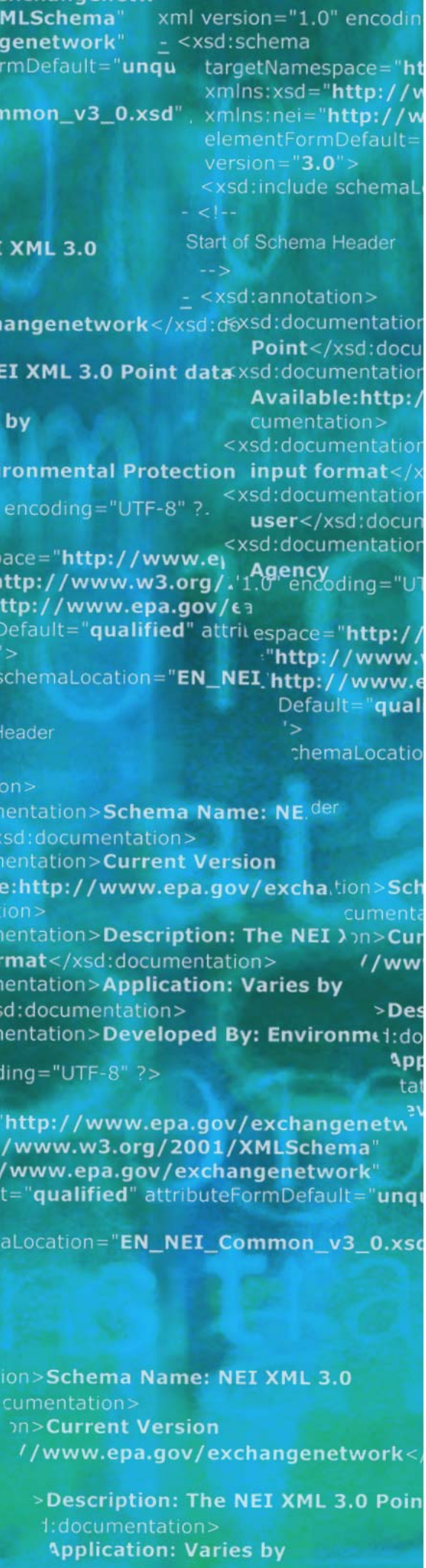


XML Schema Design Rules and Conventions (DRC)

Interim Update
For the Exchange Network

Version: 1.1

DEPRECATED

Revision Date: 04/06/2006

Environmental Information exchange Network

ACKNOWLEDGEMENTS

This document was developed with invaluable input and support from the following individuals:

Andrew Battin	U.S. EPA
Charles Freeman	U.S. EPA
Chris Clark	U.S. EPA
Dennis Murphy	Delaware DNR
Dennis Burling	Nebraska DEQ
Connie Dwyer	U.S. EPA
Glen Carr	Oregon DEQ
Joe Wilson	U.S. EPA
Nathan Wilkes	U.S. EPA
Nick Mangus	U.S. EPA
Randy Moody	North Carolina DENR
Scott Totten	Missouri DNR
Tom Aten	Wisconsin DNR
Maryane Tremaine	U.S. EPA
Mitch West	Oregon DEQ

Prepared By

WINDSOR SOLUTIONS, INC.

4000 Kruse Way Place
Building 2, Suite 285
Lake Oswego, OR 97035
(503) 675-7833
http://www.windsorsolutions.com/

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

1. Introduction

1.1. Overview

The *XML Design Rules and Conventions v1.0* (DRCs) were published in September, 2003 to define the standards for designing XML Schema for the Exchange Network (Network). Since that time, Network participants have gained a wealth of experience creating XML schema and implementing data exchanges. Also during this period, Network governance and associated workgroups have issued guidance to schema developers which supersede the information provided in the original DRCs.

The evolution of the Network has rendered portions of the original DRCs outdated, leaving schema developers with the task of collecting and reconciling the various guidance in order to determine the correct methods for constructing XML schema for the Network. This document attempts to unify the original DRCs with the latest developments in network guidance into a single resource, thus providing schema developers with a single, simple tool for building Network-compliant schema.

This document represents an interim update to the DRCs. A major DRC update (v2.0) will be performed following the release of the final Core.gov schema XML design rules. While all of the original DRCs are listed in this document (see *Appendix A),* it does not replicate the discussions in the original document describing examples, rationale and benefit/drawback of each approach. The DRC v1.0 should be referenced if this information is sought. However for new rules and significantly altered original rules, discussion is provided which explains the basis of the change.

1.2. How to Use this Document

This document is intended for use by Exchange Network schema developers. Developers should use this document as the point of reference for approved design conventions against which all Network schema must conform. This document should be consulted prior to beginning development on a new schema or modifying an existing schema. After the schema has been developed, this document should be used to check conformance of the schema with design rules, subject to the Network schema review process.

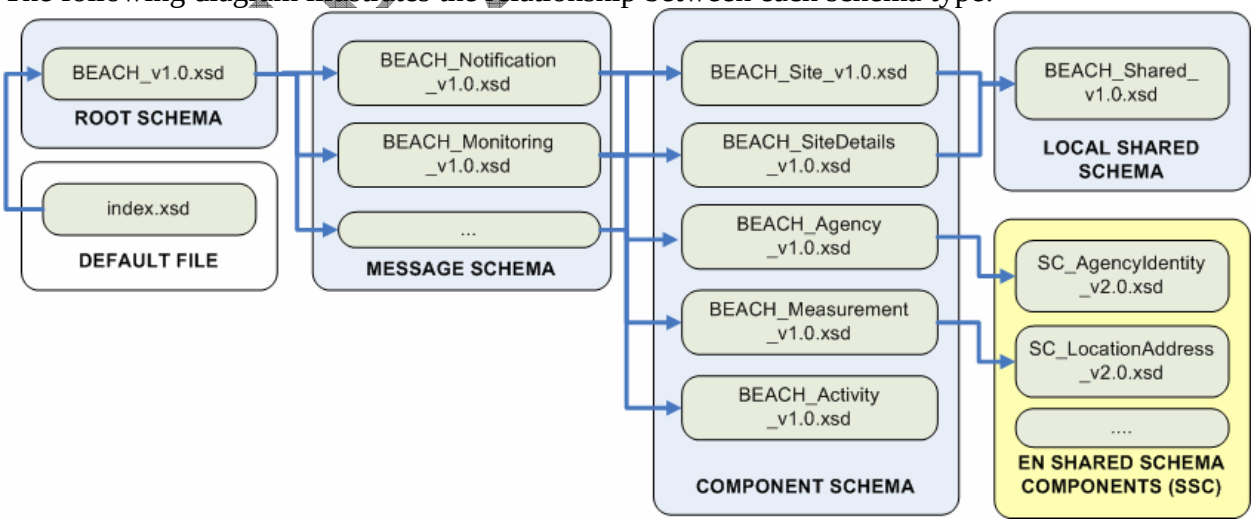
1.3. Exchange Network Schema Types

The following terms are used within this document to refer to schema based on their content or usage. Developers should be familiar with each of these terms in order to understand which schema types are affected by a given rule or guideline.

