duplication of schema constructs outweighs the potential advantages for prohibiting complex datatype derivation. Therefore, use of the "final" attribute is not recommended for Exchange Network schemas. Because duplication of schema constructs is not as critical an issue for document-centric schemas as for Exchange Network schemas, prohibiting complex datatype derivation is permissible for document-centric schemas.	

Prohibiting Use of Derived Complex Datatypes

In a schema, it is possible to prohibit the appearance of an element that is a derivation of a given complex type. This may be useful when a schema has been updated to include derived datatypes, but a processing application cannot yet accommodate the change. This prohibition is accomplished by a block attribute placed on the complex datatype definition, as follows:

```
<xsd:complexType name="FacilityAddressDetailsType" block="restriction">
  <xsd:sequence>
    <!—information removed for example purposes—>
  </xsd:sequence>
</xsd:complexType>
```

As with the final attribute, the value of the block attribute shown above can also be "extension" or "all".

Prohibiting Use of Derived Complex Types

Pros and Cons

Advantages	The ability to prohibit use of derived complex datatypes can be useful when a schema has been updated to include derived datatypes but a processing application cannot yet accommodate the change.
Disadvantages	Prohibiting the use of derived complex datatypes may cause an increase in errors from the processing of XML instance documents.
Rules and Guidelines	
Rule	**Description**
[SD7-19]	Exchange Network schemas MAY use the block attribute.
Justification	
The potential for XML instance document errors outweighs the potential advantages of prohibiting use of derived complex datatypes; therefore, prohibiting use of derived complex datatypes is not recommended.	

2.9.2. Variable Content Models

Variable content models allow the structure of information within an XML instance document to vary greatly without requiring schema updates. This section discusses two W3C Schema techniques—*abstract datatypes* and *wildcards*—that enable variable content models in XML instance documents.

Abstract Datatypes

Abstract datatypes are complex datatypes that act as templates for the derivation of other complex datatypes. Unlike base datatypes, abstract datatypes cannot be used in the declaration of elements. Instead, a derived datatype must be defined based on the abstract datatype; only then can this derived datatype be used in the declaration of elements.

Abstract datatypes are useful when representing the fewest constructs required for a series of complex datatypes, which allows a certain level of consistency in all element declarations indirectly based on the abstract datatype.

In the following example, the FacilityAddressDetailsType datatype is declared as an abstract datatype. It contains the fewest constructs required for a facility address:

```
<xsd:complexType name="FacilityAddressDetailsTypeTemplate" abstract="true">
  <xsd:sequence>
    <xsd:element name="FacilityStreetAddress" type="xsd:string"/>
    <xsd:element name="FacilityCity" type="xsd:string"/>
    <xsd:element name="FacilityState" type="xsd:string"/>
    <xsd:element name="FacilityZipCode" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

The following complex datatype is derived from the above abstract datatype and includes additional constructs:

```
<xsd:complexType name="FacilityAddressDetailsType">
  <xsd:complexContent>
    <xsd:extension base="FacilityAddressDetailsTypeTemplate">
      <xsd:sequence>
        <xsd:element name="FacilityRegion" type="xsd:string"/>
        <xsd:element name="FacilityStreetAddressExtra" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

If constructs were added to the "FacilityAddressDetailsTypeTemplate" abstract datatype, the change would propagate to all element declarations that are indirectly based on the abstract datatype (all facility address constructs).

Abstract Datatypes

Pros and Cons	
Advantages	Abstract datatypes can be useful in representing the minimum amount of constructs required for a series of complex datatypes, allowing a certain level of consistency in all element declarations that are indirectly based on the abstract datatype. A change to a complex datatype that is defined as an abstract datatype will propagate to all datatypes based on the abstract datatype. This allows a far-reaching change to be made in a single location in a schema, thereby lowering maintenance costs.
Disadvantages	A change to a complex datatype that is defined as an abstract datatype will propagate to all datatypes based on the abstract datatype. Additional schema updates may be required in such cases, thereby increasing maintenance costs. Use of abstract datatypes may add an additional level of complexity to a schema.
Rules and Guidelines	
Rule	**Description**
[SD7-21]	Exchange Network schemas MUST NOT use abstract datatypes.
Justification	
Although abstract datatypes can be useful in some situations, the additional level of complexity that they add to a schema does not outweigh their potential advantages.	

Wildcards

Wildcards are W3C Schema features used to create a placeholder where any well-formed XML can appear in an XML instance document. This may be useful when a project is beginning and it is unclear what content will be required within an XML instance document or where a portion of an XML instance document should be unconstrained (perhaps because it is a memo or notes field).

