Chapter 1
Schema Introduction

The XML technical specification identified a standard for writing a schema (i.e., an information model) for XML called a document type definition (DTD).[1] DTDs were a carryover from the SGML (ISO Standard 8879) and provided the ability to define the structure of a document, but lacked the ability to add data typing to the requirements placed on the XML document by the schema. Although DTDs work well for document-centric XML, they are not ideal for data-centric XML because they lack data typing. With a primary mission to add data typing to XML, the W3C developed and maintains another specification, XML Schema. This specification is now the Environmental Information Exchange Network standard for developing new XML message exchanges.

DTD MIGRATION TO W3C SCHEMA

Some environmental implementations may already use DTDs. In those cases, when updating a project or system, a migration from DTD to XML Schemas should be strongly considered. In addition to the aspects of datatyping mentioned above, additional technical aspects of XSD overpower DTDs such as the namespace feature of XSDs. Although namespacing is possible within DTDs, it is much easier to implement in XSD—details of namespaces in XSDs are discussed in-depth later in this document. Another important aspect of XSD is its inherent feature of providing object-oriented design features, whereas DTD allows for creation of only relational structures. This feature allows you to create objects (complexTypes) that can be both extended and restricted for other uses, providing a much higher degree of reusability.

XSDs are written in XML, enabling vendors to take advantage of XML parsers. As XSDs continue to gain further traction, vendor-supported tools are becoming more readily available and more competitive. In addition, most new standard vocabularies are based on W3C Schemas (e.g., the OASIS Universal Business Language and the UN/CEFACT's work in the Applied Technologies Group). Thus, in accordance with this document, all future efforts should use XSD, and when possible, current DTDs should be migrated to XSDs.

[1] See <http://www.w3.org/TR/2000/REC-xml-20001006>.

DATA-CENTRIC AND DOCUMENT-CENTRIC XML

This guide separates the application of XML into two types: data centric and document centric. Data-centric XML is used in data exchange environments; document-centric XML is used in a content management environment. Data-centric XML is geared toward machine processing; document-centric XML is geared toward formatting and human consumption. An example of data-centric XML is information passed between an order management system and an inventory management system. An example of document-centric XML is information formatted for inclusion in a book, brochure, or website.

Data-Centric XML

Developers use data-centric XML for the structured electronic exchange of data across the Internet (for example, when information is sent from one database to another or when a person inputs data into a web form and submits the data to a database). Data-centric XML focuses on data types and, therefore, must be more rigid than document-centric XML. Usually, in data-centric XML, the XML instance generates automatically based on an XML schema and input into a back-end database without human intervention.

The following are characteristics of data-centric XML:

- Has granular detail (numerous tags)
- Has nonvariable structure
- Is machine generated.

The Exchange Network wants to ensure consistency and interoperability for all data-centric XML exchanges. As a result, the design rules for data-centric applications are far stricter that those for document-centric applications.

Document-Centric XML

Document-centric XML is used primarily for presentations and often contains graphics. Document-centric XML is far less rigid than data-centric XML, and typically defines the structure at a higher level. In document-centric XML, an author creates the XML instance based on an XML schema, which is combined with a stylesheet that renders the information in a specified format.

The following are characteristics of document-centric XML:

- Has broad detail (few tags)
- Has free-form structure
- Is human generated (using XML authoring tools).

FREQUENTLY USED TERMS

The terms “schema construct” and “construct visibility” are frequently used throughout this document. Therefore, a detailed explanation of these terms and examples of their use are provided below to clarify their meaning.

Schema Construct

The term “schema construct” (or simply “construct”) refers to an XML element, attribute, or datatype whenever the same concept applies to all three. The term “W3C Schema construct” is used to refer to schema constructs that are part of the W3C Schema markup vocabulary (in other words, part of the W3C Schema language). For example, in the declaration below, “element,” “name,” and “type” are W3C Schema constructs:

```
<xsd:element name=“FirstName” type=“xsd:string”>
```

Construct Visibility

Construct visibility refers to the level at which a schema construct can be accessed from multiple points in a schema (and, therefore, reused). More specifically, this term is applied to elements and datatypes. “High element visibility” means that an element in a schema can be accessed from multiple places in the schema (and, therefore, reused). “Low element visibility” means that an element in a schema cannot be accessed from any other place in the schema (and, therefore, it cannot be reused). The same concept applies for the terms “high datatype visibility” and “low datatype visibility.”

It is possible to reference one or more additional schemas from within a schema. If a schema construct can be accessed from multiple points within a schema, it can be accessed from other schemas as well. This is important for the Exchange Network because the use of schema constructs across the network is closely linked to their visibility. If a construct has high visibility, it can be made visible across the network and integrated into multiple data flows. This means the construct is a candidate for harmonization efforts. If a construct has low visibility, it cannot be made visible across the network and cannot be integrated into multiple data flows.

The Exchange Network should strive for high construct visibility in data-centric schemas because high construct visibility ensures consistency and interoperability in all data-centric XML exchanges. High construct visibility is not as important in document-centric schemas because document-centric XML is used mainly for presentation purposes.

Chapter 2
Datatypes

One important advantage for the W3C Schema standard over DTDs is its capability to define datatypes for elements and attributes. Datatypes represent the kind of information elements and attributes can hold—character strings or dates, for example. This chapter discusses datatypes and their use in schemas, and provides guidance for the Exchange Network's use of XML Schema.

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.[1] 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.

Simple 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	
Data-centric:	[SD2-1] Data-centric schemas MUST use simple datatypes to the maximum extent possible.
Document-centric:	[SD2-2] Document-centric schemas SHOULD use simple datatypes.

[1] "The properties that define the characteristics of a value space are known as facets; facets include equality, order, bounds, cardinality, and numeric/non-numeric." *Professional XML Schemas*, WROX Press Ltd, 2001.

Simple Datatypes

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. Document-centric schemas often will include sections of text. These sections often will not require high levels of validation because the text is meant for human rather than machine consumption. Because of this factor, the less stringent guidance of "should" is recommended.

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 in a single location in a schema, thereby lowering maintenance costs.
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	
Data-centric:	[SD2-3] Data-centric schemas that employ complex datatypes MUST define the complex datatypes as global.
Document-centric:	[SD2-4] Document-centric schemas that employ complex datatypes SHOULD 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. Because schema construct visibility is not as important for document-centric schemas as for data-centric schemas, use of global complex datatypes is not required for document-centric schemas. 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	
Data-centric:	[SD2-5] Data-centric schemas SHOULD NOT use local complex datatypes.
Document-centric:	[SD2-6] Document-centric schemas MAY use local complex datatypes.
Justification	
Use of local complex datatypes is discouraged for data-centric schemas because they result in low datatype visibility. However, because schema construct visibility is not as important for document-centric schemas as for data-centric schemas, local complex datatypes may be used in document-centric schemas. Although there is a potential requirement for additional schema updates, the potential advantages for using local elements in document-centric schemas far outweigh the potential disadvantages.	

Chapter 3
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.

ELEMENTS

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

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

In the above example, the FacilitySiteDetails element is a subelement of the AAREASubmission element, while the FacilityIdentificationCode and FacilityAddressDetails elements are subelements of the FacilitySiteDetails element. Elements can be extended as necessary (i.e., a schema developer can add subelements 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—AAREASubmission and AEVENTSubmission. Therefore, an XML instance document that conforms to this schema can have either the AAREASubmission element or the AEVENTSubmission element as its root:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="AAREASubmission" type="AAREASubmissionType"/>
    <xsd:element name="AEVENTSubmission" type="AEVENTSubmissionType"/>
    <!—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 subelements.

Continuing with the same example, if an XML instance document has the AAREASubmission element as its root element, it can include the AAREASubmission element and its subelements (specifically, the elements contained within the AAREASubmissionType datatype). However, it cannot include the AEVENTSubmission element or its subelements.

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 subelements, 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	
Data-centric:	[SD3-1] Data-centric schemas MUST use global elements.
Document-centric:	[SD3-2] Document-centric schemas SHOULD 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.	
Because schema construct visibility is not as important for document-centric schemas as for data-centric schemas, use of global elements is not required for document-centric schemas.	
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.	

Note: There is concern that the requirement for global datatypes will require a great deal of revision of existing schema, and that the manageability of global datatypes will depend on the namespace. Global elements could become unmanageable in a large namespace.

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	
Data-centric:	[SD3-3] Data-centric schemas SHOULD NOT use local elements.
Document-centric:	[SD3-4] Document-centric schemas MAY use global elements.
Justification	
Use of local elements is discouraged for data-centric schemas because they result in low element visibility; however, because schema construct visibility is not as important for document-centric schemas as for data-centric schemas, local elements may be used in document-centric schemas.	
Although there is a potential requirement for additional schema updates in a scenario where a local element is declared in multiple places, the potential advantages for using local elements in document-centric schemas far outweigh the potential disadvantages.	

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.

One important advancement for the W3C Schema standard over DTDs is the capability to define specific cardinality values for elements. While DTDs allowed for general declaration of cardinality (“1 or more”; “0 or 1”), the W3C Schema standard allows for specification of the exact number of allowed occurrences of an element.

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="AAREASubmissionType">
            <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="AAREASubmissionType">
        <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	
Data-centric:	[SD3-5] Data-centric schemas SHOULD use occurrence indicators. [SD3-6] Data-centric schemas SHOULD NOT use occurrence indicators when the required values are the default values.
Document-centric:	[SD3-7] Document-centric schemas SHOULD use occurrence indicators. [SD3-8] Document-centric 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.	

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	
Data-centric:	[SD3-9] Data-centric schemas MUST NOT use attributes in place of data elements.
	[SD3-10] Data-centric schemas MAY use attributes for metadata.
Document-centric:	[SD3-11] Document-centric schemas MAY use attributes.
Justification	
The use of attributes is prohibited for data-centric schemas because data-centric 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. Attributes are useful in document-centric schemas to convey metadata, as in the following example: <paragraph amended="02-01-2002">. The order and extension of information is not as important for document-centric schemas as for data-centric schemas because, in document-centric scenarios, data are not exchanged. Therefore, attributes may be used in document-centric schemas.	

Global Attributes

Global attributes are direct descendants of the root element schema. 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	
Data-centric:	[SD3-12] Data-centric schemas MUST NOT use global attributes in place of data elements.
	[SD3-13] Data-centric schemas MAY use global attributes for metadata.
Document-centric:	[SD3-14] Document-centric schemas MAY use global attributes.
Justification	
Global attributes are valuable for document-centric schemas because they can be referenced within any complex datatype definition in a schema.	
Although there is a potential requirement for additional schema updates in the propagation scenario discussed above, the potential advantages for using global attributes in document-centric schemas far outweigh the potential disadvantages.	
The potential overhead on an XML processor to resolve all references to global attributes is not of high enough concern to warrant not recommending their use for document-centric schemas.	

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	
Data-centric:	[SD3-15] Data-centric schemas MUST NOT use local attributes in place of data elements. [SD3-16] Data-centric schemas MAY use local attributes for metadata.
Document-centric:	[SD3-17] Document-centric schemas MAY use local attributes.
Justification	
Although there is a potential requirement for additional schema updates for a scenario where local attributes are declared in multiple places, the potential advantages for using local attributes in document-centric schemas far outweigh the potential disadvantages.	

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).

"use" Indicator

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	
Data-centric:	[SD3-18] Data-centric schemas SHOULD use the "use" indicator.
	[SD3-19] Data-centric schemas SHOULD NOT use the "use" indicator when the required value is the default value.
Document-centric:	[SD3-20] Document-centric schemas SHOULD use the "use" indicator.
	[SD3-21] Document-centric 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.	

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	
Data-centric:	[SD3-22] Data-centric schemas SHOULD use the "sequence" compositor.
Document-centric:	[SD3-23] Document-centric schemas SHOULD use the "sequence" compositor.
Justification	
The ability to enforce element order is very valuable, especially in data-centric scenarios where the order of the information is important.	

"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	
Data-centric:	[SD3-24] Data-centric schemas SHOULD use the "choice" compositor.
Document-centric:	[SD3-25] Document-centric 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	
Data-centric:	[SD3-26] Data-centric schemas MUST NOT use the “all” compositor.
Document-centric:	[SD3-27] Document-centric schemas SHOULD use the “all” compositor.
Justification	
The ability to allow elements to appear in any order is very valuable in document-centric scenarios. However, because data-centric scenarios are more structured than document-centric scenarios, it is important that order be enforced in data-centric scenarios. The use of the “all” compositor is therefore prohibited for data-centric schemas. 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	
Data-centric:	[SD3-28] Data-centric schemas MAY use model groups.
Document-centric:	[SD3-29] Document-centric 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.

