XML Design Guidelines

Issue 1_0
11 December 2003

Table of Contents

1 Document Management

1.1 Legal Disclaimer

RosettaNet™, its members, officers, directors, employees, or agents shall not be liable for any injury, loss, damages, financial or otherwise, arising from, related to, or caused by the use of this document or the specifications herein, as well as associated guidelines and schemas. The use of said specifications shall constitute your express consent to the foregoing exculpation.

1.2 Copyright

1.3 Trademarks

RosettaNet, Partner Interface Process, PIP and the RosettaNet logo are trademarks or registered trademarks of "RosettaNet," a non-profit organization. All other product names and company logos mentioned herein are the trademarks of their respective owners. In the best effort, all terms mentioned in this document that are known to be trademarks or registered trademarks have been appropriately recognized in the first occurrence of the term.

1.4 Document Version History

Version	Date	Description
1_0	11 Dec 2003	Issued for Publication

1.5 The Modular PIP Production Process

The PIP production process is explained in the following diagram:

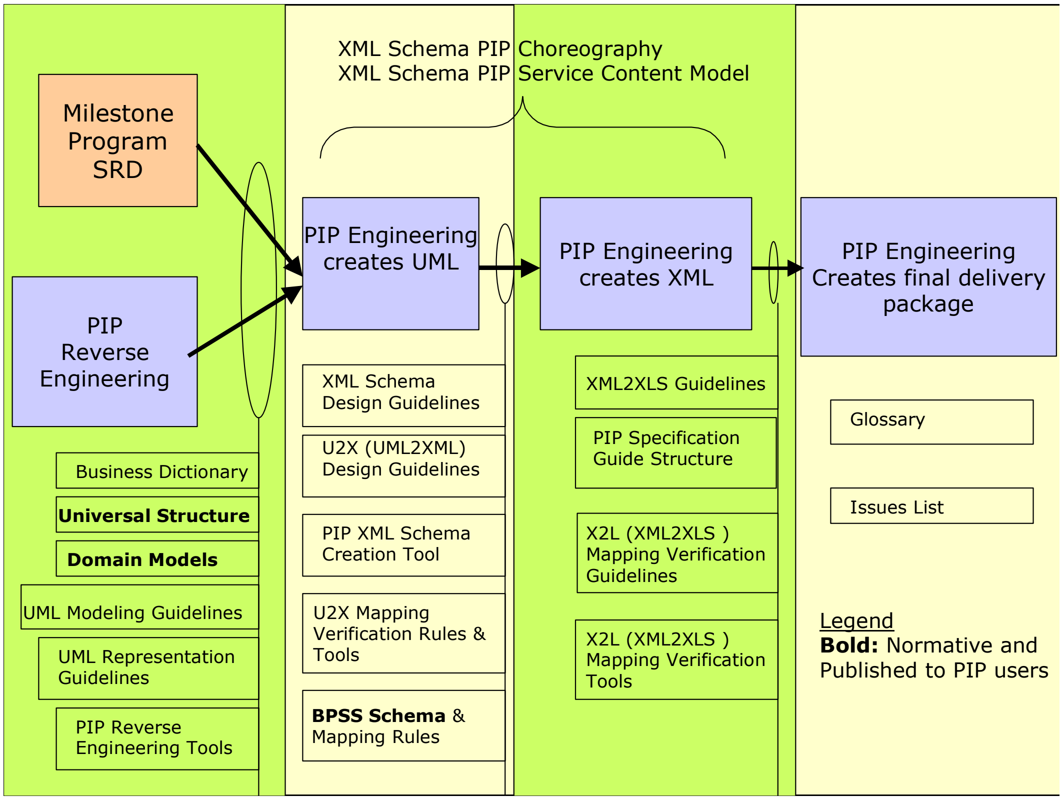
XML Schema PIP Choreography
XML Schema PIP Service Content Model
Milestone Program SRD
PIP Reverse Engineering
PIP Engineering creates UML
PIP Engineering creates XML
PIP Engineering Creates final delivery package
Business Dictionary
Universal Structure
Domain Models
UML Modeling Guidelines
UML Representation Guidelines
PIP Reverse Engineering Tools
XML Schema Design Guidelines
U2X (UML2XML) Design Guidelines
PIP XML Schema Creation Tool
U2X Mapping Verification Rules & Tools
BPSS Schema & Mapping Rules
XML2XLS Guidelines
PIP Specification Guide Structure
X2L (XML2XLS) Mapping Verification Guidelines
X2L (XML2XLS) Mapping Verification Tools
Glossary
Issues List
Legend
Bold: Normative and Published to PIP users