The W3C Schema any and anyAttribute constructs mark the occurrence of a wildcard in a schema, as in the following example:

```
<xsd:element name="FacilityAddressDetails"type="FacilityAddressDetailsType"/>
  <xsd:complexType name="FacilityAddressDetailsType">
    <xsd:sequence>
      <xsd:element name="FacilityStreetAddress" type="xsd:string"/>
      <xsd:element name="FacilityCity" type="xsd:string"/>
      <xsd:element name="FacilityState" type="xsd:string"/>
      <xsd:element name="FacilityZipCode" type="xsd:string"/>
      <xsd:any minOccurs="0"/>
    </xsd:sequence>
  <xsd:anyAttribute/>
</xsd:complexType>
```

In the above example, the FacilityZipCode element may be followed by one or more elements. Although the schema developer's intention may be to show that elements appearing at that point in an XML instance document relate to a facility address, there is no such requirement. In addition, the FacilityAddressDetails element may contain an attribute that may or may not relate to a facility address.

A certain degree of control can be placed on wildcards by controlling the namespace from which constructs that are used in place of a wildcard declaration originate. This is accomplished by a namespace attribute that is placed on the any element. The following are possible values:

- ##any—constructs can come from any namespace.
- ##targetNamespace—constructs must come from the same namespace as the target namespace of the schema.
- ##other—constructs must come from a namespace other than the target namespace of the schema.
- ##local—constructs must not be in a namespace.
- A namespace URI—constructs must come from the specified namespace.

In addition, a schema developer can specify how an XML processor should validate constructs used in place of a wildcard declaration. This is accomplished by a

processContents attribute placed on the any element. The following are possible processContents values:

- skip—the XML processor does not attempt to validate the contents.
- strict—the XML processor validates the contents according to the schema of the specified namespace; an error is generated if the schema cannot be accessed.
- lax—the XML processor validates whatever contents it can according to the schema of the specified namespace; an error is not generated if the schema cannot be accessed.

Wildcards declarations can lead to "nondeterministic" content models, which can cause an XML processor to generate an error. Consider the following variation on the above example (the wildcard declaration is now at the top of the content model):

```
<xsd:complexType name="FacilityAddressDetailsType">
  <xsd:sequence>
    <xsd:any minOccurs="0" maxOccurs="2"/>
    <xsd:element name="FacilityStreetAddress" type="xsd:string"/>
    <xsd:element name="FacilityCity" type="xsd:string"/>
    <xsd:element name="FacilityState" type="xsd:string"/>
    <xsd:element name="FacilityZipCode" type="xsd:string"/>
  </xsd:sequence>
  <xsd:anyAttribute/>
</xsd:complexType>
```

If a FacilityStreetAddress element appeared in place of the wildcard declaration in the above example, it would not be possible for an XML processor to discern

- if the FacilityStreetAddress element were used in place of the wildcard declaration, or
- if it represents the FacilityStreetAddress element declared above.

In fact, because the any element has a maxOccurs value of "2," three FacilityStreetAddress elements could appear in succession (two in place of the any element, and the one representing the FacilityStreetAddress element).

To better discern which declaration a given FacilityStreetAddress element represents, the XML processor would need to look ahead three places, something XML processors cannot do. This means the content model is nondeterministic and, therefore, illegal.

Wildcards

Pros and Cons

Advantages	Wildcards can be useful when a project is beginning and it is unclear what content will be required at certain points within an XML instance document, or where part of an XML instance document should be unconstrained (e.g., it is a memo or notes field).
Disadvantages	Wildcard declarations place very minor restrictions on the information that can be used in place of them. This can lead to a proliferation of uncontrolled XML instance documents. Wildcard declarations can lead to nondeterministic content models. Use of wildcards may add an additional level of complexity to a schema.
Rules and Guidelines	
Rule	**Description**
[SD7-23]	Exchange Network schemas MUST NOT use wildcards.
Justification	
Although wildcards can be useful in some situations, the additional level of complexity they add to a schema does not outweigh their potential advantages. In addition, their use can be counter to efforts to control the types of information that can appear in XML instance documents.	

2.9.3. Default and Fixed Element and Attribute Values

This section discusses W3C Schema default and fixed element and attribute values.

Default Element Values

When a default value is declared for an element, an XML processor will insert the default value for that element into an XML instance document when it validates the instance document. Any processing application that accepts the XML instance document as an input recognizes the default element value as the actual element value. In other words, a processing application will not be able to discern whether the value was originally included in the XML instance document or inserted by an XML processor. The following is an example:

```
<xsd:element name="FacilityIdentificationCode" type="xsd:string" default="000"/>
```

In the above example, if there is no value for a FacilityIdentificationCode element in an XML instance document, an XML processor will insert "000" in the XML instance document.