Schema Type	Definition
Default File	A schema with the name "index.xsd". This schema points to the root schema for the given namespace using the W3C *include* construct. This file allows the recipient to load all definitions for a namespace without needing to know the filename of the root schema. Additionally, it allows viewers of the Exchange Network repository to easily locate the root schema by reading the name of the file referenced in the schema's *include* construct. The Exchange Network schema repository is configured to return this document when the namespace URL is requested.
Root Schema	A schema which either directly or indirectly references (includes) all

	schema constructs in the given namespace. The Root Schema can be used to validate any XML instance document which references the namespace. Each namespace must have a single Root Schema. The Root Schema should reference one or more Message Schemas. A given namespace must have only one root schema.
Message Schema	A schema which is used to convey a payload in Exchange Network transactions. Each namespace must have at least one Message Schema but may have more than one. A Message Schema references (includes) one or more Component Schema.
Component Schema	A schema which defines a data structure only. One or more Component Schemas are referenced in one or more Message Schema to build a complex representation of data. Component schema may reference (include) other component schema. Component schema should only contain complex datatype definitions, not elements or simple types.
Local Shared Schema	A schema which defines a list of generic simple or complex datatypes which are referenced within one or more Component Schema in the namespace. Local Shared Schema are typically simple in nature and do not "include" any other schema. They are the lowest level in the Network schema hierarchy. In some instances, a namespace may not have any Local Shared Schema. There should only be one Local Shared Schema in a given namespace.
EN Shared Schema Components (SSCs)	Schema components that have been standardized by the Exchange Network for use in all registered schema that need to define a common feature such as an address, geographic coordinate or facility. All schema developers must evaluate the suitability of existing SSCs for inclusion into new or modified schema. SSCs reduce effort by allowing developers to consume standard XML structures into their schema without having to reinvent them. SSCs are the product of the Exchange Network Core Reference Model (CRM) project.

The following diagram illustrates the relationship between each schema type:



Relationship between Schema Types (Hypothetical Example)

2. DRC Change Summary

The following list highlights the significant changes to the Network schema design rules from Section 3 of this document as compared to the DRC v1.0.

Section 3.1 - General XML Design

Several new guidelines have been added which address general XML structure. Firstly, creating multiple references to global elements with complex content which recur (maxOccurs > 0) is now discouraged. Secondly, it is now recommended to nest recurring elements within a container element. Thirdly, it is recommended that Component Schema only include datatypes, not elements. Lastly, guidelines regarding comments in schema have been updated.

Section 3.2 - File Naming Rules and Guidelines

File naming rules have been updated to be consistent with the new Exchange Network schema versioning strategy. These rules supersede the file naming rules in the DRC v1.0.

Section 3.3 - XML Tag Naming Conventions

Two simple changes have been made to the rules regarding XML tag naming conventions. It is now mandatory that all element, attribute, and datatype tag names be unique within a namespace. Datatypes names must now end in either "Type" or "DataType".

Section 3.4 - Namespaces

The namespace guidance provided in the DRC v1.0 has been superseded by new guidance. All Network schema must implement a URL-formatted namespace structured as **http://www.exchangenetwork.net/schema/(*category*)[/(*subcategory)*]/(*version*)**. Additional namespace rules and guidance are provided in the corresponding section of this document.

Section 3.5 - Shared Schema Components (SSC)

It is now required that schema developers reuse (include) Network SSCs whenever a "good fit" exists. SSCs describe such common features as facilities, individuals, permits and other features common to the environmental arena.

Section 3.6 - Schema Versioning

Specific guidance on schema versioning schema is now provided. Developers are urged to pay close attention to proper versioning since significant repercussions can arise from improper implementation of these guidelines.

Section 3.7 - Exchange Network Header

Schema developers may choose to implement the Exchange Network Header in their schema although it is not required. Note that discussions regarding header use and possible revisions are ongoing at the time of this writing which may result in revised guidance.

Removal of Document-centric Schema References

The original DRC divided all guidance between two schema types; data-centric schema and document-centric schema. In the four years which have passed since the DRC release, no document-centric schemas have been produced for the Network. The focus of schema development has been on "data-centric schema" as defined by the original DRC.

References to document-centric schema have been removed in this version. As a result, readers of the *Summary of Design Rules* in Appendix A may notice that nearly every other DRC number is missing (i.e. SD2-2, SD2-4). This is due to the removal of all document-centric rules. Furthermore, rules which have remained unchanged from DRC v1.1 still contain the term "data-centric schema". In this document, "data-centric schema" is analogous to "Exchange Network schema".

3. Updated Schema Design Rules

3.1. General XML Design

3.1.1. Multiple References to Global Complex Elements

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

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" <!-- namespaces omitted for clarity -->>
    <xsd:element name="PermittedFeature">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="PermittedFeatureIdentifier"/>
                <xsd:element ref="PermittedFeatureName"/>
                <xsd:element ref="PermittedFeatureDescription"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="PermittedFeatureIdentifier" type="xsd:string"/>
    <xsd:element name="PermittedFeatureName" type="xsd:string"/>
    <xsd:element name="PermittedFeatureDescription" type="xsd:string"/>
</xsd:schema>
```

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

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

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

Many developer tools attempt to represent each of these objects as “tables” in the relational database paradigm (or more accurately, two *classes* in the same namespace). This structure would result in the tool attempting to create two “tables” with the same name; one as a child of

Permit and one as a child of Facility. This results in a name collision which cannot be resolved by the tool. Two approaches to resolving this situation are provided below.

Approach #1 – Create separate elements for each reference
Instead of creating a global element named *PermittedFeature*, create a global complex type named *PermittedFeatureDataType*. Then create two global elements with different names which reference the newly created type. The elements could be named "*PermitPermittedFeature*" and "*FacilityPermittedFeature*". This would enable the same structure to be reused but given different names based upon its implementation. In the following example, the developer uses the approach outlined above:

```
 <?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" <!-- namespaces omitted for clarity -->>
    <!-- complex global types -->
    <xsd:complexType name="PermittedFeatureDataType">
        <xsd:sequence>
            <xsd:element ref="PermittedFeatureIdentifier"/>
            <xsd:element ref="PermittedFeatureName"/>
            <xsd:element ref="PermittedFeatureDescription"/>
        </xsd:sequence>
    </xsd:complexType>
    <!-- simple global elements -->
    <xsd:element name="PermittedFeatureIdentifier" type="xsd:string"/>
    <xsd:element name="PermittedFeatureName" type="xsd:string"/>
    <xsd:element name="PermittedFeatureDescription" type="xsd:string"/>
    <!-- complex global elements -->
    <xsd:element name="PermitPermittedFeature" type="PermittedFeatureDataType"/>
    <xsd:element name="FacilityPermittedFeature" type="PermittedFeatureDataType"/>
    <xsd:element name="Permit">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="PermitPermittedFeature" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Facility">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="FacilityPermittedFeature" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```


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

Approach #2 – Use KEY and KEYREF to create multiple references to a single element list
The second option is to define a single list of referenced items within the schema. Items within the list can then be referenced using the W3C schema KEY and KEYREF constructs. This approach is very similar to what might be done in a relational database when a given table is a child to multiple other tables.

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

uses a local *ContactList* element for improved readability although global elements are required for Exchange Network schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" <!—namespaces omitted for clarity -->
>elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xsd:element name="NPDES">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Facility" maxOccurs="unbounded"/>
                <xsd:element ref="Permit" maxOccurs="unbounded"/>
                <xsd:element name="ContactList">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="Contact" maxOccurs="unbounded">
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element name="ContactIdentifier"/>
                                        <xsd:element name="ContactName"/>
                                    </xsd:sequence>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:keyref name="FacilityContactForeignKey" refer="ContactKey">
            <xsd:selector xpath="Facility"/>
            <xsd:field xpath="FacilityContactIdentifier"/>
        </xsd:keyref>
        <xsd:keyref name="PermitContactForeignKey" refer="ContactKey">
            <xsd:selector xpath="Permit"/>
            <xsd:field xpath="PermitContactIdentifier"/>
        </xsd:keyref>
        <xsd:key name="ContactKey">
            <xsd:selector xpath="ContactList/Contact"/>
            <xsd:field xpath="ContactIdentifier"/>
        </xsd:key>
    </xsd:element>
    <xsd:element name="Facility">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="FacilityIdentifier" type="xsd:integer"/>
                <xsd:element name="FacilityName" type="xsd:string"/>
                <xsd:element name="FacilityContactIdentifier"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Permit">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="PermitIdentifier" type="xsd:integer"/>
                <xsd:element name="PermitNumber" type="xsd:string"/>
                <xsd:element name="PermitType" type="xsd:string"/>
                <xsd:element name="PermitContactIdentifier"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```


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

```
<?xml version="1.0" encoding="UTF-8"?>
<NPDES xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Facility>
        <FacilityIdentifier>111</FacilityIdentifier>
```

```
        <FacilityName>ACME Industries, Inc.</FacilityName>
        <FacilityContactIdentifier>123</FacilityContactIdentifier>
    </Facility>
    <Permit>
        <PermitIdentifier>1111</PermitIdentifier>
        <PermitNumber>WA0000123</PermitNumber>
        <PermitType>Individual Permit</PermitType>
        <PermitContactIdentifier>123</PermitContactIdentifier>
    </Permit>
    <ContactList>
        <Contact>
            <ContactIdentifier>123</ContactIdentifier>
            <ContactName>Joe Smith</ContactName>
        </Contact>
    </ContactList>
</NPDES>
```

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

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

Rules and Guidelines	
Rule	**Description**
[GD1-A]	Elements with recurring, complex content SHOULD NOT be referenced more than once within the same root document, even if the element exists in different schema files. When the same structure needs to be reused in multiple areas, either create a new element with a different tag name utilizing a common datatype or use the KEY and KEYREF construct to represent the element in a single list.