Figure 1: The Modular PIP Production Pipeline

Notes on Figure 1:

1. Note that the SRD has a different color than pale blue to signify it as a non PIP engineering artifact.
2. The different stages are differentiated by the green and light-pink stripes.
3. **Bold** indicates an end user deliverable.

1.6 Audience

This document’s primary audience is the UML to XML tool developers of RosettaNet, Solution Providers and PIP implementers.

1.7 Document Conventions

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119] as quoted here:

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

XSD *Refers to XML Schema Definition language*

Schema *Refers to XML Schema document compliant with W3C XML Schema Recommendations.*

xs *Refers to W3C XML Schema namespace*

xsi *Refers to XML Schema instance namespace.This is a separate namespace for four schema-related attributes that may appear in instances. These attributes, whose names are commonly prefixed with xsi, are: type, nil, schemaLocation, and noNamespaceSchemaLocation.*

Schema Component *Refers to the building blocks of the Schema like elements, types, content models, model groups, annotation etc.*

1.8 Document Structure

This document includes the following information:

1) XML Schema Design Rules

   XML Schema Design Rules will be applied to all XML Schema generated by RosettaNet, including the creation of following types of artifacts:
   1. Universal Structure
   2. System Structure
   3. Domain Structure
   4. Interchange Structure

2) XML instance documents (PIP Action Messages) defining rules

1.9 Acknowledgements

RosettaNet acknowledges the following individuals for contributing towards this document.

Content contributors:

Suresh Damodaran (on loan from Sterling Commerce)
Hussam El-leithy (RosettaNet)
Frank Yang (RosettaNet)
Nikola Stojanovic (RosettaNet)
Nidhi Gupta (on loan from Nokia)
Shishir Saxena (on loan from IBM)

Reviewers:

Barbara Heikkenen (on loan from Nokia)
Kenji Nagahashi (on loan from Fujitsu)
Shiv Rao (RosettaNet Malaysia)
Annabelle Marlow (RosettaNet)

1.10 Approvals

Title	Name	Signature (or type name)	Date
Chief Technologist	Suresh Damodaran	Approved for publication use - SD	11 December 2003

2 Schema Design Philosophy

Reuse of Schema components is a significant objective of the design of RosettaNet XML Schema. To attain this objective, this document focuses on providing Schema design rules and guidelines while permitting extensibility. In this document, we do not try to repeat the XML Schema rules found in [XML Schema Primer (XSDP), XML Schema Structures (XSDS), and XML Schema Datatypes (XSDD)], except when such repetition enhances understanding of the rules.

3 XSD Document Structure

3.1 Prologue and Encoding declaration

3.1.1 Prologue

The XML declaration always appears on the first line of an XML document. The XML declaration is a mechanism that notifies the parser that the document is an XML document and that it conforms to a specific version of XML.

Rule 3-1

RosettaNet developers MUST specify XML prologue at the beginning of each Schema to eliminate any ambiguity that may arise in specific parser implementations. The RosettaNet Schemas SHOULD conform to XML version 1.0. [XML]

Rationale

As XML Schema is also an XML document, the XML declaration must always be present within a Schema.

3.1.2 Encoding Declaration

Rule 3-2

Either "UTF-8" or "UTF-16" MUST be used as the value for character set and encoding type for all Schema and other XML documents.

3.2 xs:schema element

Rule 3-3

"xs" or "xsd" namespace prefix MAY be used to indicate the usage of W3C XML Schema namespace in case when W3C XML Schema namespace is not the default namespace. These prefixes are reserved and MUST NOT be used for declarations binding to other namespaces.