Attribute Groups

Rules and Guidelines	
Data-centric:	[SD3-30] Data-centric schemas MUST NOT use attribute groups in place of data elements.
	[SD3-31] Data-centric schemas MAY use attribute groups for metadata.
Document-centric:	[SD3-32] Document-centric schemas MAY use attribute groups.
Justification	
Model groups allow attributes to be associated with multiple elements 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 attribute groups in document-centric schemas far outweigh the potential disadvantages. The potential overhead on an XML processor to resolve all references to attribute groups is also not of high enough concern to warrant not recommending its use.	

Chapter 4
Namespaces

Namespaces associate schema constructs with a conceptual space that defines a markup vocabulary.[1] This chapter discusses namespaces and their potential uses, and it provides guidance for the Exchange Network. It is divided into two sections:

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

NAMESPACES AND SCHEMAS

The following concepts are covered in this section:

- *Namespace declaration and qualification*—the declaration of namespaces in schemas and designation of constructs belonging to those namespaces
- *W3C Schema namespaces*—a set of namespaces specific to the W3C Schema standard
- *Target namespaces*—the mechanism used to declare constructs in a schema that can be identified as a single set of constructs associated with that schema
- *External Schema references*—the referencing of one or more additional schemas within a schema
- *Default namespaces*—an efficient way to associate schema constructs with a specific namespace without namespace prefixing
- *Namespaces and attributes*—the special treatment of attributes in namespaces.

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."

[1] Department of the Navy, *XML Developer's Guide*, October 29, 2001.

In the following example, the namespace identifier is "urn:us:net:exchangenetwork" and the namespace prefix is "ExchangeNetwork":

```
<schema xmlns:ExchangeNetwork="urn:us:net:exchangenetwork">
```

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

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

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 "state" elements are used in the same schema, but they are associated with two different namespaces. One element represents a U.S. state abbreviation (e.g., AK, AL, AR), while the other represents the state of water quality (e.g., acidic, basic, high turbidity):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:ExchangeNetwork="urn:us:net:exchangenetwork"
            xmlns:vadeq="http://www.state.va.us/xml">
    <xsd:element name="ExchangeNetwork:State" type="ExchangeNetwork:StatePostalCodeType"/>
    <xsd:element name="vadeq:State" type="vadeq:WaterQualityIndicatorType"/>
    <!—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.

Namespace Declaration and Qualification—Schemas

Pros and Cons	
Advantages:	Namespaces associate schema constructs with a conceptual space. Namespace qualification of schema constructs identifies the namespace where the constructs belong. Namespaces allow constructs with the same name but from different markup vocabularies to be used in the same schema with no adverse effects.
Disadvantages:	Namespace qualification of schema constructs can increase verbosity in a schema and hinder readability.
Rules and Guidelines	
Data-centric:	[SD4-1] Data-centric schemas MUST use namespaces. [SD4-2] Data-centric schemas MUST use namespace qualification for all schema constructs.
Document-centric:	[SD4-3] Document-centric schemas MUST use namespaces. [SD4-4] Document-centric schemas MUST use namespace qualification for all schema constructs.
Justification	
Use of namespaces will be very valuable for the Exchange Network because it will allow constructs developed in different areas to be associated with their own unique conceptual space. Although namespace qualification of schema constructs can increase verbosity, the ability to easily identify the namespace where a construct belongs (visually or automatically) is very valuable.	

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="ExchangeNetwork:FacilityIdentificationCode" type="xsd:string"/>
```

Although user-defined, the prefix "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	
Data-centric:	[SD4-5] Data-centric schemas MUST declare the W3C Schema namespace.
	[SD4-6] Data-centric schemas MUST use namespace qualification for all W3C Schema constructs.
	[SD4-7] Data-centric schemas SHOULD use "xsd" as a namespace prefix for all W3C Schema constructs.
Document-centric:	[SD4-8] Document-centric schemas MUST declare the W3C Schema namespace.
	[SD4-9] Document-centric schemas MUST use namespace qualification for all W3C Schema constructs.
	[SD4-10] Document-centric schemas SHOULD use "xsd" as a namespace prefix 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 DATATYPES 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).

The W3C Schema Datatypes 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	
Data-centric:	[SD4-11] Data-centric schemas SHOULD NOT declare the W3C Schema Datatypes namespace.
Document-centric:	[SD4-12] Document-centric 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.	

Target Namespaces

Declaration of a target namespace in a schema indicates that the schema is acting as a "collector" of constructs declared within it. While a schema may have more than one declared namespace, only one namespace can be designated as the target namespace. It is not required that a target namespace be declared in a schema.

A target namespace is declared using the namespace identifier of the selected namespace. In the following example, the "urn:us:net:exchangenetwork" namespace is declared as the target namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:ExchangeNetwork="urn:us:net:exchangenetwork"
targetNamespace="http://www.epa.gov/exchangenetwork"
```

This means that any local element or datatype in the schema with a namespace prefix "Exchange Network" belongs to the schema's target namespace. Global elements and datatypes are always included in the target namespace and, therefore, are not namespace qualified.

Pros and Cons	
Advantages:	Declaration of a target namespace in a schema indicates that the schema is acting as a "collector" of constructs declared within it.
Disadvantages:	There are no disadvantages to this technique.
Rules and Guidelines	
Data-centric:	[SD4-13] Data-centric schemas MUST use target namespaces.
Document-centric:	[SD4-14] Document-centric 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.	

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 FacilityIdentification.xsd schema (the included schema) is the "urn:us:net:exchangenetwork" namespace:

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

This means any constructs within the FacilityIdentification.xsd schema can be used in the above schema. This technique can be beneficial if an organization wants to confine the use of constructs within schemas to those that belong to a particular namespace—perhaps for configuration management purposes.

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 NEISchema.xsd schema (the included schema) is "urn:us:gov:epa":

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ExchangeNetwork="urn:us:net:exchangenetwork" xmlns:epa="urn:us:gov:epa"
targetNamespace="urn:us:net:exchangenetwork">
    <xsd:import schemaLocation= "NEISchema.xsd" namespace="urn:us:gov:epa"/>
    <!—information removed for example purposes—>
</xsd:schema>
```

Any constructs within the NEISchema.xsd schema can be used in the above schema; however, because these schemas have different target namespaces, the target namespace of the NEISchema.xsd schema must be declared in the root element of the including schema. This technique can be useful if an organization does not need to confine the use of constructs within schemas to only those that belong to a particular namespace.

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	
Data-centric:	[SD4-15] Data-centric schemas SHOULD reference external schemas.
	[SD4-16] Data-centric schemas MAY use the include construct.
	[SD4-17] Data-centric schemas MAY use the import construct.
Document-centric:	[SD4-18] Document-centric schemas MAY reference external schemas.
	[SD4-19] Document-centric schemas MAY use the include construct.
	[SD4-20] Document-centric 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 an organization, it is possible to have a single namespace that is used as the target namespace for all schemas within that organization (referred to as a single-namespace configuration) or to have multiple namespaces (referred to as a multiple- namespace configuration). The example for the include construct above demonstrates a single-namespace configuration, while the example for the import construct demonstrates a multiple-namespace configuration.

Single/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.
	It is not possible to represent the structure or organization of a markup vocabulary with a single-namespace configuration.
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	
Data-centric:	[SD4-21] Data-centric schemas SHOULD use a multiple-namespace configuration.
Document-centric:	[SD4-22] Document-centric schemas SHOULD use a multiple-namespace configuration.
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 allows constructs that are not namespace qualified to belong to a namespace. While a schema may have more than one declared namespace, only one namespace can be designated as the default namespace. It is not required 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="urn:us:net:exchangenetwork">
```

In the following example, the FacilityIdentificationCode element belongs to the default namespace ("urn:us:net:exchangenetwork") 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="urn:us:net:exchangenetwork"
targetNamespace="urn:us:net:exchangenetwork">
    <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 "urn:us:net:exchangenetwork" 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	
Data-centric:	[SD4-23] Data-centric schemas MUST NOT use default namespaces.
Document-centric:	[SD4-24] Document-centric schemas MUST NOT use default namespaces.
Justification	
Although namespace qualification of schema constructs can increase verbosity, the ability to easily identify the namespace where a construct belongs (visually or automatically) is very valuable. Use of default namespaces can cause namespace coercion, making it impossible to discern the origin of schema constructs by examining the including schema. The recommendation that schemas must use namespace qualification for all constructs ensures that namespace coercion can never occur.	

Namespaces and Attributes

Attributes do not belong to a namespace unless they are explicitly namespace qualified with a namespace prefix. Default namespaces do not apply to attributes—an attribute will never be included in a default namespace, even if the element with which the attribute is associated belongs to the default namespace of the schema.

In the following example the InformationFormatIndicator attribute does not belong to the "urn:us:net:exchangenetwork" namespace, even though the FacilitySiteDetails element belongs to that default namespace:

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

Namespaces and Attributes

Pros and Cons	
Advantages:	Namespace qualification of attributes identifies the namespace to which they belong.
Disadvantages:	Namespace qualification of attributes can increase verbosity in a schema.
Rules and Guidelines	
Data-centric:	[SD4-25] Data-centric schemas MUST use namespace qualification for all attributes.
Document-centric:	[SD4-26] Document-centric schemas MUST use namespace qualification for all attributes.
Justification	
Although namespace qualification of attributes can increase verbosity, the ability to easily identify the namespace where an attribute belongs (visually or automatically) is very valuable.	

EXCHANGE NETWORK NAMESPACE CONFIGURATION

This section addresses the architectural namespace configuration for developing Exchange Network XML namespaces under a common methodology.[2] The technical aspects namespaces are also addressed.

[2] For a detailed discussion about and rationale for developing the EPA's namespace and versioning conventions, see Logistics Management Institute, *XML Schema Namespace and Versioning Strategy for the Environmental Information Exchange Network*, LMI Report EP211L1, Mark Crawford and Jessica Glace, December 2002.

The configuration of Exchange Network namespaces comprises multiple shared namespaces where commonly used XML constructs will be housed.[3] The following namespaces will be part of this Exchange Network Schema Configuration Architecture:

An Enterprise Exchange Network namespace to hold message-level and shared schemas

One namespace for EPA message-level and functional area schemas

The option for states to use one state-specific namespace.

Multiple Namespaces

Pros and Cons	
Advantages:	A multiple-namespace configuration decreases the risk of name collision.
	A multiple-namespace configuration can call out a single enterprise namespace to promote interoperability, while organization-specific namespaces promote flexibility for rapid implementation.
	The main advantage of this option is that it has greater interoperability and reduces the complexity of allowing one namespace for each major data group.
	Allowing EPA only one namespace will promote harmonization within the agency, resulting in overall efficiency for all network stakeholders.
Disadvantages:	A multiple-namespace configuration becomes more complex as more namespaces are used.
	A multiple-namespace configuration does not enable the highest degree of interoperability.
	A multiple-namespace configuration requires the use of both the include and import constructs for external schema references, increasing the complexity of Exchange Network schemas.
	Placing all major data groups in one namespace may be too cumbersome an amount of data to have in one Exchange Network namespace, and conflicts between XML construct names may become widespread.
	Allowing EPA only one namespace initially may be resisted as a voluntary guideline.

[3] For a detailed discussion about and rationale for developing the EPA's namespace and versioning conventions, see Logistics Management Institute, *XML Schema Namespace and Versioning Strategy for the Environmental Information Exchange Network*, LMI Report EP211L1, Mark Crawford and Jessica Glace, December 2002.