3.1.2. Implementing Recurring Elements with Simple Content

In XML schema, it is possible to indicate that a given element may appear one or more times within a sequence of other elements. In the following example, the schema developer has indicated that an individual may have one or more phone numbers:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" <!—namespaces omitted for clarity -->>
    <xsd:element name="Person" type="PersonDataType"/>
    <xsd:complexType name="PersonDataType">
        <xsd:sequence>
            <xsd:element ref="FirstName"/>
            <xsd:element ref="LastName"/>
            <xsd:element ref="PhoneNumber" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="FirstName" type="xsd:string"/>
    <xsd:element name="LastName" type="xsd:string"/>
    <xsd:element name="PhoneNumber" type="xsd:string"/>
```

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

An XML instance document which conforms to this schema might look like the following:

```
<Person>
    <FirstName>Joe</FirstName>
    <LastName>Smith</LastName>
    <PhoneNumber>123-111-1111</PhoneNumber>
    <PhoneNumber>123-222-2222</PhoneNumber>
    <PhoneNumber>123-333-3333</PhoneNumber>
</Person>
```

While this is acceptable schema design by W3C standards, it can be more challenging for developers to compose and decompose the instance document since the list is embedded in a series of other sibling elements. It also makes browsing the instance document less intuitive for a human reader since most XML viewers (including Web browsers) allow the viewer to expand and collapse nodes in the instance document for easier viewing. When recurring content occurs inline with single-instance content it cannot be collapsed, requiring the user to scroll past the recurring content to look further down the document.

To address this issue, developers are encouraged to create a separate container element for any content which is expected to recur within a parent element. In the following example, the schema developer has chosen to place the phone numbers in a special *PhoneNumberDetails* element:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" <!—namespaces omitted for clarity -->>
    <!-- Complex Global Types -->
    <xsd:complexType name="PersonDataType">
        <xsd:sequence>
            <xsd:element ref="FirstName"/>
            <xsd:element ref="LastName"/>
            <xsd:element ref="PhoneNumberDetails"/>
        </xsd:sequence>
    </xsd:complexType>
    <!-- simple global elements -->
    <xsd:element name="FirstName" type="xsd:string"/>
    <xsd:element name="LastName" type="xsd:string"/>
    <!-- complex global elements -->
    <xsd:element name="Person" type="PersonDataType"/>
    <xsd:element name="PhoneNumberDetails" type="PhoneNumberDetailsDataType"/>
    <xsd:complexType name="PhoneNumberDetailsDataType">
        <xsd:sequence>
            <xsd:element ref="PhoneNumber" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="PhoneNumber" type="xsd:string"/>
</xsd:schema>
```

Using the revised schema, note the new structure of the phone numbers in an example instance document:

```
<Person>
    <FirstName>Joe</FirstName>
    <LastName>Smith</LastName>
    <PhoneNumberDetails>
        <PhoneNumber>123-111-1111</PhoneNumber>
        <PhoneNumber>123-222-2222</PhoneNumber>
        <PhoneNumber>123-333-3333</PhoneNumber>
    </PhoneNumberDetails>
</Person>
```

Rules and Guidelines	
Rule	**Description**
[GD1-B]	Recurring elements (maxOccurs > 1) SHOULD be nested within their own container element.

3.1.3. Element Declaration in Exchange Network Schema

In the *Definition of Schema Types Used in this Document* section of this document (section 1.3), six schema types are introduced depending on the role of the schema file within a given flow; the Default File, the Root Schema, Message Schemas, Component Schemas, Local Shared Schemas, and Shared Schema Components (SSCs).

Component Schemas are intended to store modular structures only. These structures should describe a single feature or activity such as a facility or inspection. These structures can then be assembled into larger structures in Message Schemas. Given this design approach, developers are encouraged to only declare XML datatypes within Component Schema. Whenever possible, all elements should be defined in a given Message Schema within the namespace. This helps reduce the possibility of multiple elements with the same name being declared in separate Message Schema which may cause confusion for implementers and lead to redundant element declarations.

As a corollary to this statement, it is also important to define the root element for a Message Schema within the Message Schema itself. Burying the root element within a schema other than the Message Schema make it difficult for schema implementers to determine which schema element is the root element. Since all elements must be global, any element can be a valid instance document root element. Placing the true root element for a given message within the Message Schema itself makes it clear which element is intended to be the root element.

Rules and Guidelines	
Rule	**Description**
[GD1-C]	Wherever possible, component schema SHOULD NOT define any elements, only datatypes.
[GD1-D]	Message-level root elements MUST be defined in the message schema.

3.1.4. Schema Documentation

This section contains minor clarifications to design rules specified in DRC v1.0.

The W3C schema specification allows for documentation to be included in schema. This allows the schema to contain embedded descriptions of each construct. Since Exchange Network schema are intended to be implemented by multiple partners, documentation is needed to communicate the context and usage of elements in schema to the consumer who may not have the same background or business expertise that the schema developer has. Note that documentation should NOT be used to store code lists or information which is not relevant to the construct which is being documented.

In the following example, documentation is provided for the *ViolationDeterminationDate* element from the *SC_SimpleContent_v2.0.xsd* schema:

```
…
<xsd:element name="ViolationDeterminedDate" type="ViolationDeterminedDateDataType">
    <xsd:annotation>
        <xsd:documentation>The calendar date the Responsible Authority determines that a regulated entity is
            in violation of a legally enforceable obligation.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>
…
```

XML schema may also contain HTML-style comments (i.e. <!—comment -->). It is recommended that schema developers avoid using this technique, especially if the purpose of the comment is to describe some aspect of a given schema construct. In this case, the documentation construct should be used as described above.

There may be times when HTML-style comments are acceptable such as when the developer wishes to insert visual delimiters between constructs such as elements and datatypes.

Rules and Guidelines	
Rule	**Description**
[SD5-28]	All schema SHOULD include schema construct documentation using the W3C documentation element. Documentation SHOULD only describe the element or datatype and SHOULD NOT contain lists of acceptable codes, implementation details or other information not directly related to the meaning of the construct.
[SD5-30]	HTML-style comments (<!--comment -->) SHOULD NOT be used in schema.

3.2. File Naming Rules and Guidelines

Schema file naming conventions are an important part of the schema versioning strategy for the Exchange Network. Schema file names must be implemented consistently to ensure the Exchange Network registry and users can differentiate between schema versions. Rules in this section are closely related to the rules in the *Schema Versioning* section of this document. Please note that the file naming convention outlined here supersedes the file naming convention in the DRC v1.0.