Note

For explanation on XML Schema namespace as default namespace see Rule 5-18.

Rule 3-4

The attribute xs:targetNamespace of xs:schema MUST be specified for all Schema documents, and its value MUST conform to RosettaNet namespaces specified in the namespace specification document [Namespace Specification and Management (NSSM)].

Rule 3-5

"tns" namespace prefix SHOULD be used to indicate xs:targetNamespace when targetNamespace is not the same as the default namespace of the Schema.

Rule 3-6

Default namespace MAY be specified as an attribute of xs:schema element.

Note

A more detailed explanation on namespaces and namespace exposure can be found in Namespace.

Rule 3-7

The xs:elementFormDefault attribute of xs:schema MUST have the value "qualified" and the xs:attributeFormDefault attribute MAY have the value of either "qualified" or "unqualified".

Rule 3-8

The xs:version attribute of xs:schema MUST be present and its value MUST reflect the version of the Schema.

Note

A more detailed explanation on versioning can be found in Versioning.

Rule 3-9

Order of xs:schema attributes MUST be as follows: targetNamespace declaration, declaration binding “xs” namespace prefix, default namespace declaration, declaration binding “tns” prefix, any other declarations binding prefixes to other namespaces, elementFormDefault declaration, attributeFormDefault declaration and version declaration.

Example XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
                                                         ThresholdReleaseForecastNotificatio
n:xsd:schema:1.21
                                                       ThresholdReleaseForecastNotification:xsd:
schema:1.21

    <xs:element name="ThresholdReleaseForecastNotification"
type="itrfn:ThresholdReleaseForecastNotification"/>
</xs:schema>
```

Example XML Instance

```
<?xml version="1.0" encoding="UTF-8"?>
<ThresholdReleaseForecastNotification>
   ……………………………
</ThresholdReleaseForecastNotification>
```

3.3 Documentation

The xs:annotation element has two child elements – xs:documentation element for human readable user documentation and xs:appinfo element for machine readable documentation. A single xs:annotation element may contain multiple xs:documentation and xs:appinfo elements, in any order.

Rule 3-10

The xs:schema root element and all reusable components in the Schema MUST have xs:annotation defined.

Rule 3-11

All Schema annotations MUST be in English and within the xs:annotation element. Schema annotations SHOULD be both human readable and machine processable.

Rule 3-12
The documentation for a Schema component SHOULD be placed as close to the component as possible, in order to insure consistency between the documentation and Schema component declaration / definition and to provide for better understanding of the Schema.

Rule 3-13
Any constraints relevant to either the whole Schema or to an individual Schema component MUST be expressed in Schematron syntax [STRON] under the "Constraint" subelement of the xs:appinfo. Other application related information SHOULD be expressed as subelements of the xs:appinfo element.

Rationale
The recommended way to add comments, documentation and other application information in a Schema is by means of the xs:annotation element. This element can be added as a subelement to most Schema components and can also be placed anywhere at the top level of Schemas.

3.3.1 Schema Documentation

Rule 3-14
Any human readable information relevant to the whole Schema MUST be contained in an xs:documentation element, nested inside an xs:annotation element. The xs:annotation element MUST be immediately under the xs:schema root element. This information SHOULD contain:

Field Name	Element Name	Element Value	Requirement
Constraints	Constraint	Text description of the Schematron constraints in the xs:appinfo element that are applicable to the whole document.	optional
RosettaNet copyright information	Copyright	©2003 RosettaNet. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the inclusion of this copyright notice. Any derivative works must cite the copyright notice. Any public redistribution or sale of this publication or derivative works requires prior written permission of the publisher.	mandatory
Legal Disclaimer	Disclaimer	RosettaNet™, its members, officers, directors, employees, or agents shall not be liable for any injury, loss, damages, financial or otherwise, arising from,	mandatory

		related to, or caused by the use of this document or the specifications herein, as well as associated guidelines and schemas. The use of said specifications shall constitute your express consent to the foregoing exculpation.	
RosettaNet Reference Program	Program	Milestone or Foundation Program	mandatory
Purpose of Schema	Purpose	Text	mandatory

Rule 3-15

Any application related information relevant to the whole Schema MUST be contained in an xs:appinfo element, nested inside an xs:annotation element. The xs:annotation element MUST be immediately under the xs:schema root element. This information SHOULD contain:

Field Name	Element Name	Element Value	Requirement
Acronyms	Acronym	Name-Value pairs	optional
Constraints	Constraint	Schematron constraints that are applicable to the whole document.	optional
RosettaNet Context specification (describes the content of the Schema, e.g., universal structures, and its relationship with other Schemas)-	Context	Text	optional
Date of Creation	CreationDate	dd/mm/yyyy	mandatory
Keywords denoting relationship to other Schemas	Keyword	Text	optional
Date of Last Update	LastUpdateDate	dd/mm/yyyy	mandatory