For a default element value to take effect, an empty tag must appear in an XML instance document for that element. For example, either of the following two excerpts would be necessary in an XML instance document for the default value to take effect:

```
<FacilityIdentificationCode/>
```

```
or
<FacilityIdentificationCode></FacilityIdentificationCode>
```

If a value appears in an XML instance document that is different than the default element value, the value in the XML instance document will take precedence.

Default Element Values

Pros and Cons	
Advantages	Default element values allow the insertion of additional information in an XML document without user intervention. A default element value can be overridden by including a different value for the element in an XML instance document.
Disadvantages	Default element values may cause data to be inserted in an XML instance document that may not have been the intention of the XML instance document author. For a default element value to take effect, an empty tag must appear in an XML instance document for that element. Certain XML processors may not be able to support default element values.
Rules and Guidelines	
Rule	**Description**
[SD7-25]	Exchange Network schemas SHOULD NOT use default element values.
Justification	
While default element values can be considered an efficient feature of W3C schema, the risk of having data inserted in an XML instance document that may not have been intended for insertion outweighs the potential benefits of their use in data-centric scenarios.	

Fixed Element Values

A fixed element value is handled by an XML processor in the same way as a default element value, with one exception: if a value appears in an XML instance document that is different than the fixed element value, the XML processor will generate an error. An example of a fixed element is as follows:

```
<xsd:element name="ActiveIndicatorCode" type="xsd:boolean" fixed="true"/>
```

In this example, if there is no value for an ActiveIndicatorCode element in an XML instance document, an XML processor will insert "true" in the XML instance document.

Fixed Element Values

Pros and Cons

Advantages	Fixed element values allow the additional insertion of information in an XML document without user intervention. A fixed element value ensures the same element value appears in an XML instance document for a given element, wherever that element appears.
Disadvantages	A fixed element value cannot be overridden in an XML instance document. Fixed element values may cause data to be inserted in an XML instance document, which may not have been the intention of the XML instance document author. For a fixed element value to take effect, an empty tag must appear in an XML instance document for that element. Certain XML processors may not be able to support fixed element values.
Rules and Guidelines	
Rule	**Description**
[SD7-27]	Exchange Network schemas SHOULD NOT use fixed element values.
Justification	
While fixed element values can be considered an efficient feature of W3C Schema, the risk of having data inserted in an XML instance document that may not have been intended for insertion outweighs the potential benefits of their use in data-centric scenarios.	

Default Attribute Values

Default attribute values perform the same function as default element values, with one exception: there is no need for an indication in an XML instance document for a default attribute value to take effect (recall that, with default element values, an empty tag must appear in an XML instance document for that element). The following is an example of a default attribute:

```
<xsd:attribute name="informationFormatIndicator" type="xsd:string" default="A"/>
```

In the above example, if there is no value for an informationFormatIndicator attribute, an XML processor will insert "A" in the XML instance document. As with default element values, if a value appears in an XML instance document that is different than the default attribute value, the value in the XML instance document will take precedence.

Default Attribute Values

Pros and Cons

Advantages	Default attribute values allow additional information to be inserted in an XML document without user intervention. A default attribute value can be overridden by including a different value for the attribute in an XML instance document. An indication is not needed in an XML instance document for a default attribute value to take effect.
Disadvantages	Default attribute values may cause data to be inserted in an XML instance document that may not have been the intention of the XML instance document author. Certain XML processors may not be able to support default attribute values.
Rules and Guidelines	
Rule	**Description**
[SD7-29]	Exchange Network schemas SHOULD NOT use default attribute values.
Justification	
As with default element values, the risk of having data inserted in an XML instance document that may not have been intended for insertion outweighs the potential benefits of using default attributes values in data-centric scenarios.	

Fixed Attribute Values

Fixed attribute values perform the same function as fixed element values. An example of a fixed attribute is as follows:

```
<xsd:attribute name="informationFormatIndicator" type="xsd:string" fixed="A"/>
```

In the above example, if there is no value for an informationFormatIndicator attribute, an XML processor will insert "A" in the XML instance document. As with fixed element values, if a value appears in an XML instance document that is different than a fixed attribute value, the XML processor will yield an error.

Fixed Attribute Values

Pros and Cons	
Advantages	Fixed attribute values allow additional information to be inserted in an XML document without user intervention. A fixed attribute value will ensure that the same element value appears in an XML instance document for a given attribute, wherever that attribute appears. An indication is not needed in an XML instance document for a fixed attribute value to take effect.

Disadvantages	A fixed attribute value cannot be overridden in an XML instance document. Fixed attribute values may cause data to be inserted in an XML instance document that may not have been the intention of the XML instance document author. Certain XML processors may not be able to support fixed attribute values.
Rules and Guidelines	
Rule	**Description**
[SD7-31]	Exchange Network schemas SHOULD NOT use fixed attribute values.
Justification	
As with fixed element values, the risk of having data inserted in an XML instance document that may not have been intended for insertion outweighs the potential benefits of using fixed attributes values in data-centric scenarios.	