Naming rules vary depending on the schema type[1]. The following list provides examples of the proper naming convention for each schema type:

Schema Type	Convention	Examples
Default File	"index.xsd"	index.xsd
Root Schema	FlowName [+ "_" + *FlowCategoryName*] + "_v" + *Version* + ".xsd".	NPDES_v1.0.xsd
Message Schema	*FlowName* [+ "_" + *FlowCategoryName*] + "_" + *MessageName* + "_v" + *Version* + ".xsd".	NPDES_Permit_v1.1.xsd NPDES_Inspection_v1.1.xsd
Component schema	*FlowName* [+ "_" + *FlowCategoryName*] + "_" + *ComponentName* + "_v" + *Version* + ".xsd".	NPDES_MonitoringPointDetails_v1.1.xsd
Local Shared Schema	*FlowName* [+ "_" + *FlowCategoryName*] + "_Shared" + "_v" + *Version* + ".xsd".	NPDES_Shared_v1.1.xsd

Each component of the name is defined as follows:

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

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

Rules and Guidelines	
Rule	**Description**
[GD2-A]	Each namespace must contain a default schema named "index.xsd" which contains only an include construct referencing the Root Schema.

[1] Please refer to the *Section 1.3 Exchange Network Schema Types* for more information

[GD2-B]	Root Schema MUST utilize the following naming format: *FlowName* [+ "_" + *FlowCategoryName*] + "_v" + *Version* + ".xsd".
[GD2-C]	Message Schema MUST utilize the following naming format: *FlowName* [+ "_" + *FlowCategoryName*] + "_" + *MessageName* + "_v" + *Version* + ".xsd".
[GD2-D]	Component Schema MUST utilize the following naming format: *FlowName* [+ "_" + *FlowCategoryName*] + "_" + *ComponentName* + "_v" + *Version* + ".xsd".
[GD2-E]	Local Shared Schema MUST utilize the following naming format: *FlowName* [+ "_" + *FlowCategoryName*] + "_Shared" + "_v" + *Version* + ".xsd".

3.3. XML Tag Naming Conventions

Two simple changes have been made to the rules regarding XML tag naming conventions. It is now mandatory that all element, attribute, and datatype tag names be unique within a namespace. In contrast, W3C guidelines allow both an element and a datatype to have the same name[2].

To further clarify schema constructs for exchange implementers, all datatype names must end with either "Type or "DataType". The latter is preferred since element names may already end in "Type".

Elements and Datatypes are often paired. In such instances, the naming convention is identical with the datatype name being post-fixed with "DataType". In the following example from the Shared Schema Components SC_CountryIdentity_v2.0.xsd, the schema developer creates an element to describe a country (*CountryIdentity*) and also defines a datatype for the element (*CountryIdentityDataType*):

```
<xsd:complexType name="CountryIdentityDataType">
    <xsd:sequence>
        <xsd:element minOccurs="0" maxOccurs="1" ref="CountryCode"/>
        <xsd:element minOccurs="0" maxOccurs="1" ref="CountryCodeListIdentifier"/>
        <xsd:element minOccurs="0" maxOccurs="1" ref="CountryName"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="CountryIdentity" type="CountryIdentityDataType">
    <xsd:annotation>
        <xsd:documentation>
            A designator and associated metadata used to identify a primary geopolitical unit of the world.
        </xsd:documentation>
    </xsd:annotation>
</xsd:element>
```

Rules and Guidelines	
Rule	**Description**
[GD3-8]	Within a namespace, all element, attribute, and datatype tag names MUST be unique.
[GD3-A]	All datatype names MUST end with either "Type" or "DataType".

[2] Two constructs of the same type (such as elements) may also have the same name, but only if the elements' scope does not overlap. Since all elements and types must be globally defined per Exchange Network rules, this situation should never occur.

3.4. Namespaces

3.4.1. Namespaces in Exchange Network Schema

In January 2006, the EN Network Technology Group (NTG) released final guidance on the use of namespaces in Exchange Network Schema. Because namespace names are an important part of the Exchange Network schema versioning strategy, it is imperative that schema developers follow the namespace naming rules when designing schema. It is highly recommended that schema developers review the *Guidance on Namespace Organization, Naming, and Schema File Location v1.11* document (Namespace Document) located on the Exchange Network Web site (see Appendix C).

As stated in the Namespace Document, Exchange network namespaces are URL formatted. The rationale for choosing a URL-formatted namespace (over a URN-formatted namespace) is as follows:

- URL formatted namespaces are simple, and intuitive to internet users and developers
- Meet namespace requirements set out by the W3C
  - By their nature URLs refer to a unique internet location
  - Within the scope of the EN, URLs pointing to `http://www.exchangenetwork.net/schema`, will be persistent
- Namespaces will be unique to flows determined by the inherent structure of the repository
- URL namespace resolution is universally supported by all web applications
  - Secondary registration is not required with IANA – reducing administrative overhead
  - On-the-spot resolution prevents naming errors – users get instantaneous feedback if they misspell a namespace by accessing the URL as a hyperlink
- The ability to resolve namespaces to a network location within the EN repository supports other EN priority initiatives including:
  - Schema management and versioning
  - Improving the Schema Repository
- Standardized implementation of URL formatted namespaces supports future development initiatives based upon resolution of namespaces (e.g. RDDL[3])

Examples of Exchange Network Namespaces are as follows:

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:TRI="http://www.exchangenetwork.net/schema/TRI/2" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!-- information removed for example purposes -->
</xsd:schema>
```

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

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

Rules and Guidelines	
Rule	**Description**
[SD4-A]	The schema namespace name MUST be URL-formatted as "http://www.exchangenetwork.net/schema/{category}[/sub-category]/Version".
[SD4-B]	Exchange Network namespaces MUST contain a category term which clearly and uniquely defines the type of data being exchanged.
[SD4-C]	Exchange Network namespaces MAY contain a subcategory term which further divides the data exchange into smaller components.
[SD4-D]	Exchange Network namespaces MUST contain a major version number as the last part of the namespace name.
[SD4-E]	Exchange Network namespaces MUST NOT contain a minor version number in the namespace name.

3.4.2. Target Namespaces

The DRC v1.0 declares that schemas MUST use target namespaces[4]. This rule is further restricted to state that the target namespace must match one of the declared namespace qualifiers. This ensures that included schemas which do not themselves have a declared target namespace (as the Shared Schema Components 2.0 will be structured) will in essence become a member of one of the local target namespace.

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

```
<xsd:schema xmlns:TRI="http://www.exchangenetwork.net/schema/TRI/2"
        targetNamespace="http://www.exchangenetwork.net/schema/TRI/2"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- information removed for example purposes -->
</xsd:schema>
```

Rules and Guidelines	
Rule	**Description**
[SD4-F]	The schema's targetNamespace MUST match the namespace name of one of the declared namespace qualifiers, but not the w3c qualifier.

3.4.3. Schema Location Attribute

Please note that this rule applies only to XML instance documents, not XML schema.

The DRC v1.0 declares that MUST use the *schemaLocation* construct when listing the storage location of the schema to which the XML instance document validates[5]. This rule is further extended to state that the value specified in the *schemaLocation* attribute must match the namespace URL. The following example shows an example of proper use of the schema location attribute:

[4] See [SD4-13] for more information.
[5] See [SD4-37] for more information.

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

Note that the *schemaLocation* attribute is formatted to repeat the namespace name and the schema location, separated by a space character. This is an acceptable format according to W3C specifications[6]. In this example, the first string defines the namespace name while the second string identifies the resolvable location where the schema can be retrieved. The Exchange Network repository will be configured to return the Default File (index.xsd) for the namespace which will enable the parser to validate the instance document against the Message Schema(s) in the given namespace.

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

If the *schemaLocation* tag is used to refer to an EN schema, then the location must match the namespace.

Rules and Guidelines	
Rule	**Description**
[SD4-H]	If a schemaLocation is specified in an XML instance document, the location MUST match the namespace URL address.

[6] The schemaLocation attribute is of type anyURI which specifically allows this syntax.

3.5. Shared Schema Components (SSCs)

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

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

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

The Network Operations Board (NOB) released a decision memorandum in October, 2005 mandating the usage of SSC in schema where possible. The memorandum is quoted in part below:

> "The Shared Schema Components (SSC) are an important tool for promoting Exchange Network consistency and interoperability. SSC incorporate approved data standards – the Exchange Network's fundamental vocabulary – and streamline schema development by organizing related data elements common to multiple data exchanges. Use of SSC leads to more stable schema and improved data quality over time."
>
> "Consequently, developers must use SSC in schema when they are an appropriate fit for the targeted business process. Neither conformance to the data model of an underlying system nor the presence of unneeded individual XML tags are sufficient reasons to reject SSC usage. Alternatively, the use of SSC could be rejected when the basic component or data element definitions do not fit the business process' context."(NOB 2005-02)

As is made clear by the memorandum, it is imperative that schema developers thoroughly examine the elements and datatypes available in the SSCs and evaluate their fitness for use in the target schema.

There are three levels in which SSCs may be integrated into a target schema. All options begin by including one or more SSC 2.0 schemas in the target schema. Each option is listed below:

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

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

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

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