Multiple Namespaces

Rules and Guidelines	
Data-centric:	[SD4-27] Exchange Network schemas MAY use multiple namespaces.
	[SD4-28] Exchange Network schemas MUST use urn:us:net:exchangenetwork as the target namespace.
	[SD4-29] EPA schemas MUST use urn:us:gov:epa as the target namespace.
	[SD4-30] Each state MAY have one unique namespace for use in Network exchanges.
Document-centric:	[SD4-31] Exchange Network schemas MAY use multiple namespaces.
	[SD4-32] Exchange Network schemas MUST use urn:us:net:exchangenetwork as the target namespace.
	[SD4-33] EPA schemas MUST use urn:us:gov:epa as the target namespace.
	[SD4-34] Each state MAY have one unique namespace for use in Network exchanges.
Justification	
Creating a shared Exchange Network namespace where commonly used XML constructs will be housed mitigates the risk of creating a lower level of interoperability. Allowing additional namespaces (e.g., EPA and state namespaces) increases flexibility and reduces initial cost and time by allowing organizations to individually develop within their own namespaces. Limiting the number of additional possible namespaces (e.g., EPA and states are each allowed only one) mitigates the risks of the configuration becoming too complex and lowering interoperability. One EPA namespace promotes harmonization within the agency. Although this solution is not optimal for *complete* interoperability in future systems, it is a compromise solution that enables development to continue organizationally in less time and less expense than a single namespace for all trading partners.	

NAMESPACES AND 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
- *Namespace declaration and qualification*—the declaration of namespaces in XML instance documents and designation of constructs belonging to those namespaces
- *The W3C Schema Instance namespace*—a set of namespaces specific to the W3C Schema standard
- *Namespace scope*—the range of applicability of namespaces within XML instance documents and methods for altering this range.

XML Instance Document Validation

There are several possible ways that an XML instance document can be associated with a schema for validation purposes:

- The validating system (sending or receiving system) selects the schema based on information that is external to the XML instance document, such as a file name.
- The validating system selects the schema based on information contained in the XML instance document, such as trading partner or transaction version.
- The validating system selects the schema based on its exact location as specified in the XML instance document.

The first and second approaches will not be discussed in this document because they involve concepts that are more pertinent to processing applications than schemas. In the third 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"?>
<ExchangeNetwork:RCRAInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:us:net:exchangenetwork CorrectiveAction.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 "urn:us:net:exchangenetwork" namespace identifier is the target namespace of the CorrectiveAction.xsd schema.

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

```
<?xml version="1.0"?>
<ExchangeNetwork:RCRAInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="CorrectiveAction.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.

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	
Data-centric:	[SD4-35] Data-centric XML instance documents MUST be validated against a schema during processing. [SD4-36] Data-centric XML instance documents SHOULD list the storage location of the schema where the XML instance document validates in the root element. [SD4-37] Data-centric XML instance documents MUST use the schemaLocation construct when listing the storage location of the schema to which the XML instance document validates. [SD4-38] Data-centric XML instance documents MUST NOT use the noNamespaceSchemaLocation construct when listing the storage location of the schema to which the XML instance document validates.
Document-centric:	[SD4-39] Document-centric XML instance documents SHOULD be validated against a schema during processing. [SD4-40] Document-centric XML instance documents SHOULD list the storage location of the schema to which the XML instance document validates in the root element. [SD4-41] Document-centric XML instance documents MUST use the schemaLocation construct when listing the storage location of the schema to which the XML instance document validates. [SD4-42] Document-centric 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 data-centric 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 CorrectiveAction.xsd schema have a namespace prefix of "ExchangeNetwork" in the XML instance document:

```
<?xml version="1.0"?>
    <ExchangeNetwork:RCRAInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:us:net:exchangenetwork CorrectiveAction.xsd"
    xmlns:ExchangeNetwork="http://www.epa.gov/network">
    <ExchangeNetwork:FacilityIdentificationCode>15691</ExchangeNetwork:FacilityIdentificationCode>
    <!—information removed for example purposes—>
</ExchangeNetwork:RCRAInformation>
```

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"
xmlns:ExchangeNetwork="urn:us:net:exchangenetwork"
targetNamespace="urn:us:net:exchangenetwork" 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—
XML Instance Documents

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	
Data-centric:	[SD4-43] Data-centric XML instance documents MUST use namespace qualification for all elements.
Document-centric:	[SD4-44] Document-centric XML instance documents MUST use namespace qualification for all elements and attributes.
Justification	
Although namespace qualification of elements and attributes in an XML instance document can increase verbosity, the ability to easily identify the namespace where an element or attribute belongs (visually or automatically) is very valuable.	

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"?>
    <ExchangeNetwork:RCRAInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ExchangeNetwork="urn:us:net:exchangenetwork">
    <!—information removed for example purposes—>
</ExchangeNetwork:RCRAInformation>
```

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.

The 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	
Data-centric:	[SD4-45] Data-centric XML instance documents MUST declare the W3C Schema Instance namespace when W3C Schema Instance constructs are used. [SD4-46] Data-centric XML instance documents SHOULD use "xsi" as a namespace prefix for all W3C Schema Instance constructs.
Document-centric:	[SD4-47] Document-centric XML instance documents MUST declare the W3C Schema Instance namespace when W3C Schema Instance constructs are used. [SD4-48] Document-centric 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 "urn:us:net:exchangenetwork" namespace is the entire XML instance document:

```
<?xml version="1.0"?>
<ExchangeNetwork:RCRAInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:us:net:exchangenetwork CorrectiveAction.xsd"
xmlns:ExchangeNetwork="urn:us:net:exchangenetwork">
```

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"?>
    <ExchangeNetwork:RCRAInformation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:us:net:exchangenetwork CorrectiveAction.xsd"
    xmlns:ExchangeNetwork="urn:us:net:exchangenetwork">
    <!—information removed for example purposes—>
    <ExchangeNetwork:State>VA</ExchangeNetwork:State>
    <vadeq:State xmlns:vadeq="http://www.state.va.us/xml">Acidic</vadeq:State>
    <!—information removed for example purposes—>
</ExchangeNetwork:RCRAInformation>
```

In this example, the scope of the "http://www.state.va.us/xml" namespace is the vadeq:State element (along with its attributes and subelements, if it contained any). Therefore, the following example would cause an XML processor to generate an error because the vadeq:WaterExtractionDate element appears outside of the scope of the "http://www.state.va.us/xml" namespace:

```
<?xml version="1.0"?>
    <ExchangeNetwork:RCRAInformation
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:us:net:exchangenetwork CorrectiveAction.xsd"
    xmlns:ExchangeNetwork="urn:us:net:exchangenetwork">
    <!—information removed for example purposes—>
    <ExchangeNetwork:State>VA</ExchangeNetwork:State>
    <vadeq:State xmlns:vadeq="http://www.state.va.us/xml">Acidic</vadeq:State>
    <vadeq:WaterExtractionDate>02-02-2002</vadeq:WaterExtractionDate>
    <!—information removed for example purposes—>
</ExchangeNetwork:RCRAInformation>
```

Local Namespace Declarations

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	
Data-centric:	[SD4-49] Data-centric XML instance documents MUST NOT use local namespace declarations.
Document-centric:	[SD4-50] Document-centric XML instance documents SHOULD NOT use local namespace declarations.
Justification	
Although processing efficiencies may be gained through the use of local namespace declarations, the ability to visually identify all namespaces declared in an instance document by examining the root element is more valuable.	

Chapter 5
Schema Configuration and Documentation

This chapter discusses the Exchange Network Schema Configuration Architecture and two related concepts: nested includes and code lists. This chapter also discusses Exchange Network schema versioning and documentation within Exchange Network schemas.

EXCHANGE NETWORK SCHEMA CONFIGURATION ARCHITECTURE

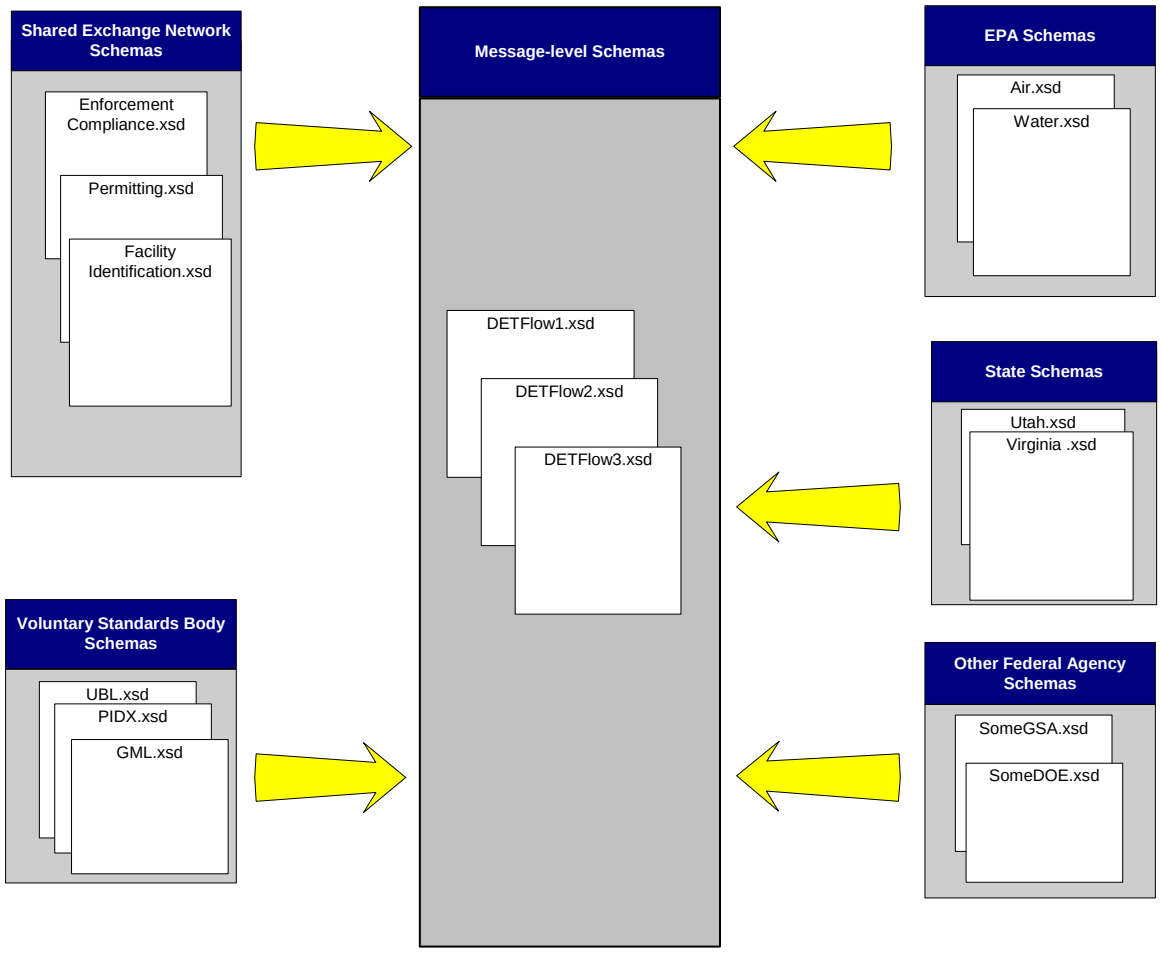
The Exchange Network Schema Configuration Architecture is a flexible modular architecture that uses schemas both within and external to the Exchange Network. Those schemas, each serving a specific purpose, are as follows:

- Message-level schemas
- Shared Exchange Network schemas
- Voluntary standards body schemas
- Functional Area Schema
  - EPA schemas
  - State agency schemas
  - Other federal agency schemas.

The Exchange Network Schema Configuration Architecture is shown in Figure 2.5-1. The following sections describe each type of schema in greater detail and provide recommendations on their use. (Chapter 4 discusses the multiple-namespace configuration for this architecture.)

This document does not address where schemas are stored, but ideally—and at a minimum—the message-level schemas and the shared Exchange Network schemas should reside in the Environmental Network Registry for optimal access by all parties.

Figure 2.5-1. Exchange Network Schema Configuration Architecture



Notes: UBL = Universal Business Language; GML = Geography Markup Language; PIDX = Petroleum Industry Data Exchange.

Message-Level Schemas

Message-level schemas act as central consolidators of other schemas. They externally reference all other types of schemas listed above, but also may contain construct definitions. *Blueprint for a National Environmental Information Exchange Network* explains that data exchange templates (DETs) identify—according to predefined standards—the kind of information that is required or allowable for a particular type of data set. In the Exchange Network Schema Configuration Architecture, message-level schemas are the equivalent of DETs. Therefore, a message-level schema represents all the metadata necessary for a functional data flow.

Message-Level Schema

Pros and Cons	
Advantages:	Use of message-level schemas promotes an efficient, modular configuration architecture.
Disadvantages:	Use of message-level schemas could be considered more complex and more difficult to process due to multiple layers of included or imported schemas.
Rules and Guidelines	
Data-centric:	[SD5-1] Message-level schemas SHOULD be used. [SD5-2] Data-centric message-level schemas SHOULD use a target namespace identifier as identified in the Exchange Network Namespace architecture.
Document-centric:	[SD5-3] Message-level schemas MAY be used.
Justification	
A modular approach to schema configuration is consistent with long-standing software industry best practices.	

Following is the order in which the types of schemas should be selected for optional inclusion in a message-level schema:

1. Voluntary standards body schemas
2. Shared Exchange Network schemas
3. Functional Area Schema: EPA, state and other federal agency schemas.

Voluntary standards body schemas should be examined first to identify candidates for use in the Exchange Network. If no suitable constructs are found, existing federal, state, EPA, and other individual other federal agency schema should be reviewed as reusable sources material for Exchange Network Schema.

Only if all of these sources fail to yield reusable components should new schema be developed. If these new constructs are beneficial for global use and produce no redundancies, they should be added to a shared Exchange Network Schema. Otherwise, they should be placed in an EPA schema.

Shared Exchange Network Schemas

Shared Exchange Network schemas contain constructs that correspond to the final Environmental Data Registry (EDR) standards as well as constructs created by EPAs. Shared Exchange Network schemas are meant to create standard constructs that may be used by multiple message-level schemas. Shared Exchange Network schemas promote interoperability across the Exchange Network and provide a mechanism for program areas to leverage the schema constructs developed by other program areas.

An example of a shared Exchange Network schema is a schema containing user-defined simpleTypes. If all message-level schemas refer to the same shared Exchange Network schema for simpleType reuse, name harmonization will happen automatically. If a name change is made to a simpleType, the message-level schema will inherit those changes.

Shared Exchange Network Schemas

Pros and Cons	
Advantages:	Use of shared Exchange Network schemas promotes reuse within the Exchange Network, generating economies of scale for initial development of data flows, future operations, and maintenance of the schema.
Disadvantages:	A greater level of effort is initially required to develop shared Exchange Network schemas than to develop schemas independently because of the level of collaboration required between schema developers to create consistent, useful schema constructs for Exchange Network use.
Rules and Guidelines	
Data-centric:	[SD5-4] Shared Exchange Network schemas MUST be used when available. [SD5-5] Data-centric shared Exchange Network schemas SHOULD use a target namespace identifier of "urn:us:gov:epa".
Document-centric:	[SD5-6] Shared Exchange Network schemas MUST be used when available.
Justification	
Shared Exchange Network schemas move the Exchange Network toward the goal of harmonized, interoperable DETs. Although a greater level of effort is required initially to develop shared Exchange Network schemas, the benefits of schema construct harmonization outweigh the costs of schema construct duplication. Owners of specific data flows must remain continually aware of other schema development efforts, which will require standard operating procedures for coordination of Exchange Network-wide XML activity. The burden of this coordination has already been reduced through the creation of the Exchange Network Technical Resources Group.	

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).

The Exchange Network Schema Configuration Architecture encourages the use of voluntary standards body schemas from horizontal efforts. However, it is impractical to mandate the use of such schemas at this time because, currently, no such schemas are widely accepted.

Voluntary Standards Body Schemas

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	
Data-centric:	[SD5-7] Appropriate voluntary standard body schemas SHOULD be adopted, when appropriate.
Document-centric:	[SD5-8] 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." [a] The use of Voluntary Standard Body Schemas is in accordance with OMB Circular A-119. [b]	

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

[b] 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. Available from <http://www.whitehouse.gov/omb/circulars/a119/a119.html#6>.

Functional Area Schemas

Functional area schemas are schemas developed by a specific organization to meet to support a narrow functional area or a group of stakeholders less than the total body of Exchange Network Participants. Such schemas should be considered and perhaps revised to support the wider body.

EPA SCHEMAS

With EPA schemas (either agency-wide or program area specific), the EPA (possibly in conjunction with selected states) can develop specific schema constructs. 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 EPA constructs because

- the constructs represent information specific to that state or EPA program, or

- it may be unduly burdensome at times for a program area to harmonize schema constructs across the Exchange Network.

Although specific functional areas are not defined in this guide, a finite number of functional areas should be identified to ensure that EPA functional area schemas do not proliferate.

EPA Schemas

Pros and Cons	
Advantages:	Use of EPA schemas allows program areas to develop functional area-specific schema constructs.
Disadvantages:	Use of functional area schemas may promote duplication of schema constructs that are already available in another schema, such as a shared Exchange Network schema.
Rules and Guidelines	
Data-centric:	[SD5-9] Functional area schemas MAY be used. [a]
	[SD5-10] Data-centric Exchange Network schemas SHOULD use a target namespace identifier as identified in the Exchange Network Namespace architecture. [a]
Document-centric:	[SD5-11] Functional area schemas MAY be used. [a]
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.	

[a] Logistics Management Institute, *XML Schema Namespace and Versioning Strategy for the Environmental Information Exchange Network*, Report EP211L1, Mark Crawford and Jessica Glace, December 2002.

STATE AND OTHER FEDERAL AGENCY SCHEMAS

It is anticipated that various individual federal agency and individual state government schemas will be available in the future and may be appropriate for use by the Exchange Network. There are also current efforts to guide XML development at the federal level. The Federal CIO XML Working Group has prepared *Federal XML Developer's Guide*,[1] based closely on the Department of the Navy's guide.[2] Use of the *Federal XML Developer's Guide* will ensure that the Exchange Network's schema development adheres to the emerging standards of the federal government.

[1] U.S. Federal CIO Council, Architecture and Infrastructure Committee XML Working Group, *Draft Federal XML Developer's Guide*, April 2002.

[2] Department of the Navy, DON XML Working Group, *DON XML Developer's Guide, Version 1.1*, May 2002.

State and Other Federal Agency Schemas

Pros and Cons	
Advantages:	Use of federal and state government schemas promotes interoperability of data exchange within the U.S. government.
Disadvantages:	It is possible that federal and state government schemas may not follow the guidelines for schema development, as set forth by the Federal CIO XML Working Group or the guidelines set forth in this document. Therefore, use of federal and state government schemas may introduce inconsistent constructs in Exchange Network schemas.
Rules and Guidelines	
Data-centric:	[SD5-12] Federal and state government schemas MAY be used if they are consistent with the guidelines for schema development as set forth by the Federal CIO XML Working Group or those set forth in this document.
Document-centric:	[SD5-13] Federal and state government schemas MAY be used if they are consistent with the guidelines for schema development as set forth by the Federal CIO XML Working Group or those set forth in this document.
Justification	
Use of federal and state government schemas promotes interoperability of data exchange within the U.S. government. By using federal and state government schemas that are consistent with the guidelines noted above, the risk of introducing inconsistent constructs in Exchange Network schemas is eliminated.	

NESTED INCLUDES

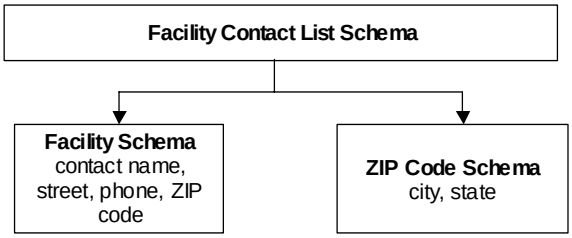
To achieve the recommended Exchange Network Schema Configuration Architecture, “nested includes” or “nested imports” may be necessary. A discussion of nested includes follows.

Nested and Single 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.

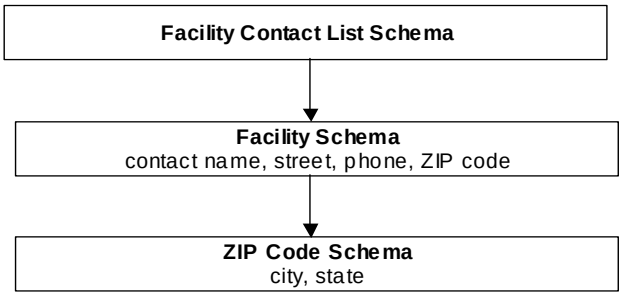
Figure 2.5-2 illustrates a single include. In this example, the facility contact list schema references a facility schema to get a facility’s contact name, street address, telephone number, and Zip Code. It then references a Zip Code schema to retrieve the city and state for the facility’s Zip Code.

Figure 2.5-2. Single Include Example



A nested include is illustrated in Figure 2.5-3. As with the single include example above, a facility contact list schema references the facility schema to get the contact, street address, telephone number, city, state, and Zip Code information. However, instead of the facility contact list schema referencing the Zip Code schema, the facility schema retrieves the city and state from the Zip Code schema for the facility contact list schema.

Figure 2.5-3. Nested Include Example



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.

Number of Nested Includes

Message schemas may contain one shared construct, a set of similar shared constructs, or all constructs used by an application. An application using one schema for each shared construct (e.g., every complexType resides in a distinct file) may experience heavy maintenance and tracking burdens, while an application with one message schema for all shared constructs may face readability and performance issues.

The recommended method of handling constructs for Exchange Network shared schemas is to group similar constructs into one schema to reduce the number of nested schemas referenced by the message-level schemas.

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	
Data-centric:	[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.
Document-centric:	[SD5-16] Exchange Network schemas SHOULD group like constructs into one schema.
	[SD5-17] 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. Run-time schemas, based on the design schemas, may need to be constructed to make processing more efficient. Storage of similar shared complex or simpleType constructs in one message schema is a more logical choice for developing schemas than accessing constructs through multiple layers of schemas.	

CODE LISTS

Several strategies exist for handling lists of values or "code lists." The main points and recommended method for handling them are summarized below.[3]

In XSD, the values contained in the list are in reality tokens for more detailed values. For example, the EPA maintains a B_Type_Code list that supports field number two in the B_Geographic_Area file layout for the federal version of the Safe Drinking Water Information System (SDWIS) application. This list is, in reality, a list of values that have assigned tokens. Multiple namespaced types are used for handling code lists in Exchange Network schemas.

[3] For a detailed discussion, see Logistics Management Institute, *XML Enumeration and Code Lists for the Environmental Information Exchange Network*, LMI Report EP211L2, Mark Crawford, Jessica Glace, and Alison Kittle, December 2002.

Multiple namespaced types require the use of an element dedicated to containing codes from a particular code list bound to a unique type that is qualified with an external namespace. An instance document would look like this:

```
<GeographicAreaCode>
<SDWISGeographicAreaCodeContent>ARV</SDWISGeographicAreaCodeContent>
</GeographicAreaCode>
```

In the above example, SDWIS Geographic Area Permitted Value List is defined as a code list with a token of "SDWISGeographicAreaCode". The "GeographicAreaCode" element does not contain the value, but a subelement of "SDWISGeographicAreaCodeContent".

In this instance, the "SDWISGeographicAreaCodeContent" value space is populated with a value of "ARV" which is a token for the value "Alaskan Remote Village." The part of the schema code that would define this looks as follows:

```
<xsd:element name="GeographicAreaCode" type="EPA103:GeographicAreaCodeType">
<xsd:element name="SDWISGeographicAreaCodeContent"
 type="EPA103:SDWISGeographicAreaCodeContentType">
<xsd:complexType name="GeographicAreaCodeType">
<xsd:choice>
<xsd:element name="GeographicAreaCode" type="EPA103:GeographicAreaCodeType"/>
<xsd:element name="XXXCode" type="xxx:CodeType"/>
<xsd:element name="YYYCode" type="yyy:CodeType"/>
</xsd:choice>
</xsd:complexType>
```

The namespace qualified the location of the elements and types. Three options are provided for the associated elements of the complex type GeographicAreaCodeType. This allows for the selective use of either the EPA or state code list at runtime, depending on the element conveyed.

Using this technique, schema modules maintain code lists separately from the message-level schema. All code lists used must be published in a standard module available on the Internet so that code list contents are always available for validation from the authoritative source at runtime. Code lists not published by code list owners that the Exchange Network uses will need to be published by the EPA. Code list owners will be responsible for maintaining their code lists.

Code Lists

Pros and Cons	
Advantages:	This technique provides strong semantic clarity, strong interoperability, and simple, externalized maintenance. It fully supports XML processor validation and provides easy readability.
Disadvantages:	This technique requires cooperation of external activities and an initial effort to define and publish schema modules.
Data-centric:	[SD5-18] Exchange Network schemas SHOULD support code lists through multiple namespaced types.
Document-centric:	[SD5-19] Exchange Network schemas SHOULD support code lists through multiple namespaced types.
Justification	
The code lists method presents the best solution because it provides the ability to minimize maintenance workload while achieving access to authoritative source code lists in real time. This ability supports runtime parser validation.	

EXCHANGE NETWORK SCHEMA VERSIONING

A consistent version control strategy based on current technology needs to be applied to Exchange Network schemas. This section summarizes versioning at the schema root, the root instance, and the targetNamespace levels.[4]

The numbering used in schema versioning should include both a major version component and a minor version component, such as Version 1.2 ("1" is the major version component, and "2" is the minor version component). A major change may not be backward compatible, but a minor change must be. For example, a minor change might add optional elements, annotations, or anything in which an instance created against an old Schema 1.0 can still be valid against 1.x. A major change might add a mandatory element complex type, or anything in which an instance created against an old Schema 1.0 may not be valid against 2.0.

Built-In 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:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        xmlns:ExchangeNetwork="http://www.ExchangeNetwork.gov/XML"
version="1.2">
```

[4] Logistics Management Institute, *XML Schema Namespace and Versioning Strategy for the Environmental Information Exchange Network*, Report EP211L1, Mark Crawford and Jessica Glace, December 2002.