```
<xs:annotation>
     <xs:documentation xml:lang="US_EN">
        <Copyright>©2003 RosettaNet. All rights reserved. No part of this publication may be
        reproduced, stored in a retrieval system, or transmitted, in any form or by any
        means, electronic, mechanical, photocopying, recording, or otherwise, without the
        inclusion of this copyright notice. Any derivative works must cite the copyright
        notice. Any public redistribution or sale of this publication or derivative works
        requires prior written permission of the publisher.
        <Disclaimer>RosettaNet™, its members, officers, directors, employees, or agents
        shall not be liable for any injury, loss, damages, financial or otherwise, arising
        from, related to, or caused by the use of this document or the specifications herein,
        as well as associated guidelines and schemas.  The use of said specifications shall
        constitute your express consent to the foregoing exculpation.
        <Program> MileStone/Foundational </Program>
        <Purpose> State the purpose here </Purpose>
     </xs:documentation>
     <xs:appinfo>
         <Constraint/>
     </xs:appinfo>
```

```
</xs:annotation>
```

3.3.2 Component Documentation

Rule 3-16

Any human readable information relevant to reusable types MUST be contained in an xs:documentation element, nested inside an xs:annotation element. This information SHOULD contain:

Field Name	Element Name	Element Value	Requirement
Constraints	Constraint	Text description of the Schematron constraints in the xs:appinfo element	optional
Purpose of Component	Purpose	Text	optional

Rule 3-17

Any application related information relevant to reusable types MUST be contained in an xs:appinfo element, nested inside an xs:annotation element. This information SHOULD contain:

Field Name	Element Name	Element Value	Requirement
Constraints	Constraint	Schematron constraints	optional
RosettaNet Context specification (describes the content of the Schema, e.g., universal structures, and its relationship with other Schemas)-	Context	Text	optional
Date of Creation	CreationDate	dd/mm/yyyy	mandatory
Definition of the component	Definition	Text	mandatory
Keywords denoting relationship to other Schemas and components	Keyword	Text	optional
Date of Last Update	LastUpdateDate	dd/mm/yyyy	mandatory
Version	TypeVersion	Versioning Scheme see:[Versioning]	mandatory

Rule 3-18

Only when the name of a reusable element is different than its default name (i.e. type name without the suffix) the reusable element SHOULD have its own documentation. This rule SHOULD also apply to any element defined within a complex type.

Rule 3-19

Component documentation for any lower level element SHOULD be defined only by “definition” where deemed necessary to enhance understanding.

Field Name	Element Name	Element Value	Requirement

Constraints	Constraint	Text description of the Schematron constraints in the xs:appinfo element	optional
RosettaNet Context specification (describes the content of the Schema, e.g., universal structures, and its relationship with other Schemas)-	Context	Text	optional
Date of Creation	CreationDate	dd/mm/yyyy	mandatory
Definition of the component	Definition	Text	mandatory
Keywords denoting relationship to other Schemas and components	Keyword	Text	optional
Date of Last Update	LastUpdateDate	dd/mm/yyyy	mandatory
Version	TypeVersion	Versioning Scheme see:[Versioning]	mandatory