```
<xsd:schema xmlns:TRI="http://www.exchangenetwork.net/schema/TRI/2" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- include other TRI component schemas -->
    <xsd:include schemaLocation="TRI_MailingAddress_v2.0.xsd"/>
    <xsd:include schemaLocation="TRI_GeographicLocationDescription_v2.0.xsd"/>
    <!-- bring desired SSCs into the TRI namespace -->
    <xsd:include schemaLocation="http://www.exchangenetwork.net/schema/SC/SC_SimpleContent_v2.0.xsd"/>
    <xsd:include schemaLocation="http://www.exchangenetwork.net/schema/SC/SC_LocationAddress_v2.0.xsd"/>
    <xsd:element name="Facility" type="TRI:FacilityDataType">
        <xsd:annotation>
            <xsd:documentation>Facility Identification data</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:complexType name="FacilityDataType">
        <xsd:sequence>
            <xsd:element ref="TRI:FacilityIdentifier" minOccurs="0"/>
            <xsd:element ref="TRI:FacilitySiteName" minOccurs="0"/>
            <xsd:element ref="TRI:LocationAddress" minOccurs="0"/>
            <xsd:element ref="TRI:MailingFacilitySiteName" minOccurs="0"/>
            <xsd:element ref="TRI:MailingAddress" minOccurs="0"/>
            <xsd:element ref="TRI:FacilitySICDetails" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element ref="TRI:GeographicLocationDescription" minOccurs="0"/>
            <xsd:element ref="TRI:ParentCompanyNameNAIndicator" minOccurs="0"/>
            <xsd:element ref="TRI:ParentCompanyNameText" minOccurs="0"/>
            <xsd:element ref="TRI:ParentDunBradstreetCode" minOccurs="0"/>
            <xsd:element ref="TRI:FacilityDunBradstreetCodeDetails" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <!--information removed for example purposes -->
</xsd:schema>
```

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

Rules and Guidelines	
Rule	**Description**
[SD5-A]	Developers MUST use SSC in schema when they are an appropriate fit for the targeted business process.
[SD5-B]	Developers SHOULD create custom datatypes and elements only after determining that no existing SSC adequately describes the given construct.
[SD5-C]	Existing schema SHOULD be evaluated for SSC compatibility and subsequently updated to include SSC elements and/or datatypes at the time of the next revision.
[SD5-D]	Developers SHOULD use XML datatype extension and restriction to modify SSC types.

3.6. Schema Versioning

NOTE: The Exchange Network Versioning strategy has not been finalized at the time of this writing, although the guidelines provided in this section are consistent with the strategy as it exists to date. Note that there may be a revision to this guidance in the future.

There are several components to the Exchange Network schema versioning strategy:

1. Schema namespace naming conventions
2. Schema file naming conventions
3. Use of the built-in XML schema version attribute
4. The use of a developer-defined *schemaVersion* attribute in the root element of each message schema in the namespace
5. Guidelines as to what constitutes a major or minor version change in a schema
6. Configuration of the Exchange Network schema repository

Each component is elaborated upon in the following section.

The role of namespace in schema versioning
As described in the *Section 3.4.1 Namespaces in Exchange Network Schema*, the namespace name must contain only the major version number of the schema. This is important to maintain compatibility between minor versions of schema. Since older minor version instance files must continue to validate against newer minor versions of the schema, the namespace must be retained between minor versions. The namespace naming rules set forth in Section 3.4 are consistent with the Exchange Network schema versioning strategy.

Because namespaces only contain a major version number, the namespace alone can not be used to readily identify the exact version of a given schema or instance document. This is where other aspects of schema versioning become important, as described below.

The role of file naming conventions in schema versioning
As described in Section *3.1.4*, schema file names must contain both a major and minor version number. This will enable a schema developer to easily identify the version of a schema by examining the file name. The file naming rules set forth in section *3.1.4* are consistent with the Exchange Network schema versioning strategy.

The role of the built-in XML schema "version" attribute in schema versioning
The W3C Schema specification allows for a "version" attribute to be defined in the root element of an XML schema. The DRC v1.0 specifies in rule [SD5-20] that schemas MUST include a version number using the W3C Schema version attribute. This rule has been retained and is consistent with the Exchange Network schema versioning strategy. In the example below, the built-in schema version attribute is used to indicate the schema version:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:NPDES="http://www.exchangenetwork.net/schema/NPDES"
xmlns:xs="http://www.w3.org/2001/XMLSchema"  version="1.1">
    <!—information removed for example purposes—>
</xsd:schema>
```

While the built-in version attribute is very straightforward for a human reader, it is not natively enforced by an XML parser. Furthermore, the value of this attribute as defined in the schema is never visible in a given XML instance document. This requires that yet another strategy be used to ensure that a given instance document directly references a specific schema version. The developer-defined *schemaVersion* attribute addresses this need as described below.

The role of the developer-defined *schemaVersion* attribute in schema versioning

Because namespace alone cannot be used to identify the schema version against which an instance document will validate, a separate mechanism has been established to identify the version of a schema that the instance document targets. Developers must create a required attribute in the root element of each message schema named *schemaVersion*. This attribute will then clearly define the schema version which the instance document targets.

In the example below, usage of the *schemaVersion* attribute is illustrated:

```
<xsd:element name="TRI">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="TRI:TRIDataType">
                    <xsd:attribute name="schemaVersion" use="required">
                        <xsd:simpleType>
                            <xsd:restriction base="xsd:decimal">
                                <xsd:pattern value="2\.\d*"/>
                            </xsd:restriction>
                        </xsd:simpleType>
                    </xsd:attribute>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
```

In this example, the *schemaVersion* attribute has been restricted using a pattern (i.e. regular expression). The pattern requires that the instance document contain a value for the *schemaVersion* attribute in the format "2.*x*" where *x* is one or more digits. The pattern is not required by the design rules, however if it is to be used, it is recommended to fix the major version number while allowing variations in the minor version component.

The following example displays portion of an instance document that properly implements this attribute and pattern:

```
<TRI xmlns="http://www.exchangenetwork.net/schema/TRI/2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
schemaVersion="2.1">
    <!—information removed for example purposes—>
</TRI>
```

Major versus minor version changes in a schema

In order for a schema change to be considered a minor version change, the following must be true:

- The namespace must remain unchanged from previous minor versions
- Only new, optional elements can be added
- Existing required elements may be made optional
- Restrictions may be loosened, such as eliminating a facet restriction or pattern from an existing element

If revisions to a schema are exceed those listed above, then the schema must be given a new major version number. The namespace must also be changed to reflect the new major version number.

To test whether a schema has met the criteria for a minor version, attempt to validate a prior-version instance document against the revised schema. This will require modifying the *schemaLocation* attribute in the instance document to point to the location of the new schema. To best ensure compatibility, the test instance document must be fully populated with data for all elements and attributes defined in the prior-version schema[7].