2.9.4. Substitution Groups

Substitution groups allow a global element to replace another global element in an XML instance document without any further modifications to the schema. Substitution groups do not apply to local elements. This feature is useful when there are multiple trading partners and an element needs to be represented by a group of trading partners using one name and by another group of trading partners using another name (for instance, if the name is location specific).

The following is an example of a substitution group declaration:

```
<xsd:element name="FacilityIdentificationCode" type="xsd:string"/>
<xsd:element name="StateIdentificationCode" type="xsd:string"
    substitutionGroup="FacilityIdentificationCode"/>
```

The above example declares the StateIdentificationCode element as substitutable for the FacilityIdentificationCode element in an XML instance document. The FacilityIdentificationCode element is known as the head element. Therefore, the following two excerpts are both valid at the same point in an XML instance document:

```
<FacilityIdentificationCode>15849</FacilityIdentificationCode>
and
<StateIdentificationCode>VA</StateIdentificationCode>
```

As with complex datatype derivation, the "xsi:type" W3C Schema instance construct must be used to indicate to an XML processor exactly which declaration—head element or substitutable element—applies.

It is possible to prohibit an element from being "substituted for" in an XML instance document. This is done using a block attribute that prohibits the element from being used as the head element in a substitution group. For example, the following declaration prohibits the FacilityIdentificationCode element from being substituted for another:

```
<xsd:element name="FacilityIdentificationCode" type="xsd:string" block="substitution"/>
```

Substitution Groups

Pros and Cons	
Advantages	Substitution groups allow a global element to replace another global element in an XML instance document without any further modifications to the schema.
Disadvantages	Substitution groups do not promote the harmonization of element names.
Rules and Guidelines	
Rule	**Description**
[SD7-33]	Exchange Network schemas MUST NOT use substitution groups.
Justification	
Harmonization is the key to interoperable data exchange, and use of substitution groups moves away from harmonization.	

2.9.5. Supplemental Instructions

This section discusses inclusion of supplemental instructions in a schema for the purpose of passing them to a processing application. This section begins with a discussion of the W3C Schema appinfo element, which indicates the processing instructions in schemas. The concept of notations is then discussed as a way to allow non-XML data to be associated with an XML document.

W3C appinfo Element

The W3C Schema appinfo element is used to indicate processing instructions in schemas. An example is as follows:

```
<xsd:group name="WaterSampleGroup">
  <xsd:annotation>
    <xsd:appinfo>
      if (WaterCharacteristics.WaterState="Acidic")
        docParser.execute(AcidicProcessing);
      else
        docParser.execute(DefaultProcessing);
    </xsd:appinfo>
  </xsd:annotation>
```

```
  <!—information removed for example purposes—>
</xsd:group>
```

The information within the <xsd:appinfo> element above indicates a certain type of script that is passed to a processing application by an XML processor that processes the XML instance document. The intent of the script in the above example is to test the value of a database field for Acidic and execute a particular program based on that value.

W3C Schema appinfo Element

Pros and Cons	
Advantages	The appinfo element can be useful for passing processing commands or other supplemental information to a processing application.
Disadvantages	Specifying application logic or other commands in the appinfo element mixes data with potentially specific processing commands. Processing logic should be communicated using other available mechanisms for Exchange Network partners.
Rules and Guidelines	
Rule	**Description**
[SD7-38]	Exchange Network schemas MUST NOT use the appinfo element.
Justification	
There is no guarantee that a given XML processor will properly pass the processing instructions to an application, or, if it does, that an application will be able to accept them or handle them properly.	

Notations

A notation is a formal declaration to an XML processor of non-XML external content that is not meant to be parsed (for example, image data). In the W3C Schema standard, notations can be represented only through a derived type, such as an enumeration.

In the following example, the user is given a list of image types (JPEG or GIF). The pertinent program (jpegviewer.exe or gifviewer.exe) is then initiated on user selection:

```
<xsd:notation name=“jpeg” public=“image/jpeg” system=“jpegviewer.exe”/>
<xsd:notation name=“png” public=“image/png” system=“gifviewer.exe”/>
<xsd:element name=“Picture”>
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base=“xsd:hexBinary”>
        <xsd:attribute name=“pictureType”>
          <xsd:simpleType name=“notation.Image”>
            <xsd:restriction base=“xsd:NOTATION”>
              <xsd:enumeration value=“jpeg”/>
```

```
                <xsd:enumeration value="png"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:attribute>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
</xsd:element name>
```

Notations

Pros and Cons	
Advantages	Notations provide an efficient method for including non-XML data in a schema.
Disadvantages	XML processors are not yet mature enough to be able to properly handle this technique.
Rules and Guidelines	
Rule	**Description**
[SD7-40]	Exchange Network schemas MUST NOT use notations.
Justification	
There is no guarantee that a given XML processor or Exchange Network partner will properly process notation information. Specification of highly-specific processing instructions within a schema is not consistent with Exchange Network mechanisms for providing these instructions.	