Built-In 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	
Data-centric:	[SD5-20] Data-centric schemas MUST include a version number using the W3C Schema version attribute. [SD5-21] The version number MUST include both a major version component and a minor version component. [SD5-22] Data-centric schemas SHOULD include a version number in their filename.
Document-centric:	[SD5-23] Document-centric schemas MUST include a version number using the W3C Schema version attribute. [SD5-24] The version number MUST include both a major version component and a minor version component. [SD5-25] Document-centric schemas SHOULD include a version number in their filename.
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.	

User-Defined Version Attribute on Instance Root

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:

```
<xs:element name="AsbestosPublicState" type="EPAMetadata"/>
<xs:complexType name="EPAMetadata"/>
<xs:attribute name="schemaVersion" type="xs:decimal" use="required"/>
</xs:complexType>
```

In the XML document, the versioning would look like this:

```
<AsbestosPublicState schemaVersion="1.0" >
…
</AsbestosPublicState>
```

User-Defined Version Attribute on Instance Root

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	
Data-centric:	[SD5-26] Data-centric schemas MUST define a schema version attribute for use on the instance root.
Document-centric:	[SD5-27] Document-centric schemas MUST define a schema version attribute for use on the instance root.
Justification	
Use of a version attribute in the instance root ensures matching the instance with the appropriate major version of a schema. It provides additional information for efficiently processing information and reducing errors caused by assumptions about which schema should be used for validating an instance. Pattern matching can be used to eliminate the need for updates when a minor change is implemented.	

EXCHANGE NETWORK SCHEMA DOCUMENTATION

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

One important advancement for the W3C Schema standard over DTDs is the capability to create machine-processable comments. In DTDs, comments were marked as follows:

```
<!—this is a comment—!>
```

This was an adequate technique for DTDs, but it does not allow for machine processing of comments. The W3C Schema documentation element is very useful

when documenting schemas. Following is an example of the documentation element:

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

Although the W3C Schema standard still supports DTD-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 subelement of the W3C Schema annotation element and is used to document the function of a specific element:

```
<xsd:element name="epa: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	
Data-centric:	[SD5-28] Data-centric schemas SHOULD include schema construct documentation.
	[SD5-29] Data-centric schemas SHOULD use the documentation element for schema construct documentation.
	[SD5-30] Data-centric schemas MAY use DTD-style comments for comments pertaining to the structure of the schema.
Document-centric:	[SD5-31] Document-centric schemas SHOULD include schema construct documentation.
	[SD5-32] Document-centric schemas SHOULD use the documentation element for schema construct documentation.
	[SD5-33] Document-centric schemas MAY use DTD-style comments for comments pertaining to the structure of the schema.

Schema Construct Documentation

Justification
DTD-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.

Table 2.5-1. Header Documentation Information

Header item name	Description
Schema Name	The schema file
Current Version Available At	The URL where the schema is located, if it is URL accessable
Description	Plain text description of the information described by the schema
Application	The name and version of the application that produces XML instance documents that conform to the schema
Developed by	State or “Environmental Protection Agency,” followed by name of the organization that developed the schema
Point of Contact	Person to contact with questions about the schema
Change History or a URI reference to the change history (optional)	A history of the changes to the schema. Each entry should include the ➢ change number, ➢ schema version, ➢ change date, ➢ change description, and ➢ XML processor (vendor and version) used to validate schema, or the change history file name and location.

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

```
<xsd:documentation>Schema Name: Enforcement Compliance Schema</xsd:documentation>
<xsd:documentation>Description: This schema contains…</xsd:documentation>
```

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	
Data-centric:	[SD5-34] Data-centric schemas MUST include schema header documentation.
Document-centric:	[SD5-35] Document-centric schemas SHOULD include schema header documentation.
	[SD5-36] Document-centric schemas SHOULD use the documentation element for schema construct 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.	

Chapter 6
Information Association and Uniqueness

Information association allows XML processors or processing applications to link information items. Uniqueness involves the nonduplication 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.

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.

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:

```
<Facilities>
    <FacilitySiteDetails facilityIdentificationCode="A15849">
        <FacilityAddressDetails>
            <!—information removed for example purposes—>
        </FacilityAddressDetails>
        </FacilitySiteDetails>
    </Facilities>
    <Submitters>
    <SubmitterDetails submitterFacilityIdentificationCode="A15849">
        <SubmitterNameDetails>
            <!—information removed for example purposes—>
        </SubmitterNameDetails>
    </SubmitterDetails>
</Submitters>
```

In the above example, the facilityIdentificationCode attribute is of datatype "ID," while the submitterFacilityIdentificationCode 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 submitterFacilityIdentificationCode 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. (In the example above, if there were no facilityIdentificationCode attributes with a value of A15849, but another attribute that was completely unrelated to the association coincidentally contained the value of A15849, an error would *not* result.) However, a processing application could recognize associations between ID and IDREF values during its processing of an XML instance document.

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 data-centric 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.

ID/IDREF Technique

Rules and Guidelines	
Data-centric:	[SD6-1] Data-centric schemas MUST NOT use the ID/IDREF technique for information association.
Document-centric:	[SD6-2] Document-centric 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 superceded within the W3C Schema standard by the new KEY/KEYREF technique, which is discussed below.	

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

The above declaration stipulates that the facilityIdentificationCode attribute (the attribute listed with the field element) is used for information association in an

XML instance document. In addition, the value of this attribute must be unique within all FacilitySiteDetails elements that appear under the Facilities element (the XPath expression listed with the selector element). This means a facility identification code can be duplicated within an XML instance document, but not within the Facilities content model.

Similarly, the following KEYREF construct declares a foreign key for the submitterFacilityIdentificationCode 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 submitterFacilityIdentificationCode attribute must be unique within all SubmitterDetails 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 SubmitterFacilityIdentificationCode value in an XML instance document.

KEY/KEYREF Technique

Pros and Cons	
Advantages:	Use of attributes is not required.
	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	
Data-centric:	[SD6-3] Data-centric 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.
Document-centric:	[SD6-6] Document-centric schemas SHOULD use the KEY/KEYREF technique for information association.
	[SD6-7] Extreme caution SHOULD be applied when writing an XPath expression in a selector element to ensure it specifies the intended range.
	[SD6-8] Special attention SHOULD be paid to the restrictions on KEY and KEYREF declaration names given above.

KEY/KEYREF Technique

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 data-centric schemas prohibit attributes.

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="Facilities/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	
Data-centric:	[SD6-9] Data-centric 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.

KEY Technique

Document-centric:	[SD6-12] Document-centric 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. [SD6-13] Extreme caution SHOULD be applied when writing an XPath expression in a selector element to ensure it specifies the intended range. [SD6-14] 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.	

XLink/XPointer Technique

The XLink/XPointer technique uses two relatively new W3C standards to associate information in an XML instance document. 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 SubmitterDetails element:

```
<Facilities>
    <FacilitySiteDetails FacilityIdentificationCode="A15849">
        <FacilityAddressDetails>
            <!—information removed for example purposes—>
        </FacilityAddressDetails>
    </FacilitySiteDetails>
</Facilities>
<Submitters>
    <SubmitterDetails>
        <SubmitterNameDetails>
            <!—information removed for example purposes—>
        </SubmitterNameDetails>
        <SubmitterIdentificationCode>9187234</SubmitterIdentificationCode>
        <SubmitterFacilityIdentificationCode xlink:type="simple" xlink:href="#A15849"/>
```

```
    </SubmitterDetails>
</Submitters>
```

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 submitters with their facilities:

```
<xlink:extended role="Link Submitters to Facilities" title="Links">
    <xlink:locator href="#9187234" role="Submitter" label="Submitter 9187234">
    <xlink:locator href="#147341" role="Submitter" label="Submitter 147341">
    <xlink:locator href="#A15849" role="Facility" label="Facility A15849">
    <xlink:arc from="Submitter 9187234" to="Facility A54346" arcrole="Submitter Works For">
    <xlink:arc from="Submitter 147341" to="Facility A54346" arcrole="Submitter Works For">
</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.

It is also possible to specify the extended links shown above in a separate XML instance document. Suppose the XML instance document containing submitter and facility information were in Submissions.xml. The earlier extended link declaration would change only in that the href attributes began with the name of the XML instance document file and were followed by the ID values listed above:

```
<xlink:extended role="Link Submitters to Facilities" title="Links">
    <xlink:locator href="Submissions.xml#9187234" role="Submitter" label="Submitter 9187234">
    <xlink:locator href="Submissions.xml#147341" role="Submitter" label="Submitter 147341">
    <xlink:locator href="Submissions.xml#A15849" role="Facility" label="Facility A15849">
    <xlink:arc from="Submitter 9187234" to="Facility A54346" arcrole="Submitter Works For">
    <xlink:arc from="Submitter 147341" to="Facility A54346" arcrole="Submitter Works For">
</xlink:extended>
```

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. Because the XLink and XPointer standards are both new (XLink became a W3C Recommendation in June 2001, and XPointer is currently a Candidate Recommendation), there is currently very little XML processor support for them.

Rules and Guidelines	
Data-centric:	[SD6-15] Data-centric schemas MUST NOT use the XLink/XPointer technique for information association.
Document-centric:	[SD6-16] Document-centric 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.	

UNIQUENESS

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="Facilities/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	
Data-centric:	[SD6-17] Data-centric 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.
Document-centric:	[SD6-20] Document-centric 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. [SD6-21] Extreme caution SHOULD be applied when writing an XPath expression in a selector element to ensure it specifies the intended range. [SD6-22] Special attention SHOULD be paid to the restrictions on UNIQUE declaration names given above.
Justification	
The UNIQUE technique is extremely useful for enforcing uniqueness in an XML instance document, and its use is therefore recommended.	

Chapter 7
Advanced W3C Schema 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.
- *Code lists*—the handling of lists of values through multiple namespaced types.
- *Supplemental instructions*—the use of supplemental instructions in a schema to pass them to a processing application.

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:RangeOneToTenType"/>
    <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="RangeThreeToTenType">
    <xsd:restriction base="xsd:RangeOneToTenType">
        <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. A processing application can perform the same validations on an XML instance document that are enforced by facets in a schema (such as a range of allowed values). Additional effort is required in such situations to ensure that changes to the processing application validations and schema validations remain in sync.
Rules and Guidelines	
Data-centric:	[SD7-1] Data-centric schemas SHOULD NOT use simple datatype restriction when a data standard or an approved schema exists. [SD7-2] Data-centric schemas MUST use global simple datatypes. [SD7-3] Data-centric schemas MUST NOT use local simple datatypes.
Document-centric:	[SD7-4] Document-centric schemas MAY use simple datatype restriction. [SD7-5] Document-centric schemas MAY use global simple datatypes. [SD7-6] Document-centric schemas MUST NOT use local simple datatypes.

Simple Datatype Restriction

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 list datatype is defined based on the NMTOKEN datatype (a W3C Schema built-in datatype):

```
<xsd:element name="ReportMonthsList" type= "MonthListType">
<xsd:simpleType name="MonthListType">
    <xsd:list itemType="xsd:NMTOKENS"/>
</xsd:simpleType>
```

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

```
<ReportMonthsList>February March September</ReportMonthsList>
```

Although the intent of the ReportMonthsList element is to hold month names, an XML processor cannot verify that the contents of the element are valid month names, given the above declarations.