A second test may be performed to ensure that no elements which were previously optional have become required in the new schema. This test should utilize an instance document containing only the required elements from the prior-version schema. Again, attempt to validate this document against the latest schema. If it fails validation, the new schema requires elements to be present which were not required in the prior-version schema. This would violate the rule for a minor version schema change.

If schema versioning rules are followed, then in theory receivers of XML instance documents can continue to validate, parse and utilize new minor version of instance documents using toolsets built against previous minor versions of the schema. This gives data exchange implementers more flexibility in choosing when to upgrade parsing routines for inbound XML documents. However, these parsing routines will not be able to take advantage of new data elements added to the latest minor version of the schema.

Configuration of the Exchange Network schema repository

The Exchange Network schema repository will be structured in a manner which will always return the root schema for the latest production minor schema version in a given namespace. For example, assume the latest version of a given schema is 1.2 and the namespace is defined as http://www.exchangenetwork.net/schema/myflow/1. When the namespace URL is resolved, the repository will return http://www.exchangenetwork.net/schema/myflow/1/index.xsd. Index.xsd will reference the Message Schema(s) associated with the latest version, 1.2. If the schema are correctly versioned according to the rules defined in this document, any instance document (regardless of the minor version) will validate against the schema located in the namespace URL.

A broader discussion of the Exchange Network repository can be found in the *Guidance on Namespace Organization, Naming, and Schema File Location v1.11* document referenced in Appendix C of this document. While no specific schema design rules hinge on this information, it is useful to understand how schema are stored in the Exchange Network schema repository.

Rules and Guidelines	
Rule	**Description**
[SD5-21]	The W3C Schema version attribute MUST include both a major version component and a minor version component for each schema file root.
[SD5-22]	Data-centric schemas MUST include a major and minor version number in their filename.
[SD5-26]	Data-centric schemas MUST define a required attribute named "schemaVersion" in the root element of all message schema.

[7] Tools such as Altova XMLSpy can generate a sample instance document with sample data for all elements and attributes in a given schema. This document can then be validated against the newer schema to ensure compatibility.

[SD5-E]	New minor versions of schema MUST be able to validate instance documents created with preceding minor versions of that schema. However, instance documents should not be expected to validate against versions of schema preceding the one they were created with.
[SD5-F]	New minor versions of a schema MUST only add new optional elements and/or attributes to prior minor versions.
[SD5-G]	New minor versions of a schema MUST NOT remove elements and/or attributes from prior minor versions.
[SD5-H]	New minor versions MUST utilize the exact same namespace as prior minor versions.
[SD5-I]	All existing instance documents in the same namespace MUST validate against the new minor version.
[SD5-J]	The schema major version MUST be incremented if any elements or attributes are removed or if new mandatory elements or attributes are added.
[SD5-K]	The schema file name, XSD version attribute, header documentation, and namespace MUST all contain matching version information.
[SD5-L]	When any schema construct is altered in a given namespace, all schema in the namespace MUST undergo a version increment.

3.7. Exchange Network Header

The Exchange Network Header provides information to identify the contents of a data payload. The Header was developed to further automate the data exchange process so that data can be more readily identified during transport and at its processing destination. The generic nature of the Header gives it the flexibility to meet the needs of many operational scenarios.

The Header describes the sender, receiver, and message metadata. Metadata includes information such as what a data payload contains, as well as instructions on processing payload contents, such as whether the contents are additions, deletions, or updates.

The following example illustrates the implementation of the Header in an example RCRA XML instance document:

```
<Document xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.exchangenetwork.net/schema/v1.0/ExchangeNetworkDocument.xsd">
    <Header>
        <Author>John Doe.</Author>
        <Organization>State Environmental Commission</Organization>
        <Title>HandlerSubmission</Title>
        <CreationTime>2004-09-09T14:05:20.4259708-07:00</CreationTime>
        <DataService>HandlerSubmission</DataService>
        <Sensitivity>Unclassified</Sensitivity>
        <Property>
            <name>RCRAInfoUserID</name>
            <value>JIL</value>
        </Property>
        <Property>
            <name>RCRAInfoStateCode</name>
            <value>MI</value>
        </Property>
    </Header>
    <Payload Operation="RCRA-Transactional|HD">
        <!—payload instance document removed for example purposes—>
    </Payload>
</Document>
```


The Header is independent of any particular data exchange. If a schema developer chooses to implement the Header in a given data exchange, the Header schema file should not be included with the schema package. Many flows do not utilize the Header, and use of the header is not required.

Some data exchange scenarios require that the submitter include metadata about whether the receiver is to insert, update or delete data in the receiving system based on the message contents. Two common approaches exist for specifying this information. Some developers choose to include a *TransactionType* element on each “record” in the XML document indicating the action to be taken by the receiver. Alternatively, the Header can be used to separate transactions into different payloads, each indicating a different transaction type. For example, the first payload may represent inserts and updates while the second payload represents data to be deleted in the receiving system. The *Payload* element’s *Operation* attribute may be used to indicate the operation to be performed.

Note that since the Header was created before many of the current design rules were enacted; it does not comply with many of design rules outlined in this document. The header may be revised

in the near future to align it with current design rules and to meet a more structured implementation guideline.

Rules and Guidelines	
Rule	**Description**
[SD5-P]	Schema developers MAY implement the EN Header as a method of adding metadata to one or more payloads.
[SD5-Q]	The Header MAY be used to include message metadata (sender identification, transaction type such as INSERT or DELETE, and other message-level parameters) or to bundle multiple XML messages into a single XML document.

Appendix A – Summary of XML Rules

This table contains the complete list of updated XML design rules and conventions. Note that document-centric rules from DRC v1.0 have been omitted. New rules are printed in dark blue bold print. The Rule ID indexing convention uses the same categorization approach as the DRC v1.0 (i.e. SD*x* or GD*x*) however rules are now are post-fixed with an alpha character. Blue, bold rules which do not have an alpha-character postfix are a modified version of the original rule or guideline.

Topic	Sec.	Rule ID	Rule
General XML Design			
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 datatypes.
		[GD1-15]	EPA messages shall reuse registered datatypes to the maximum extent practicable.
		[GD1-16]	In a schema, information that expresses associations between data elements in different classification schemes (in other words, "mappings") MAY be regarded as metadata. This information should be accessible in the same manner as the rest of the information in the schema.