Appendix A – Summary of Schema Design Rules

This appendix summarizes the XML design rules found in this document. This appendix is intended as a quick reference for developers. For additional information on the feature or on the advantages, disadvantages, and justification for a particular rule, please see the corresponding section for the full commentary as noted by the rule prefix.

Rules and Guidelines	
General XML Design	
[GD1-1]	All Exchange Network schema MUST be based on the W3C suite of technical specifications that hold Recommendation status.
[GD1-2]	Only W3C technical specifications holding Recommendation, Proposed Recommendation, or Candidate Recommendation status shall be used for production activities.
[GD1-3]	W3C technical specifications holding Draft status MAY be used for prototyping. Such prototypes will not be put into production until the associated specifications reach a Recommendation, Proposed Recommendation, or Candidate Recommendation status.
[GD1-4]	All XML parsers, generators, validators, enabled applications, servers, databases, operating systems, and other software acquired or used by partners' activities shall be fully compliant with all W3C XML specifications that hold a Recommendation status.
[GD1-5]	The normative schema documents that implement the partner document types shall conform to XML Schema Part 1: Structures and XML Schema Part 2: Datatypes.
[GD1-6]	Each message MUST represent a single logical unit of information (such as facility permit compliance data) conveyed in the root element.
[GD1-7]	The business function of a message set MUST be unique and must not duplicate the business function of another message.
[GD1-8]	The name of the message set MUST be consistent with its definition.
[GD1-10]	Messages MUST use the UTF-8/UNICODE character set.
[GD1-11]	XML instance documents conforming to schemas SHOULD be readable and understandable, and should enable reasonably intuitive interactions.
[GD1-12]	Messages shall be modeled for the abstractions of the user, not the programmer.
[GD1-13]	Messages shall use markup to make data substructures explicit (that is, distinguish

	separate data items as separate elements and attributes).
[GD1-14]	Messages shall use well-known datatypes.
[GD1-15]	EPA messages shall reuse registered datatypes to the maximum extent practicable.
[GD1-16]	In a schema, information that expresses associations between data elements in different classification schemes (in other words, “mappings”) MAY be regarded as metadata. This information should be accessible in the same manner as the rest of the information in the schema.
XML Tag Naming Conventions	
[GD3-1]	Element names MUST be in “Upper Camel Case” (UCC) convention, where UCC style capitalizes the first character of each word and compounds the name. Example: <UpperCamelCaseElement/>
[GD3-2]	Schema type names MUST be in UCC convention. Example: <DataType/>
[GD3-3]	Attribute names MUST be in “Lower Camel Case” (LCC) convention where LCC style capitalizes the first character of each word except the first word. Example: <UpperCamelCaseElement lowerCamelCaseAttribute=“Whatever”/>
[GD3-4]	Acronyms SHOULD NOT be used, but in cases where they are used, the capitalization SHALL remain Example: <XMLSignature/>, and the acronym SHOULD be defined in the comments of the DTD or Schema or in a separate document noted in the DTD or Schema as providing a tag dictionary so that the meaning of the acronym is clear.
[GD3-5]	Abbreviations SHOULD NOT be used. In cases where they are used, they MUST be a major part of the federal or data standards vocabulary, and the abbreviation SHOULD be defined within the comments of the DTD or Schema or in a separate document (noted in the DTD or Schema) as providing a tag dictionary so that the meaning of the abbreviation is clear. An exception to this rule is when identifier is used as a representation term, ID SHOULD be used as part of the tag name.
[GD3-6]	Underscores (_), periods (.) and dashes (-) MUST NOT be used.
[GD3-7]	Verbosity in tag length SHOULD be limited to what is required to conform to the Tag Name Content recommendations. When tags will be used in database structures, a limit of 30 characters is recommended.
[GD3-8]	Element, attribute, and datatype tag names MUST be unique.
[GD3-10]	High-level parent element tag names SHOULD consist of a meaningful aggregate name followed by the term “Details”. The aggregate name may consist of more than one word. Example: <SiteFacilityDetails/>
[GD3-11]	Tag names SHOULD be concise and MUST NOT contain consecutive redundant words.
[GD3-12]	Lowest level (it has no children) element tag name SHOULD consist of the Object Class, the name of a Property Term, and the name of a Representation Term. An