```
<xs:annotation>
     <xs:appinfo xml:lang="US_EN">
        <Constraint> Schematron constraint if any</Constraint>
        <Context> Reusable type here </Context>
        <CreationDate> 20/06/2003 </CreationDate>
        <Keyword> Invoicing </Keyword>
        <LastUpdateDate> 20/06/2003 </ LastUpdateDate >
        <Definition> State the definition here </Definition>
        <TypeVersion> 0.14 </TypeVersion>
     </xs:appinfo>
</xs:annotation>
```

3.3.3 Codelist Documentation

Rule 3-20

Any human readable information relevant to codelists MUST be contained in an xs:documentation element, nested inside an xs:annotation element. This information SHOULD contain:

Field Name	Element Name	Element Value	Requirement
Constraints	Constraint	Text description of the Schematron constraints in the xs:appinfo element	optional
Purpose of codelist	Purpose	Text statement describing the codelist and stating its purpose	optional

Rule 3-21

Any application related information relevant to codelists MUST be contained in an xs:appinfo element, nested inside an xs:annotation element. This information SHOULD contain:

Field Name	**Element Name**	**Element Value**	**Requirement**
Constraints	Constraint	Schematron constraints	optional
RosettaNet Context specification (describes the content of the Schema, e.g., universal structures, and its relationship with other Schemas)-	Context	Text explanation of context and dependencies of the codelist	optional
Date of creation	CreationDate	dd/mm/yyyy	mandatory
Definition of the Codelist	Definition	Text	mandatory
Codelist identifier	Identifier	Identification Scheme	mandatory
Date of Last Update	LastUpdateDate	dd/mm/yyyy	mandatory
Registration Authority	RegisteredBy	Text name of the registration authority of the codelist	mandatory
Version	TypeVersion	Versioning Scheme see:[Versioning]	mandatory