Topic	Sec.	Rule ID	Rule
	3.1.1	**[GD1-A]**	**Elements with recurring, complex content SHOULD NOT be referenced more than once within the same root document, even if the element exists in different schema files. When the same structure needs to be reused in multiple areas, either create a new element with a different tag name utilizing a common datatype or use the KEY and KEYREF construct to represent the element in a single list.**
	3.1.2	**[GD1-B]**	**Recurring elements (maxOccurs > 1) SHOULD be nested within their own container element.**
	3.1.3	**[GD1-C]**	**Wherever possible, component schema SHOULD NOT define any elements, only datatypes.**
	3.1.3	**[GD1-D]**	**Message-level root elements MUST be defined in the Message Schema.**
File Naming Rules and Guidelines			
File Naming Rules and Guidelines		[GD2-2]	File names MUST NOT use abbreviations unless their meaning is beyond question (EPA, GSA, FBI).
	3.2	**[GD2-A]**	**Each namespace must contain a default schema named "index.xsd" which contains only an include construct referencing the Root Schema.**
	3.2	**[GD2-B]**	**Root Schema MUST utilize the following naming format: *FlowName* [+ "_" + *FlowCategoryName*] + "_v" + Version + ".xsd".**
	3.2	**[GD2-C]**	**Message Schema MUST utilize the following naming format: *FlowName* [+ "_" + *FlowCategoryName*] + "_" + *MessageName* + "_v" + *Version* + ".xsd".**
	3.2	**[GD2-D]**	**Component Schema MUST utilize the following naming format: *FlowName* [+ "_" + *FlowCategoryName*] + "_" + *ComponentName* + "_v" + *Version* + ".xsd".**
	3.2	**[GD2-E]**	**Local Shared Schema MUST utilize the following naming format: *FlowName* [+ "_" + FlowCategoryName] + "_Shared" + "_v" + *Version* + ".xsd".**
XML Tag Naming Conventions			
XML Tag Naming Conventions		[GD3-1]	Element names MUST be in "Upper Camel Case" (UCC) convention, where UCC style capitalizes the first character of each word and compounds the name. Example: <UpperCamelCaseElement/>
		[GD3-2]	Schema type names MUST be in UCC convention. Example: <DataType/>
		[GD3-3]	Attribute names MUST be in "Lower Camel Case" (LCC) convention where LCC style capitalizes the first character of each word except the first word. Example: <UpperCamelCaseElement lowerCamelCaseAttribute="Whatever"/>
		[GD3-4]	Acronyms SHOULD NOT be used, but in cases where they are used, the capitalization SHALL remain Example: <XMLSignature/>, and the acronym SHOULD be defined in the comments of the DTD or Schema or in a separate document noted in the DTD or Schema as providing a tag dictionary so that the meaning of the acronym is clear.
		[GD3-5]	Abbreviations SHOULD NOT be used. In cases where they are used, they MUST be a major part of the federal or data standards vocabulary, and the abbreviation SHOULD be defined within the comments of the DTD or Schema or in a separate document (noted in the DTD or Schema) as providing a tag dictionary so that the meaning of the abbreviation is clear. An exception to this rule is when identifier is used as a representation term, ID SHOULD be used as part of the tag name.
		[GD3-6]	Underscores (_), periods (.) and dashes (-) MUST NOT be used.
		[GD3-7]	Verbosity in tag length SHOULD be limited to what is required to conform to the Tag Name Content

Topic	Sec.	Rule ID	Rule
			recommendations. When tags will be used in database structures, a limit of 30 characters is recommended.
Tag Name Content (Semantic Guidelines)	**3.3**	**[GD3-8]**	**Element, attribute, and datatype tag names MUST be unique.**
		[GD3-10]	High-level parent element tag names SHOULD consist of a meaningful aggregate name followed by the term "Details". The aggregate name may consist of more than one word. Example: <SiteFacilityDetails/>
		[GD3-11]	Tag names SHOULD be concise and MUST NOT contain consecutive redundant words.
		[GD3-12]	Lowest level (it has no children) element tag name SHOULD consist of the Object Class, the name of a Property Term, and the name of a Representation Term. An Object Class identifies the primary concept of the element. It refers to an activity or object within a business context and may consist of more than one word. Example: <LocationSupplementalText/>
		[GD3-13]	A Property Term identifies the characteristics of the object class. The name of a Property Term SHALL occur naturally in the tag definition and may consist of more than one word. A name of a Property Term shall be unique within the context of an Object Class but may be reused across different Object Classes. Example: <LocationZipCode/> and <MailingAddressZipCode/> may both exist.
		[GD3-14]	If the name of the Property Term uses the same word as the Representation Term (or an equivalent word), this Property Term SHALL be removed from the tag name. In this case, only the Representation Term word will remain.
		[GD3-15]	A Representation Term categorizes the format of the data element into broad types. A list of UN/CEFACT Representation Terms is included at the end of this list of rules, but the EPA and its partners may need to augment this list to accommodate the specific needs for environmental data. When possible the pre-defined UN/CEFACT list SHOULD be used. Proposed additions should be submitted to the TRG for consideration.
		[GD3-16]	The name of the Representation Term MUST NOT be truncated in the tag name.
		[GD3-17]	A tag name and all its components MUST be in singular form unless the concept itself is plural.
		[GD3-18]	Non-letter characters MUST only be used if required by language rules.
		[GD3-19]	Tag names MUST only contain verbs, nouns and adjectives (no words like "and", "of", "the").
	3.3	**[GD3-A]**	**All datatype names MUST end with either "Type" or "DataType".**
Datatypes			
Simple Datatypes		[SD2-1]	Data-centric schemas MUST use simple datatypes to the maximum extent possible.
Global Complex Datatypes		[SD2-3]	Data-centric schemas that employ complex datatypes MUST define the complex datatypes as global.
Local Complex Datatypes		[SD2-5]	Data-centric schemas SHOULD NOT use local complex datatypes.
Elements and Attributes			
Global Elements		[SD3-1]	Data-centric schemas MUST use global elements.
Local Elements		[SD3-3]	Data-centric schemas SHOULD NOT use local 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.

Topic	Sec.	Rule ID	Rule
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.
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.
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.
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.
sequence Compositor		[SD3-22]	Data-centric schemas SHOULD use the "sequence" compositor.
choice Compositor		[SD3-24]	Data-centric schemas SHOULD use the "choice" compositor.
all Compositor		[SD3-26]	Data-centric schemas MUST NOT use the "all" compositor.
Model Groups		[SD3-28]	Data-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.
Namespaces			
Exchange Network Namespaces	**3.4.1**	**[SD4-A]**	**The schema namespace name MUST be URL-formatted as "http://www.exchangenetwork.net/schema/{category}[/sub-category]/Version".**
	3.4.1	**[SD4-B]**	**Exchange Network namespaces MUST contain a category term which clearly and uniquely defines the type of data being exchanged.**
	3.4.1	**[SD4-C]**	**Exchange Network namespaces MAY contain a subcategory term which further divides the data exchange into smaller components.**
	3.4.1	**[SD4-D]**	**Exchange Network namespaces MUST contain a major version number as the last part of the namespace name.**
	3.4.1	**[SD4-E]**	**Exchange Network namespaces MUST NOT contain a minor version number in the namespace name.**
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.
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.
The W3C Schema Datatypes Namespace		[SD4-11]	Data-centric schemas SHOULD NOT declare the W3C Schema Datatypes namespace.
Target Namespaces		[SD4-13]	Data-centric schemas MUST use target namespaces.
	3.4.2	**[SD4-F]**	**The schema's targetNamespace MUST match the namespace name of one of the declared**

Topic	Sec.	Rule ID	Rule
			namespace qualifiers, but not the w3c qualifier.
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.
Single/Multiple Namespaces		[SD4-21]	Data-centric schemas SHOULD use a multiple-namespace configuration.
Default Namespaces		[SD4-23]	Data-centric schemas MUST NOT use default namespaces.
Namespaces and Attributes		[SD4-25]	Data-centric schemas MUST use namespace qualification for all attributes.
Multiple Namespaces		[SD4-27]	Exchange Network schemas MAY use multiple namespaces.
		[SD4-30]	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.
	3.4.3	**[SD4-H]**	**If a schemaLocation is specified in an XML instance document, the location MUST match the namespace URL address.**
		[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.
Namespace Declaration and Qualification-XML Instance Documents		[SD4-43]	Data-centric XML instance documents MUST use namespace qualification for all elements.
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.
Local Namespace Declarations		[SD4-49]	Data-centric XML instance documents MUST NOT use local namespace declarations.
Schema Configuration and Documentation			
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.
Shared Exchange Network Schemas	**3.5**	**[SD5-A]**	**Developers MUST use SSC in schema when they are an appropriate fit for the targeted business process.**