	Object Class identifies the primary concept of the element. It refers to an activity or object within a business context and may consist of more than one word. Example: <LocationSupplementalText/>
[GD3-13]	A Property Term identifies the characteristics of the object class. The name of a Property Term SHALL occur naturally in the tag definition and may consist of more than one word. A name of a Property Term shall be unique within the context of an Object Class but may be reused across different Object Classes. Example: <LocationZipCode/> and <MailingAddressZipCode/> may both exist.
[GD3-14]	If the name of the Property Term uses the same word as the Representation Term (or an equivalent word), this Property Term SHALL be removed from the tag name. In this case, only the Representation Term word will remain. *Examples:* If the Object Class is "*Goods*", the Property Term is "*Delivery Date*", and Representation Term is "*Date*", the tag name is *<GoodsDeliveryDate/>*
[GD3-15]	A Representation Term categorizes the format of the data element into broad types. A list of UN/CEFACT Representation Terms is included at the end of this list of rules, but the EPA and its partners may need to augment this list to accommodate the specific needs for environmental data. When possible the pre-defined UN/CEFACT list SHOULD be used. Proposed additions should be submitted to the TRG for consideration.
[GD3-16]	The name of the Representation Term MUST NOT be truncated in the tag name.
[GD3-17]	A tag name and all its components MUST be in singular form unless the concept itself is plural.
[GD3-18]	Non-letter characters MUST only be used if required by language rules.
[GD3-19]	Tag names MUST only contain verbs, nouns and adjectives (no words like "and", "of", "the").
[GD3-A]	All datatype names MUST end with either "Type" or "DataType".
Datatypes	
[SD2-1]	Exchange Network schemas MUST use simple datatypes to the maximum extent possible.
[SD2-A]	Schema developers SHOULD ensure that use of built-in data types in schema are appropriate to the data being represented.
[SD2-3]	Exchange Network schemas that employ complex datatypes MUST define the complex datatypes as global.
[SD2-5]	Exchange Network schemas SHOULD NOT use local complex datatypes.
Elements and Attributes	
[SD3-1]	Exchange Network schemas MUST use global elements.
[SD3-3]	Exchange Network schemas SHOULD NOT use local elements.

[SD3-5]	Exchange Network schemas SHOULD use occurrence indicators.
[SD3-6]	Exchange Network schemas SHOULD NOT use occurrence indicators when the required values are the default values.
[SD3-9]	Exchange Network schemas MUST NOT use attributes in place of data elements.
[SD3-10]	Exchange Network schemas MAY use attributes for metadata.
[SD3-12]	Exchange Network schemas MUST NOT use global attributes in place of data elements
[SD3-13]	Exchange Network schemas MAY use global attributes for metadata.
[SD3-15]	Exchange Network schemas MUST NOT use local attributes in place of data elements.
[SD3-16]	Exchange Network schemas MAY use local attributes for metadata.
[SD3-18]	Exchange Network schemas SHOULD use the "use" indicator.
[SD3-19]	Exchange Network schemas SHOULD NOT use the "use" indicator when the required value is the default value.
[SD3-22]	Exchange Network schemas SHOULD use the "sequence" compositor.
[SD3-24]	Exchange Network schemas SHOULD use the "choice" compositor.
[SD3-26]	Exchange Network schemas MUST NOT use the "all" compositor.
[SD3-28]	Exchange Network schemas MAY use model groups.
[SD3-30]	Exchange Network schemas MUST NOT use attribute groups in place of data elements.
[SD3-31]	Exchange Network schemas MAY use attribute groups for metadata.
Namespaces	
[SD4-1]	Exchange Network schemas MUST use namespaces.
[SD4-2]	Exchange Network schemas MUST use namespace qualification for all schema constructs.
[SD4-A]	The schema namespace name MUST be URL-formatted as “http://www.exchangenetwork.net/schema/”{ExchangeIdentifier}[“/”{Category}]”/”{Version}.
[SD4-B]	Exchange Network namespaces MUST contain the Exchange Identifier term that clearly and uniquely defines the type of data being exchanged.
[SD4-C]	Exchange Network namespaces MAY contain a category term which further divides the data exchange into smaller components.