List Technique

Pros and Cons	
Advantages:	The list technique allows datatypes to be defined to represent a whitespace-delimited list of values.
Disadvantages:	An XML processor cannot validate the contents of a list.
Rules and Guidelines	
Data-centric:	[SD 7-7] Data-centric schemas MAY use the list technique.
	[SD 7-8] Data-centric 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).
Document-centric:	[SD 7-9] Document-centric schemas MAY use the list technique.
	[SD 7-10] Document-centric 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).

List Technique

Justification
The usefulness of this technique is limited because an XML processor cannot validate the contents of a list. In addition, if the values within a list contain spaces themselves, each component of the value (between spaces) will be considered a separate value in the list. This may yield incorrect results from a processing application.

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	
Data-centric:	[SD7-11] Data-centric schemas MAY use the union technique.
Document-centric:	[SD7-12] Document-centric 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 FacilityAddressDetailsType datatype is restricted to contain a lower number of maximum occurrences for the FacilityStreetName element (all constructs from the base datatype must be listed in the restriction):

```
<xsd:complexType name="FacilityAddressDetailsType">
    <xsd:sequence>
        <xsd:element name="FacilityStreetAddress"type="xsd:string"maxOccurs="3"/>
        <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>
<xsd:complexType name="FacilityAddressDetailsTypeAbbreviated">
    <xsd:complexContent>
        <xsd:restriction base="FacilityAddressDetailsType">
            <xsd:sequence>
            <xsd:element name="FacilityStreetAddress" type="xsd:string" maxOccurs="2"/>
            <xsd:element name="FacilityCity"type="xsd:string"/>
            <xsd:element name="FacilityState"type="xsd:string"/>
            <xsd:element name="FacilityZipCode"type="xsd:string"/>
            </xsd:sequence>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

Because the FacilityAddressDetailsTypeAbbreviated complex datatype is derived from the FacilityAddressDetailsType datatype, any element declared as part of the FacilityAddressDetailsTypeAbbreviated datatype can appear anywhere in an XML instance document in which the base datatype is expected.

The "xsi:type" W3C Schema instance construct must be used to indicate to an XML processor exactly which datatype (base or derived) applies:

```
<FacilityAddressDetails xsi:type="FacilityAddressDetailsTypeAbbreviated">
    <FacilityStreetAddress>15 Main St.</FacilityStreetAddress>
    <!—information removed for example purposes—>
</FacilityAddressDetails>
```

A base datatype can also be restricted through the removal of constructs; however, the construct must be declared as optional in the base datatype. With

this type of restriction, all constructs from the base datatype are repeated in the restriction, with the exception of the construct being removed.

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	
Data-centric:	[SD7-13] Data-centric schemas MAY use complex datatype restriction.
Document-centric:	[SD7-14] Document-centric 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 FacilityAddressDetailsType datatype extends to contain an additional FacilityRegion element (only the additional element needs to be listed in the extension):

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