Topic	Sec.	Rule ID	Rule
	3.5	**[SD5-B]**	**Developers SHOULD create custom datatypes and elements only after determining that no existing SSC adequately describes the given construct.**
	3.5	**[SD5-C]**	**Existing schema SHOULD be evaluated for SSC compatibility and subsequently updated to include SSC elements and/or datatypes at the time of the next revision.**
	3.5	**[SD5-D]**	**Developers SHOULD use XML datatype extension and restriction to modify SSC types.**
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.
Voluntary Standards Body Schemas		[SD5-10]	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.
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.
Code Lists		[SD5-18]	Exchange Network schemas SHOULD support code lists through multiple namespaced types.
Built-In Schema Version Attribute		[SD5-20]	Data-centric schemas MUST include a version number using the W3C Schema version attribute.
	3.6	**[SD5-21]**	**The W3C Schema version attribute MUST include both a major version component and a minor version component for each schema file root.**
	3.6	**[SD5-22]**	**Data-centric schemas MUST include a major and minor version number in their filename.**
User-Defined Version Attribute on Instance Root	**3.6**	**[SD5-26]**	**Data-centric schemas MUST define a required attribute named "schemaVersion" in the root element of all message schema.**
Schema Construct Documentation	**3.1**	**[SD5-28]**	**All schema SHOULD include schema construct documentation using the W3C documentation element. Documentation SHOULD only describe the element or datatype and SHOULD NOT contain lists of acceptable codes, implementation details or other information not directly related to the meaning of the construct.**
		[SD5-29]	Data-centric schemas SHOULD use the documentation element for schema construct documentation.
	3.1	**[SD5-30]**	**HTML-style comments (<!--comment -->)SHOULD NOT be used in schema.**
Minor Version Changes	**3.6**	**[SD5-E]**	**New minor versions of schema MUST be able to validate instance documents created with preceding minor versions of that schema. However, instance documents should not be expected to validate against versions of schema preceding the one they were created with.**
	3.6	**[SD5-F]**	**New minor versions of a schema MUST only add new optional elements and/or attributes to prior minor versions.**
	3.6	**[SD5-G]**	**New minor versions of a schema MUST NOT remove elements and/or attributes from prior minor versions.**
	3.6	**[SD5-H]**	**New minor versions MUST utilize the exact same namespace as prior minor versions.**

Topic	Sec.	Rule ID	Rule
	3.6	**[SD5-I]**	**All existing instance documents in the same namespace MUST validate against the new minor version.**
Major Version Changes	**3.6**	**[SD5-J]**	**The schema major version MUST be incremented if any elements or attributes are removed or if new mandatory elements or attributes are added.**
Version Synchronization	**3.6**	**[SD5-K]**	**The schema file name, XSD version attribute, header documentation and namespace MUST all contain matching version information.**
		[SD5-L]	**When any schema construct is altered in a given namespace, all schema in the namespace MUST undergo a version increment.**
Schema Header Documentation		[SD5-34]	Data-centric schemas MUST include schema header documentation.
Exchange Network Header	**3.7**	**[SD5-P]**	**Schema developers MAY implement the EN Header as a method of adding metadata to one or more schema payloads.**
	3.7	**[SD5-Q]**	**The Header MAY be used to include message metadata (sender identification, transaction type such as INSERT or DELETE, and other message-level parameters) or to bundle multiple XML messages into a single XML document.**
Information Association and Uniqueness			
ID/IDREF Technique		[SD6-1]	Data-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.
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.
XLink/XPointer Technique		[SD6-15]	Data-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.
Advanced W3C Schema Concepts			
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.

Topic	Sec.	Rule ID	Rule
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).
Union Technique		[SD7-11]	Data-centric schemas MAY use the union technique.
Complex Datatype Restriction		[SD7-13]	Data-centric schemas MAY use complex datatype restriction.
Complex Datatype Extension		[SD7-15]	Data-centric schemas MAY use complex datatype extension.
Prohibiting Complex Datatype Derivation		[SD7-17]	Data-centric schemas MAY use the final attribute derivation.
Prohibiting Use of Derived Complex Datatypes		[SD7-19]	Data-centric schemas MAY use the block attribute.
Abstract Datatypes		[SD7-21]	Data-centric schemas MUST NOT use abstract datatypes.
Wildcards		[SD7-23]	Data-centric schemas MUST NOT use wildcards.
Default Element Values		[SD7-25]	Data-centric schemas SHOULD NOT use default element values.
Fixed Element Values		[SD7-27]	Data-centric schemas SHOULD NOT use fixed element values.
Default Attribute Values		[SD7-29]	Data-centric schemas SHOULD NOT use default attribute values.
Fixed Attribute Values		[SD7-31]	Data-centric schemas SHOULD NOT use fixed attribute values.
Substitution Groups		[SD7-33]	Data-centric schemas MUST NOT use substitution groups.
W3C Schema appinfo Element		[SD7-38]	Data-centric schemas MUST NOT use the appinfo element.
Notations		[SD7-40]	Data-centric schemas MUST NOT use notations.

Appendix B – Deprecated Design Rules

The following rules have been either removed or modified. An explanation of why each rule has been omitted or modified is included below.

Rule ID	Rule Description	Modification	Reason
[GD2-1]	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).	deleted	While the basic composition of file names is unchanged, new rules allow for naming variation based on schema type.
[GD1-3]	Message-level schemas SHOULD have their versions changed when a referenced external modular schema is updated.	Deleted	Replaced by [SD5-L] which clarifies the rule.
[GD3-8]	Element, attribute, and datatype tag names SHOULD be unique.	Updated	Updated to MUST
[GD3-9]	Element tag names MUST be extracted from the Environmental Data Registry (EDR) where possible.	Deleted	The EDR is still a valuable resource for environmental data standards, the shared schema components (SSCs) are the primary source for tag names. New tags should use the UN/CEFACT naming methods.
[SD4-7]	Data-centric schemas SHOULD use "xsd" as a namespace prefix for all W3C Schema constructs	Deleted	Prefix “xsd” can add 10% or more to schema file size. By removing this rule, developers may use any namespace prefix or none at all.
[SD4-28]	Exchange Network schemas MUST use urn:us:net:exchangenetwork as the target namespace.	Deleted	Superseded with new naming conventions
[SD4-29]	EPA schemas MUST use urn:us:gov:epa as the target namespace.	Deleted	Superseded with new naming conventions
[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.	Updated	Clearer rule provided by new namespace guidance
[SD5-4]	Shared Exchange Network Schemas MUST be used when available.	Deleted	Vague. SSC guidelines are much clearer
[SD5-5]	Data-centric Shared Exchange Network Schemas SHOULD use a target namespace identifier of "urn:us:gov:epa".	Deleted	Superseded by new namespace guidance
[SD5-21]	The version number MUST include both a major version component and a minor version component.	Updated	Clarified to mention schema version attribute
[SD5-22]	Data-centric schemas SHOULD include a version number in their filename.	Updated	Updated to MUST
[SD5-26]	Data-centric schemas MUST define a schema version attribute for use on the instance root.	Updated	Clarified to include exact syntax and make attribute required
[SD5-28]	Data-centric schemas SHOULD include schema construct documentation.	Updated	Added detail about what type of information should be included in schema documentation.
[SD5-30]	Data-centric schemas MAY use DTD-style comments for comments pertaining to the structure of the schema.	Updated	Changed to SHOULD NOT

Appendix C – References

- XML Design Rules & Conventions for the Exchange Network version 1 (2003)

  http://www.exchangenetwork.net/dev_schema/EN_xml_drc_v1.0.pdf

- Guidance on Namespace Organization, Naming, and Schema File Location

  http://www.exchangenetwork.net/dev_schema/Network_Namespace_v1.11.pdf

- Shared Schema Components (SSC) Usage Guide 1.0

  http://www.exchangenetwork.net/registry/SharedSchemaComponents-UsageGuide.pdf

- Shared Schema Components (SSC) Technical Reference 1.0

  http://www.exchangenetwork.net/registry/SchemaSchemaComponents-TechnicalReference.pdf

- Data Exchange Design Guidance and Best Practices Document v1.1

  http://www.exchangenetwork.net/dev_schema/exchange_design_guidance_v1.1.pdf

- Network Operations Board (NOB) Decision Memoranda
  - 2005-02: Shared Schema Component Usage
    - http://www.exchangenetwork.net/operations/nob/DM_2005_02_SSCUsage_final.doc
  - 2006-01: Exchange Network XML Namespaces
    - http://www.exchangenetwork.net/operations/nob/DM_2006_01_Namespace.pdf

- Exchange Network Frequently Asked Questions (FAQ) – Header

  http://test.epacdxnode.net/faq/ch03s12.html