[SD4-D]	Exchange Network namespaces MUST contain a major version number as the last part of the namespace name.
[SD4-E]	Exchange Network namespaces MUST NOT contain a minor version number in the namespace name.
[SD4-13]	Exchange Network schemas MUST use target namespaces.
[SD4-5]	Exchange Network schemas MUST declare the W3C Schema namespace.
[SD4-6]	Exchange Network schemas MUST use namespace qualification for all W3C Schema constructs.
[SD4-11]	Exchange Network schemas SHOULD NOT declare the W3C Schema Datatypes namespace.
[SD4-15]	Exchange Network schemas SHOULD reference external schemas.
[SD4-16]	Exchange Network schemas MAY use the include construct.
[SD4-17]	Exchange Network schemas MAY use the import construct.
[SD4-27]	Exchange Network schemas MAY use multiple namespaces.
[SD4-23]	Exchange Network schemas MUST NOT use default namespaces.
[SD4-35]	Exchange Network XML instance documents MUST be validated against a schema during processing.
[SD4-36]	Exchange Network XML instance documents SHOULD list the storage location of the schema where the XML instance document validates in the root element.
[SD4-H]	If a schemaLocation is specified in an XML instance document, the location MUST match the namespace URL address.
[SD4-38]	Exchange Network XML instance documents MUST NOT use the noNamespaceSchemaLocation construct when listing the storage location of the schema to which the XML instance document validates.
[SD4-43]	Exchange Network XML instance documents MUST use namespace qualification for all elements.
[SD4-45]	Exchange Network XML instance documents MUST declare the W3C Schema Instance namespace when W3C Schema Instance constructs are used.
[SD4-46]	Exchange Network XML instance documents SHOULD use "xsi" as a namespace prefix for all W3C Schema Instance constructs.
[SD4-49]	Exchange Network XML instance documents MUST NOT use local namespace declarations.
Schema Configuration and Documentation	

[SD5-1]	Message-level Schemas SHOULD be used.
[GD1-D]	Message-level root elements MUST be defined in the Message Schema.
[SD5-R]	Exchange Network schema MUST be modularized into default, message, component, and shared schema as described in schema guidelines.
[SD5-S]	The exchange default schema (index.xsd) MUST be stored in a directory with a name that matches the exchange major version number and the message, component, and shared schema files MUST be stored in a subdirectory matching the exchange minor version number.
[GD1-A]	Elements with recurring, complex content SHOULD NOT be referenced more than once within the same root document, even if the element exists in different schema files. When the same structure needs to be reused in multiple areas, either create a new element with a different tag name utilizing a common datatype or use the KEY and KEYREF construct to represent the element in a single list.
[SD5-A]	Developers MUST use SSC in schema when they are an appropriate fit for the targeted business process.
[SD5-B]	Developers SHOULD create custom datatypes and elements only after determining that no existing SSC adequately describes the given construct.
[SD5-C]	Existing schema SHOULD be evaluated for SSC compatibility and subsequently updated to include SSC elements and/or datatypes at the time of the next revision.
[SD5-D]	Developers SHOULD use XML datatype extension and restriction to modify SSC types.
[SD5-7]	Appropriate Voluntary Standard Body Schemas SHOULD be adopted, when appropriate.
[SD5-12]	Schemas developed outside of the Exchange Network MAY be used if they are consistent with the guidelines for schema development as set forth in Exchange Network rules and guidance.
[SD5-14]	Exchange Network schemas SHOULD group like constructs into one schema.
[SD5-15]	Message-level schemas SHOULD maintain a reasonable number of nested includes.
[SD5-28]	All schema SHOULD include schema construct documentation using the W3C documentation element. Documentation SHOULD only describe the element or datatype and SHOULD NOT contain lists of acceptable codes, implementation details or other information not directly related to the meaning of the construct.
[SD5-29]	Exchange Network schemas SHOULD use the documentation element for schema construct documentation.
[SD5-30]	HTML-style comments (<!--comment -->) SHOULD NOT be used in schema.
[SD5-34]	Exchange Network schemas MUST include schema header documentation.

Schema Versioning	
[SD5-E]	New minor versions of schema MUST be able to validate instance documents created with preceding minor versions of that schema. However, instance documents should not be expected to validate against versions of schema preceding the one they were created with.
[SD5-F]	New minor versions of a schema MUST only add new optional elements and/or attributes to prior minor versions.
[SD5-G]	New minor versions of a schema MUST NOT remove elements and/or attributes from prior minor versions.
[SD5-H]	New minor versions MUST utilize the exact same namespace as prior minor versions.
[SD5-I]	All existing instance documents in the same namespace MUST validate against the new minor version.
[SD5-J]	The schema major version MUST be incremented if any elements or attributes are removed or if new mandatory elements or attributes are added.
[SD5-K]	The schema file name, XSD version attribute, header documentation and namespace MUST all contain matching version information.
[SD5-L]	When any schema construct is altered in a given namespace, all schema in the namespace MUST undergo a version increment.
[SD4-A]	The schema namespace name MUST be URL-formatted as “http://www.exchangenetwork.net/schema/”{ExchangeIdentifier}[“/”{Category}]”/”{Version}.
[SD4-B]	Exchange Network namespaces MUST contain the Exchange Identifier term that clearly and uniquely defines the type of data being exchanged.
[SD4-C]	Exchange Network namespaces MAY contain a module name which further divides the data exchange into smaller components.
[SD4-D]	Exchange Network namespaces MUST contain a major version number as the last part of the namespace name.
[SD4-E]	Exchange Network namespaces MUST NOT contain a minor version number in the namespace name.
[GD2-2]	File names MUST NOT use abbreviations unless their meaning is beyond question (EPA, GSA, FBI).
[GD2-A]	Each namespace must contain a default schema named “index.xsd” which contains only an include construct referencing the root schema.
[GD2-C]	Message schema MUST utilize the following naming format: {ExchangeIdentifier}["_"{ExchangeCategoryName}]"_"{MessageName} "_v"{Version}".xsd".