```
        </xsd:complexContent>
    </xsd:complexType>
```

As with the complex datatype restriction example, any element declared as part of the FacilityAddressDetailsTypeExtended datatype can appear anywhere in an XML instance document that the base datatype is expected, and the "xsi:type" construct must be used to indicate to an XML processor exactly which datatype applies.

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	
Data-centric:	[SD7-15] Data-centric schemas MAY use complex datatype extension.
Document-centric:	[SD7-16] Document-centric 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	
Data-centric:	[SD7-17] Data-centric schemas MAY use the final attribute derivation.
Document-centric:	[SD7-18] Document-centric schemas MAY prohibit complex datatype 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 data-centric schemas. Because duplication of schema constructs is not as critical an issue for document-centric schemas as for data-centric 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 Datatypes

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	
Data-centric:	[SD7-19] Data-centric schemas MAY use the block attribute.
Document-centric:	[SD7-20] Document-centric schemas SHOULD NOT prohibit use of derived complex datatypes.
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.	

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	
Data-centric:	[SD7-21] Data-centric schemas MUST NOT use abstract datatypes.
Document-centric:	[SD7-22] Document-centric 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	
Data-centric:	[SD7-23] Data-centric schemas MUST NOT use wildcards.
Document-centric:	[SD7-24] Document-centric 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.	

DEFAULT AND FIXED ELEMENT AND ATTRIBUTE VALUES

One important advancement for the W3C Schema standard over DTDs is the capability to declare default and fixed element values (with DTDs, default and fixed values were allowed only for attributes). This section discusses the new W3C Schema features of 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	
Data-centric:	[SD7-25] Data-centric schemas SHOULD NOT use default element values.
Document-centric:	[SD7-26] Document-centric schemas MAY 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. However, in document-centric scenarios, the reduction of burden on an XML instance document author through the use of default element values outweighs the risks. For example, a default element value may be used to insert the text of a standard disclaimer into an XML document instance, thereby eliminating the burden on the XML instance document author of having to enter the text each time it is required.	

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	
Data-centric:	[SD7-27] Data-centric schemas SHOULD NOT use fixed element values.
Document-centric:	[SD7-28] Document-centric schemas MAY 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. However, in document-centric scenarios, the reduction of burden on an XML instance document author through the use of fixed element values outweighs the risks. For example, a fixed element value may be used to insert the text of a standard disclaimer into an XML document instance, thereby eliminating the burden on the XML instance document author of having to enter the text each time it is required.	

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	
Data-centric:	[SD7-29] Data-centric schemas SHOULD NOT use default attribute values.
Document-centric:	[SD7-30] Document-centric schemas MAY 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. However, in document-centric scenarios, the reduction of burden on an XML instance document author through the use of default attribute values outweighs the risks.	

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	
Data-centric:	[SD7-31] Data-centric schemas SHOULD NOT use fixed attribute values.
Document-centric:	[SD7-32] Document-centric schemas MAY 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. However, in document-centric scenarios, the reduction of burden on an XML instance document author through the use of fixed attribute values outweighs the risks.	

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	
Data-centric:	[SD7-33] Data-centric schemas MUST NOT use substitution groups.
Document-centric:	[SD7-34] Document-centric schemas SHOULD NOT use substitution groups. [SD7-35] Document-centric schemas MAY "block" use of substitution groups.
Justification	
Harmonization is the key to interoperable data exchange, and use of substitution groups moves away from harmonization. Because harmonization is not as critical an issue for document-centric schemas as for data-centric schemas, use of substitution groups is permissible for document-centric schemas.	

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 Schema 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 very useful for passing processing commands or other supplemental information to a processing application.
Disadvantages:	XML processors are not yet mature enough to be able to properly handle this technique.
Rules and Guidelines	
Data-centric:	[SD7-36] Data-centric schemas MUST NOT use the appinfo element.
Document-centric:	[SD7-37] Document-centric schemas MUST NOT use the appinfo element.
Justification	
The use of the appinfo element is considered very risky at this time because certain XML processors may not support its use. 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). With DTDs, it was possible to directly associate a notation with an attribute. In the W3C Schema

standard, however, 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="gif" 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="gif"/>
                        </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	
Data-centric:	[SD7-38] Data-centric schemas MUST NOT use notations.
Document-centric:	[SD7-39] Document-centric schemas MUST NOT use notations.
Justification	
The use of notations is considered very risky because there is no guarantee that an XML processor will properly handle a notation declaration.	

Appendix A
Summary of XML Rules

This appendix summarizes the 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.

The rules contain certain words that have an explicit meaning. Those words, defined in Request for Comments 2119 issued by the Internet Engineering Task Force, are as follows:[1]

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 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 that does include a particular option MUST be prepared to

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

interoperate with another implementation that does not include the option (except, of course, for the feature the option provides).

All design rules are normative. Design rules are identified through a prefix of [XXc-nn].

- The value "XX" is a prefix to categorize the type of rule, where XX corresponds to a particular section; GD indicates a general design rule (Section 1), and SD indicates a schema design rule (Section 2).
- The value "c" indicates the chapter where the rule is located.
- The value "nn" indicates the sequential number of the rule.

For example, the rule identifier [SD6-22] identifies the 22nd rule in Chapter 6 of Section 2, *Schema Design Rules.*

SECTION 1—GENERAL DESIGN RULES

File Naming Convention—Schema

	[GD1-1] Schemas and style sheets MUST follow a four part, hierarchical naming convention, based on responsible party, data flow, root, and version (for message-level schemas) or responsible party, data flow or CRM, Major Data Group and version (for shared schemas). [GD1-2] File names MUST NOT use abbreviations unless their meaning is beyond question (EPA, GSA, FBI). [GD1-3] Message-level schemas SHOULD have their versions changed when a referenced external modular schema is updated.

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. [GD1-9] Each message set should correspond to a business process model or models in the ebXML catalog of business processes. [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 data types.

[GD1-15] EPA messages shall reuse registered data types 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.

TAG Structure

[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.

Tag Name Content

[GD3-8] Element, attribute, and data type tag names SHOULD be unique.

[GD3-9] Element tag names MUST be extracted from the Environmental Data Registry (EDR) where possible.

[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.
E*xample: <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. *Example: <Goods/>* [GD3-18] Non-letter characters MUST only be used if required by language rules. [GD2-19] Tag names MUST only contain verbs, nouns and adjectives (no words like “and”, “of”, “the”).

SECTION 2—SCHEMA DESIGN RULES

Topic	Data-centric	Document-centric
Simple Datatypes	[SD2–1] Data-centric schemas MUST use simple datatypes to the maximum extent possible.	[SD2–2] Document-centric schemas SHOULD use simple datatypes.
Global Complex Datatypes	[SD2-3] Data-centric schemas that employ complex datatypes MUST define the complex datatypes as global.	[SD2-4] Document-centric schemas that employ complex datatypes SHOULD define the complex datatypes as global.
Local Complex Datatypes	[SD2–5] Data-centric schemas SHOULD NOT use local complex datatypes.	[SD2-6] Document-centric schemas MAY use local complex datatypes.
Global Elements	[SD3-1] Data-centric schemas MUST use global elements.	[SD3-2] Document-centric schemas SHOULD use global elements.
Local Elements	[SD3-3] Data-centric schemas SHOULD NOT use local elements.	[SD3-4] Document-centric schemas MAY use global elements.
Occurrence Indicators	[SD3-5] Data-centric schemas SHOULD use occurrence indicators. [SD3-6] Data-centric schemas SHOULD NOT use occurrence indicators when the required values are the default values.	SD3-7] Document-centric schemas SHOULD use occurrence indicators. SD3-8] Document-centric schemas SHOULD NOT use occurrence indicators when the required values are the default values.
Attributes (General)	[SD3-9] Data-centric schemas MUST NOT use attributes in place of data elements. [SD3-10] Data-centric schemas MAY use attributes for metadata.	[SD3-11] Document-centric schemas MAY use attributes.
Global Attributes	[SD3-12] Data-centric schemas MUST NOT use global attributes in place of data elements [SD3-13] Data-centric schemas MAY use global attributes for metadata.	[SD3-14] Document-centric schemas MAY use global attributes.
Local Attributes	[SD3-15] Data-centric schemas MUST NOT use local attributes in place of data elements. [SD3-16] Data-centric schemas MAY use local attributes for metadata.	[SD3-17] Document-centric schemas MAY use local attributes.
“use” Indicator	[SD3-18] Data-centric schemas SHOULD use the “use” indicator. [SD3-19] Data-centric schemas SHOULD NOT use the “use” indicator when the required value is the default value.	[SD3-20] Document-centric schemas SHOULD use the “use” indicator. [SD3-21] Document-centric schemas SHOULD NOT use the “use” indicator when the required value is the default value.

Topic	Data-centric	Document-centric
“sequence” Compositor	[SD3-22] Data-centric schemas SHOULD use the “sequence” compositor.	[SD3-23] Document-centric schemas SHOULD use the “sequence” compositor.
“choice” Compositor	[SD3-24] Data-centric schemas SHOULD use the “choice” compositor.	[SD3-25] Document-centric schemas SHOULD use the “choice” compositor.
“all” Compositor	[SD3-26] Data-centric schemas MUST NOT use the “all” compositor.	[SD3-27] Document-centric schemas SHOULD use the “all” compositor.
Model Groups	[SD3-28] Data-centric schemas MAY use model groups.	[SD3-29] Document-centric schemas MAY use model groups.
Attribute Groups	[SD3-30] Data-centric schemas MUST NOT use attribute groups in place of data elements. [SD3-31] Data-centric schemas MAY use attribute groups for metadata.	[SD3-32] Document-centric schemas MAY use attribute groups.
Namespace Declaration and Qualification—Schemas	[SD4-1] Data-centric schemas MUST use namespaces. [SD4-2] Data-centric schemas MUST use namespace qualification for all schema constructs.	[SD4-3] Document-centric schemas MUST use namespaces. [SD4-4] Document-centric schemas MUST use namespace qualification for all schema constructs.
W3C Schema Namespace	[SD4-5] Data-centric schemas MUST declare the W3C Schema namespace. [SD4-6] Data-centric schemas MUST use namespace qualification for all W3C Schema constructs. [SD4-7] Data-centric schemas SHOULD use “xsd” as a namespace prefix for all W3C Schema constructs.	[SD4-8] Document-centric schemas MUST declare the W3C Schema namespace. [SD4-9] Document-centric schemas MUST use namespace qualification for all W3C Schema constructs. [SD4-10] Document-centric schemas SHOULD use “xsd” as a namespace prefix for all W3C Schema constructs.
The W3C Schema Datatypes Namespace	[SD4-11] Data-centric schemas SHOULD NOT declare the W3C Schema Datatypes namespace.	[SD4-12] Document-centric schemas SHOULD NOT declare the W3C Schema Datatypes namespace.
Target Namespaces	[SD4-13] Data-centric schemas MUST use target namespaces.	[SD4-14] Document-centric schemas MUST use target namespaces.
External Schema References	[SD4-15] Data-centric schemas SHOULD reference external schemas. SD4-16] Data-centric schemas MAY use the include construct. [SD4-17] Data-centric schemas MAY use the import construct.	[SD4-18] Document-centric schemas MAY reference external schemas. [SD4-19] Document-centric schemas MAY use the include construct. [SD4-20] Document-centric schemas MAY use the import construct.
Single/Multiple Namespaces	[SD4-21] Data-centric schemas SHOULD use a multiple-namespace configuration.	[SD4-22] Document -centric schemas SHOULD use a multiple-namespace configuration.
Default Namespaces	[SD4-23] Data-centric schemas MUST NOT use default namespaces.	[SD4-24] Document-centric schemas MUST NOT use default namespaces.

Topic	Data-centric	Document-centric
Namespaces and Attributes	[SD4-25] Data-centric schemas MUST use namespace qualification for all attributes.	[SD4-26] Document-centric schemas MUST use namespace qualification for all attributes.
Multiple Namespaces	[SD4-27] Exchange Network schemas MAY use multiple namespaces. [SD4-28] Exchange Network schemas MUST use urn:us:net:exchangenetwork as the target namespace. [SD4-29] EPA schemas MUST use urn:us:gov:epa as the target namespace. [SD4-30] Each state MAY have one unique namespace for use in Network exchanges.	[SD4-31] Exchange Network schemas MAY use multiple namespaces. [SD4-32] Exchange Network schemas MUST use urn:us:net:exchangenetwork as the target namespace. [SD4-33] EPA schemas MUST use urn:us:gov:epa as the target namespace. [SD4-34] Each state MAY have one unique namespace for use in Network exchanges.
XML Instance Document Validation	[SD4-35] Data-centric XML instance documents MUST be validated against a schema during processing. [SD4-36] Data-centric XML instance documents SHOULD list the storage location of the schema where the XML instance document validates in the root element. [SD4-37] Data-centric XML instance documents MUST use the schemaLocation construct when listing the storage location of the schema to which the XML instance document validates. [SD4-38] Data-centric XML instance documents MUST NOT use the noNamespaceSchemaLocation construct when listing the storage location of the schema to which the XML instance document validates.	[SD4-39] Document-centric XML instance documents SHOULD be validated against a schema during processing. [SD4-40] Document-centric XML instance documents SHOULD list the storage location of the schema to which the XML instance document validates in the root element. [SD4-41] Document-centric XML instance documents MUST use the schemaLocation construct when listing the storage location of the schema to which the XML instance document validates. [SD4-42] Document-centric XML instance documents MUST NOT use the noNamespaceSchemaLocation construct when listing the storage location of the schema to which the XML instance document validates.
Namespace Declaration and Qualification—XML Instance Documents	[SD4-43] Data-centric XML instance documents MUST use namespace qualification for all elements.	[SD4-44] Document-centric XML instance documents MUST use namespace qualification for all elements and attributes.
The W3C Schema Instance Namespace	[SD4-45] Data-centric XML instance documents MUST declare the W3C Schema Instance namespace when W3C Schema Instance constructs are used. [SD4-46] Data-centric XML instance documents SHOULD use “xsi” as a namespace prefix for all W3C Schema Instance constructs.	[SD4-47] Document-centric XML instance documents MUST declare the W3C Schema Instance namespace when W3C Schema Instance constructs are used. [SD4-48] Document-centric XML instance documents SHOULD use “xsi” as a namespace prefix for all W3C Schema Instance constructs.

Topic	Data-centric	Document-centric
Local Namespace Declarations	[SD4-49] Data-centric XML instance documents MUST NOT use local namespace declarations.	[SD4-50] Document-centric XML instance documents SHOULD NOT use local namespace declarations.
Message-level Schemas	[SD5-1] Message-level Schemas SHOULD be used. [SD5-2] Data-centric Message-level Schemas SHOULD use a target namespace identifier as identified in the Exchange Network Namespace architecture.	[SD5-3] Message-level Schemas MAY be used.
Shared Exchange Network Schemas	[SD5–4] Shared Exchange Network Schemas MUST be used when available. [SD5–5] Data-centric Shared Exchange Network Schemas SHOULD use a target namespace identifier of “urn:us:gov:epa”.	[SD5-6] Shared Exchange Network Schemas MUST be used when available
Functional Area Schemas	[SD5–7] Functional Area Schemas MAY be used. [SD5–8] Data-centric Exchange Network Schemas SHOULD use a target namespace identifier as identified in the Exchange Network Namespace architecture.	[SD5-9] Functional Area Schemas MAY be used.
Voluntary Standards Body Schemas	[SD5-10] Appropriate Voluntary Standard Body Schemas SHOULD be adopted, when appropriate.	[SD5-11] Appropriate Voluntary Standard Body Schemas SHOULD be adopted, when appropriate.
Federal and State Government Schemas	[SD5-12] Federal and state government schemas MAY be used if they are consistent with the guidelines for schema development as set forth by the Federal CIO XML Working Group or those set forth in this document.	[SD5-13] Federal and state government schemas MAY be used if they are consistent with the guidelines for schema development as set forth by the Federal CIO XML Working Group or those set forth in this document.
Nested Includes	[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.	[SD5–16] Exchange Network Schemas SHOULD group like constructs into one schema. [SD5–17] Message-level schemas SHOULD maintain a reasonable number of nested includes.
Code Lists	[SD5–18] Exchange Network schemas SHOULD support code lists through multiple namespaced types.	[SD5–19] Exchange Network schemas SHOULD support code lists through multiple namespaced types.

Topic	Data-centric	Document-centric
Built-In Schema Version Attribute	SD5–20] Data-centric schemas MUST include a version number using the W3C Schema version attribute. [SD5–21] The version number MUST include both a major version component and a minor version component. [SD5–22] Data-centric schemas SHOULD include a version number in their filename.	[SD5–23] Document-centric schemas MUST include a version number using the W3C Schema version attribute. [SD5–24] The version number MUST include both a major version component and a minor version component. [SD5–25] Document-centric schemas SHOULD include a version number in their filename.
User-Defined Version Attribute on Instance Root	[SD5-26] Data-centric schemas MUST define a schema version attribute for use on the instance root.	[SD5-27] Document-centric schemas MUST define a schema version attribute for use on the instance root.
Schema Construct Documentation	[SD5-28] Data-centric schemas SHOULD include schema construct documentation. [SD5-29] Data-centric schemas SHOULD use the documentation element for schema construct documentation. [SD5-30] Data-centric schemas MAY use DTD-style comments for comments pertaining to the structure of the schema.	[SD5-31] Document-centric schemas SHOULD include schema construct documentation. [SD5-32] Document-centric schemas SHOULD use the documentation element for schema construct documentation. [SD5-33] Document-centric schemas MAY use DTD-style comments for comments pertaining to the structure of the schema.
Schema Header Documentation	[SD5-34] Data-centric schemas MUST include schema header documentation.	[SD5-35] Document-centric schemas SHOULD include schema header documentation. [SD5-36] Document-centric schemas SHOULD use the documentation element for schema construct documentation.
ID/IDREF Technique	[SD6-1] Data-centric schemas MUST NOT use the ID/IDREF technique for information association.	[SD6-2] Document-centric schemas MUST NOT use the ID/IDREF technique for information association.
KEY/KEYREF Technique	[SD6-3] Data-centric 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.	[SD6-6] Document-centric schemas SHOULD use the KEY/KEYREF technique for information association. [SD6-7] Extreme caution SHOULD be applied when writing an XPath expression in a selector element to ensure it specifies the intended range. [SD6-8] Special attention SHOULD be paid to the restrictions on KEY and KEYREF declaration names given above.

Topic	Data-centric	Document-centric
Key Technique	[SD6-9] Data-centric 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.	[SD6-12] Document-centric 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. [SD6-13] Extreme caution SHOULD be applied when writing an XPath expression in a selector element to ensure it specifies the intended range. [SD6-14] Special attention SHOULD be paid to the restrictions on KEY declaration names given above.
XLink/XPointer Technique	[SD6-15] Data-centric schemas MUST NOT use the XLink/XPointer technique for information association.	[SD6-16] Document-centric schemas MUST NOT use the XLink/XPointer technique for information association.
UNIQUE Technique	[SD6-17] Data-centric 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.	[SD6-20] Document-centric 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. [SD6-21] Extreme caution SHOULD be applied when writing an XPath expression in a selector element to ensure it specifies the intended range. [SD6-22] Special attention SHOULD be paid to the restrictions on UNIQUE declaration names given above.
Simple Datatype Restriction	[SD7-1] Data-centric schemas SHOULD NOT use simple datatype restriction when a data standard or an approved schema exists. [SD7-2] Data-centric schemas MUST use global simple datatypes. [SD7-3] Data-centric schemas MUST NOT use local simple datatypes.	[SD7-4] Document-centric schemas MAY use simple datatype restriction. [SD7-5] Document-centric schemas MAY use global simple datatypes. [SD7-6] Document-centric schemas MUST NOT use local simple datatypes.
List Technique	[SD7-7] Data-centric schemas MAY use the list technique. [SD7-8] Data-centric 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).	[SD7-9] Document-centric schemas MAY use the list technique. [SD7-10] Document-centric 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).
Union Technique	[SD7-11] Data-centric schemas MAY use the union technique.	[SD7-12] Document-centric schemas MAY use the union technique.
Complex Datatype Restriction	[SD7-13] Data-centric schemas MAY use complex datatype restriction.	[SD7-14] Document-centric schemas MAY use complex datatype restriction.

Topic	Data-centric	Document-centric
Complex Datatype Extension	[SD7-15] Data-centric schemas MAY use complex datatype extension.	[SD7-16] Document-centric schemas MAY use complex datatype extension.
Prohibiting Complex Datatype Derivation	[SD7-17] Data-centric schemas MAY use the final attribute derivation.	[SD7-18] Document-centric schemas MAY prohibit complex datatype derivation.
Prohibiting Use of Derived Complex Datatypes	[SD7-19] Data-centric schemas MAY use the block attribute.	[SD7-20] Document-centric schemas SHOULD NOT prohibit use of derived complex datatypes.
Abstract Datatypes	[SD7-21] Data-centric schemas MUST NOT use abstract datatypes.	[SD7-22] Document-centric schemas MUST NOT use abstract datatypes.
Wildcards	[SD7-23] Data-centric schemas MUST NOT use wildcards.	[SD7-24] Document-centric schemas MUST NOT use wildcards.
Default Element Values	[SD7-25] Data-centric schemas SHOULD NOT use default element values.	[SD7-26] Document-centric schemas MAY use default element values.
Fixed Element Values	[SD7-27] Data-centric schemas SHOULD NOT use fixed element values.	[SD7-28] Document-centric schemas MAY use fixed element values.
Default Attribute Values	[SD7-29] Data-centric schemas SHOULD NOT use default attribute values.	[SD7-30] Document-centric schemas MAY use default attribute values.
Fixed Attribute Values	[SD7-31] Data-centric schemas SHOULD NOT use fixed attribute values.	[SD7-32] Document-centric schemas MAY use fixed attribute values.
Substitution Groups	[SD7-33] Data-centric schemas MUST NOT use substitution groups.	[SD7–34] Document-centric schemas SHOULD NOT use substitution groups. [SD7–35] Document-centric schemas MAY “block” use of substitution groups.
W3C Schema appinfo Element	[SD7-38] Data-centric schemas MUST NOT use the appinfo element.	[SD7-39] Document-centric schemas MUST NOT use the appinfo element.
Notations	[SD7-40] Data-centric schemas MUST NOT use notations.	[SD7-41] Document-centric schemas MUST NOT use notations.

Appendix B
Glossary

abstract datatype	A W3C Schema complex datatype that acts as a "template" that cannot be directly used in an XML instance document
Accredited National Standards Institute X12	A committee that links the standards for multiple industries and sets the norm for a more effective exchange of information
API	application program interface
ASC	Accredited Standards Committee
attribute	A W3C Schema construct that is associated with an element and represents a property or characteristic of that element
attribute group	A W3C Schema construct that contains two or more attributes
attributeFormDefault	A W3C Schema construct that controls the namespace qualification of attributes in an XML instance document
base datatype	A datatype used as the basis for a derived datatype in a schema
block attribute	A W3C Schema construct that prohibits the appearance in an XML instance document of any datatype derived from a given complex datatype
built-in datatype	A datatype, such as string, that is predefined in the W3C Schema standard
cardinality	A W3C Schema property referring to the number of times an element can appear in a given content model in an XML instance document
child element	An element that appears beneath another element in a schema (also known as a subelement)
CIO	chief information officer
complex datatype	A user-defined datatype that contains child elements or attributes

complex datatype extension		A W3C Schema technique in which the definition of a complex datatype is used as the basis of a new complex datatype and further expanded through the addition of constructs
compositor		A W3C Schema construct that groups element declarations
content model		A term used to describe two or more XML constructs grouped together in an XML instance document (in schemas, complex datatypes define content models)
Data Exchange Template	DET	A template for data presentation and exchange that defines the types of information that are required for a particular document as established in predefined standards or agreements (from the State and EPA Information Management Workgroup's *Blueprint for a National Environmental Information Exchange Network*)
datatype derivation		A W3C Schema technique in which new datatypes from existing datatypes in a schema
default attribute value		A value specified for an attribute in a schema that becomes the actual value of the attribute in an XML instance document if the attribute does not appear in the XML instance document
default element value		A value specified for an element in a schema that becomes the actual value of the element in an XML instance document if the element appears as an empty element in the XML instance document
default namespace		A namespace to which all constructs in a schema that are not namespace qualified belong
derived datatype		A datatype that is defined in terms of an existing datatype (the existing datatype is known as a base datatype)
DET		Data Exchange Template
document type definition	DTD	A document that defines the required structure of an XML document and the constraints on its content
DOM		document object model
DTD		document type definition
ECOS		Environmental Council of States
EDR		Environment Data Registry
EIEIT		Enterprise Interoperability and Emerging Information Technology

element		A W3C construct that is represented within an XML instance document by a pair of tags that enclose data
elementFormDefault		A W3C Schema construct that controls the namespace qualification of elements in an XML instance document
empty element		An element that contains no data
enumeration facet		A W3C Schema construct that specifies a set of allowed values for a datatype
Environmental Data Registry	EDR	EPA's comprehensive authoritative source of reference about environmental data
EPA		U.S. Environmental Protection Agency
extensibility		A term referring to the level of expandability of something; used in information technology to describe a program, programming language, or protocol that is designed so that users or designers can later extend its capabilities
Extensible Markup Language	XML	A markup language for documents that contain structured information; XML is a project of the W3C
Extensible Stylesheet Language	XSL	A W3C standard used to transform, query, and format XML instance documents
Extensible Stylesheet Language Transformations	XSLT	A W3C standard used to transform and query XML instance documents; originated from the original XSL specification, XSLT is designed to be used as part of, or independently of, XSL
external schema		A schema that is included in another schema, thereby allowing the constructs from the external schema to be referenced in that schema
facet		A W3C Schema construct that specifies various properties of datatypes
Federal CIO XML Working Group		A working group established by the EIEIT Committee, whose purpose is to accelerate, facilitate, and catalyze the effective and appropriate implementation of XML technology in the information systems and planning of the federal government
final attribute		A W3C Schema construct that prohibits the use of a given complex datatype as a base datatype for datatype derivation

fixed attribute value		An attribute value specified in a schema that is considered by an XML processor to be the actual value of the attribute in an XML document if the attribute does not appear in the XML document
fixed element value		An element value specified in a schema that is considered by an XML processor to be the actual value of the element in an XML document if the element appears in the XML document as an empty element
Geography Markup Language	GML	A markup vocabulary for the transport and storage of geographic information
global complex datatype		A datatype that is a direct descendant of the root element of a schema (also known as a named complex datatype)
global element		An element that is a direct descendant of the root element of a schema
GML		Geography Markup Language
harmonization		The process of ensuring that redundant declarations of registered data elements do not appear within an organization
Hypertext Markup Language	HTML	A W3C standard that defines a set of markup symbols inserted into a file intended for display in a World Wide Web browser; HTML originated from Standard Generalized Markup Language
ID datatype		A W3C Schema built-in datatype used along with the IDREF datatype to associate information in an XML document
IDREF datatype		A W3C Schema built-in datatype used along with the ID datatype to associate information in an XML document
IDREFS datatype		A W3C Schema built-in datatype used along with the ID datatype to associate information in an XML document; the IDREFS datatype is similar to the IDREF datatype, but contains a whitespace-delimited list of ID values for representing one-to-many associations
IETF		Internet Engineering Task Force
IM		information management
IRM		information resource management

lexical format		The format used to define a W3C Schema built-in datatype; for example, the lexical format for the "date" datatype is YYYY-MM-DD
list technique		A W3C Schema technique in which a whitespace–separated list of values is created from a base datatype
local complex datatype		A datatype that is associated with a single element in a schema (also known as an anonymous complex datatype)
local element		An element that is not a direct descendant of the root element of a schema, but is instead nested inside the schema structure
Lower Camel Case	LCC	A naming convention that capitalizes the first character of each word except the first word(*Example:* <UpperCamelCaseElement lowerCamelCaseAttribute="Whatever"/>)
mixed content		A free-form combination of character data and child elements within a content model
model group		A W3C Schema construct containing one or more elements
name collision		A condition that occurs when two constructs of the same name but different datatypes are included in the same schema; name collisions are avoided through the use of namespaces
namespace		A W3C Schema mechanism used to associate schema constructs with a conceptual space that defines that defines a markup vocabulary
namespace coercion		A condition in which constructs in an included schema are "coerced" to become part of the including schema's target namespace
namespace prefix		An identifier in a schema that associates constructs with the namespace to which they belong; such constructs are said to be "namespace qualified"
NEI		National Emissions Inventory
NMTOKEN datatype		A W3C Schema built-in datatype that is a legal XML name string without any constraints placed upon its initial character
NMTOKENS datatype		A W3C Schema built-in datatype that is a whitespace-delimited list of NMTOKEN values

nondeterministic		A term referring to the availability of more than one choice of a next move at some step in a computational process; when used regarding W3C Schema, it refers to content models in a schema that require an XML processor to look ahead in order to be able to associate the content model with a declaration
OASIS		Organization for the Advancement of Structured Information Standards
occurrence indicator		A W3C construct that indicates the cardinality of an element or model group
OEI		Office of Environmental Information
Office of Environmental Compliance and Assurance	OECA	An EPA office whose primary mission is to ensure compliance with the nation's environmental laws, thereby reducing threats to public health and the environment.
OIC		Office of Information Collection
OIT		Office of Information Technology
OMB Circular A-119		A document that directs agencies to use voluntary consensus standards in lieu of government-unique standards except where using the consensus standards would be inconsistent with law or impractical
Organization for the Advancement of Structured Information Standards	OASIS	A nonprofit international consortium that creates interoperable industry specifications based on public standards, such as XML and Standard Generalized Markup Language (SGML)
Petroleum Industry Data Exchange	PIDX	The American Petroleum Industry's EDI and XML action group
PIDX		Petroleum Industry Data Exchange
processContents attribute		An attribute used with the W3C Schema wildcard feature that specifies how an XML processor should validate constructs used in place of a wildcard declaration
RCRA		Resource Conservation and Recovery Act
RFC		Request for Comments
root element		The top-level element of a schema or XML instance document
SAX		Simple API for XML

schema		See *XML Schema*
SGML		Standard Generalized Markup Language
simple datatype		A datatype defined by the W3C Schema team and included in the W3C Schema standard (also known as a “built-in” datatype)
simple datatype restriction		A W3C Schema technique in which the properties of a simple datatype are used for the basis of a new simple datatype and further restricted
Simple Object Access Protocol	SOAP	An XML-based protocol that allows software running on disparate operating systems and in different environments to communicate over the Internet; originally a W3C Note, continuing development of SOAP taking place as the XML Protocol W3C Standard.
subelement		An element that appears beneath another element in a schema (also known as a child element)
substitution group		A W3C Schema feature that allows an element to replace another element in an XML instance document without requiring schema updates
target namespace		A namespace associated with a single schema that indicates that the schema is acting as a “collector” of constructs declared within it
trading partner agreement	TPA	A document that defines the conditions under which two partners will transact business together
TRG		Technical Resource Group
UBL		Universal Business Language
UCS		Universal Character Set
Uniform Resource Identifier	URI	An identifier for a resource on the World Wide Web
Uniform Resource Locator	URL	A type of URI that indicates the address of a resource on the Internet
Uniform Resource Name	URN	A type of URI that is more closely tied to the actual location and meaning of the resource it represents than is a URL
union technique		A W3C Schema technique in which a range of possible values for a simple datatype is restricted through the union of two or more simple datatypes

Universal Business Language	UBL	An OASIS technical committee whose purpose is to develop a freely available standard library of XML business documents that can be used in international electronic commerce
Universal Character Set	UCS	
Upper Camel Case	UCC	A naming convention that capitalizes the first character of each word and compounds the name (*Example:* <UpperCamelCaseElement/>)
URI		Uniform Resource Identifier
URL		Uniform Resource Locator
URN		Uniform Resource Name
use indicator		A W3C Schema construct that specifies optionality for an attribute
UTF		UCS transformation format
VADEQ		Virginia Department of Environmental Quality
Variable Content Model		A W3C Schema feature that allows the structure of information within an XML instance document to vary greatly without requiring schema updates
W3C		World Wide Web Consortium
well-formed XML		A term referring to XML that conforms to the syntax requirements of the W3C XML standard
wildcard		A W3C Schema feature used to create a "placeholder" at which any well-formed XML can appear in an XML instance document
World Wide Web Consortium	W3C	A group that develops specifications to lead the World Wide Web to its full potential as a forum for information, commerce, communication, and collective understanding; XML is a project of the W3C
W3C XML Schema		An XML-based constraint language that defines the required structure of an XML document and constrains its content
XML		Extensible Markup Language
XML instance document		A set of data that conforms to XML standards

XML Linking Language	XLink	A W3C standard used to describe links between constructs both within and across XML instance documents
XML Pointer Language	XPointer	A W3C standard used to reference locations in XML instance documents both from within and outside those documents
XML processor		A general term for a software product that validates a schema or XML instance document, and that may also validate an XML instance document against its associated schema (also known as an XML parser)
XML tag		A sequence of one or more characters surrounded by <> symbols that is used to mark up data in an XML document
XML-RPC		An XML-based protocol that allows software running on disparate operating systems and in different environments to communicate over the Internet; similar in concept to SOAP
XPointer Bare Names Notation		An XPointer technique that uses a shorthand notation to reference elements that contain an attribute of datatype ID
XSD		Schema Definition Language
xsi:type construct		A W3C instance construct used in datatype derivation scenarios that indicates to an XML processor exactly which datatype (base or derived) applies
XSL		Extensible Stylesheet Language
XSL stylesheet		An XML document conforming to the Extensible Stylesheet Language W3C standard, the purpose of which is to transform, query, or format an XML instance document
XSLT		Extensible Stylesheet Language Transformations