[GD2-D]	Component Message schema MUST utilize the following naming format: {ExchangeIdentifier}["_"{ExchangeCategoryName}]"_"{ComponentName} "_v"{Version}".xsd".
[GD2-E]	Local shared schema MUST utilize the following naming format: {ExchangeIdentifier}["_"{ExchangeCategoryName}]"_Shared_v"{Version}".xsd".
[SD5-22]	Exchange Network schemas MUST include a major and minor version number in their filename.
[SD5-20]	Exchange Network schemas MUST include a version number using the W3C Schema version attribute.
[SD5-21]	The W3C Schema version attribute MUST include both a major version component and a minor version component.
[SD5-26]	Exchange Network schemas MUST define a required attribute named "schemaVersion" in the root element of all message schema.
Information Association and Uniqueness	
[SD6-1]	Exchange Network schemas MUST NOT use the ID/IDREF technique for information association.
[SD6-3]	Exchange Network schemas SHOULD use the KEY/KEYREF technique for information association.
[SD6-4]	Extreme caution SHOULD be applied when writing an XPath expression in a selector element to ensure it specifies the intended range.
[SD6-5]	Special attention SHOULD be paid to the restrictions on KEY and KEYREF declaration names given above.
[SD6-9]	Exchange Network schemas SHOULD use the KEY technique to enforce uniqueness of values in an XML instance document when their constructs are required to appear within the specified range.
[SD6-10]	Extreme caution SHOULD be applied when writing an XPath expression in a selector element to ensure it specifies the intended range.
[SD6-11]	Special attention SHOULD be paid to the restrictions on KEY declaration names given above.
[SD6-15]	Exchange Network schemas MUST NOT use the XLink/XPointer technique for information association.
[SD6-17]	Exchange Network schemas SHOULD use the UNIQUE technique to enforce uniqueness of values in an XML instance document when their constructs are not required to appear within the specified range.
[SD6-18]	Extreme caution SHOULD be applied when writing an XPath expression in a selector element to ensure it specifies the intended range.

[SD6-19]	Special attention SHOULD be paid to the restrictions on UNIQUE declaration names given above.
Advanced W3C Concepts	
[SD7-1]	Exchange Network schemas SHOULD NOT use simple datatype restriction when a data standard or an approved schema exists.
[SD7-2]	Exchange Network schemas MUST use global simple datatypes.
[SD7-3]	Exchange Network schemas MUST NOT use local simple datatypes.
[SD7-A]	Exchange Network schemas SHOULD define facets such as string lengths, patterns, and numeric ranges when the schema targets a specific data system.
[SD7-7]	Exchange Network schemas MAY use the list technique.
[SD7-8]	Exchange Network schemas MUST NOT use the list technique if the values within the list may contain spaces themselves (e.g., a person's first and last name).
[SD7-11]	Exchange Network schemas MAY use the union technique.
[SD7-13]	Exchange Network schemas MAY use complex datatype restriction.
[SD7-15]	Exchange Network schemas MAY use complex datatype extension.
[SD7-17]	Exchange Network schemas MAY use the final attribute derivation.
[SD7-19]	Exchange Network schemas MAY use the block attribute.
[SD7-21]	Exchange Network schemas MUST NOT use abstract datatypes.
[SD7-23]	Exchange Network schemas MUST NOT use wildcards.
[SD7-25]	Exchange Network schemas SHOULD NOT use default element values.
[SD7-27]	Exchange Network schemas SHOULD NOT use fixed element values.
[SD7-29]	Exchange Network schemas SHOULD NOT use default attribute values.
[SD7-31]	Exchange Network schemas SHOULD NOT use fixed attribute values.
[SD7-33]	Exchange Network schemas MUST NOT use substitution groups.
[SD7-38]	Exchange Network schemas MUST NOT use the appinfo element.
[SD7-40]	Exchange Network schemas MUST NOT use notations.