```
<xs:annotation>
     <xs:appinfo xml:lang="US_EN">
        <Constraint> Schematron constraint if any </Constraint>
        <Context> Reusable type here </Context>
        <CreationDate> 20/06/2003 </CreationDate>
        <Identifier> Identification here </Identifier>
        <LastUpdateDate> 20/06/2003 </LastUpdateDate>
        <RegisteredBy> Registering agency </RegisteredBy>
        <TypeVersion> 1.1 </TypeVersion>
     </xs:appinfo>
</xs:annotation>
```

3.4 Component Ordering

Rule 3-22

Schemas MUST follow consistent physical placement and ordering rules for its constituent components.

Rationale

Consistent placement / ordering of components helps with human readability and debuggability of Schemas.

3.4.1 Placement of various Schema components

Rule 3-23

1. Logically related constructs SHOULD be placed together in the same file in order to support better abstraction, reusability and clarity.
2. Logically related constructs within the same file SHOULD be placed in close proximity to promote understanding.

3. The documentation for a Schema SHOULD be placed just after the top-level xs:schema element. The documentation for individual components as listed above SHOULD be placed immediately after the component name declaration / definition.
4. When not in violation of the previous rules, the following SHOULD be the desired order of global Schema components.

   Reusable global element(s),
   Global element named groups,
   Global reusable attributes,
   Global attribute named groups,
   Global simple types,
   Global complex types with sequence content model,
   Global complex types with choice content model,

All of these components are internally sorted alphabetically by names.

3.4.2 Ordering of components within Type definition

Rule 3-24

Within the type definition, the sequences, choice, groups and sub-content models SHOULD be ordered in alphabetical order. Also within each content model (like sequence, choice, groups etc) elements SHOULD be sorted in alphabetical order.
The only exception is in the order of attributes and attribute groups. In element declarations and type definitions, the attributes and attribute groups SHOULD be listed alphabetically at the end, after the content model and elements.

Rationale

This ordering scheme permits easy reading of Schemas for debugging purposes.

4 Reusing Schemas

4.1 Import

Rule 4-1

The xs:import element MUST contain the schemaLocation attribute that points to the imported schema(s) via relative paths with respect to the location where the current Schema is stored.

Rule 4-2

Import SHOULD be used where needed. Circular imports MUST be avoided. Duplicate imports SHOULD be avoided.

See Figure 1 below, where Schema A imports from Schema B twice. Figure 2 shows circular import where Schema B imports Schema A, Schema C imports Schema B and Schema A imports Schema C.

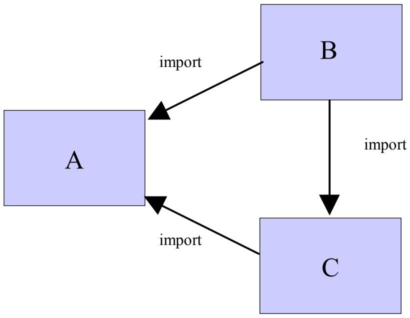

Figure 1: Duplicate Import

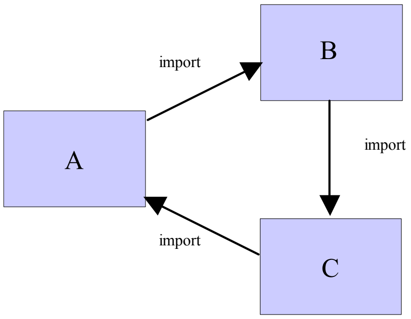

Figure 2: Circular Import

Rationale

An xs:import is used to refer to components from another namespace. When other XML Schemas are imported using xs:import, avoid the duplicate import trap shown in the picture. The symptom usually is, when validating Schema A, it could give “duplicate definitions” error in some parsers.

4.2 Include

Rule 4-3

xs:include MAY be used where needed.

Rationale

An xs:include is used when you want to include other Schemas in a Schema document that has the same target namespace. Include may find some use in modularization of Schemas.

4.3 Redefine

Rule 4-4

xs:redefine MUST NOT be used.

Rationale

A xs:redefine is similar to an include, with the additional option of specifying new definitions of some or all of the components in the redefined Schema. Besides of the possibly of changing the semantics of redefined definitions, xs:redefine might also cause conflicts when further modifications to redefined definitions are needed. Possibility of redefining already redefined definitions makes the usage of xs:redefine even more problematic.

5 Naming Conventions

5.1 General Naming Guidelines

5.1.1 Internationalization Features

Rule 5-1
The name of an XML Schema component MUST be an NCName (XML Name minus the ":").

Rule 5-2
All names MUST be composed of alphanumeric characters only. They MUST NOT include periods, hyphens, underscores, spaces or other separators.

Rule 5-3
The name of an XML Schema component MUST correspond to the name in UML model. This correspondence must be canonical and automate-able.

Rule 5-4
All Schema names and values created and maintained by RosettaNet SHOULD be understandable by an English speaking audience.

5.1.2 Acronyms

Rule 5-5
Acronyms SHOULD be written using uppercase. Word abbreviations SHOULD be avoided. Definition of an acronyms SHOULD be present in the corresponding Schema xs:appinfo element.

Example

```
<xs:element name="GTIN"/>

<xs:complexType name="GTINType"/>
```

Rationale
While it is unavoidable to use established acronyms, it is very helpful to include their definitions in the Schema in order to help with the understanding of their semantics.

5.2 Element

Rule 5-6
For element names, the Upper Camel Case ("UCC") convention MUST be used, i.e. the leading character of each word is capitalized. The remainder of each word is lower case.

Example

```
<xs:element name="PartnerDescription" type="PartnerDescriptionType"/>
```

Rule 5-7
While creating names for inner elements, concatenating the name of the inner element to the name of the outer element SHOULD be avoided. The exception to this rule is the following:

if the outer element name cannot be prefixed with *all* inner element names sensibly, then each inner element name SHOULD be created by concatenating the outer element name to it.

In the example below, both elements “Address” and “Phone” are placed inside the same context “Contact”; because of this, concatenating Contact with the Address and Phone is avoided.

Example

```
   <complexType name="ContactType">
      <complexContent>
         <extension base="us:SomeBaseType">
            <sequence>
               <element name="Address" type="xyz:AddressType"/>
               <element name="Phone" type="xyz:PhoneType"/>
            </sequence>
         </extension>
      </complexContent>
   </complexType>
   <element name="Contact" type="ContactType"/>
```

5.3 Attribute

Rule 5-8

For attribute names, the Lower Camel Case (“LCC”) convention MUST be used, i.e. the leading character of each word is capitalized except the first word, which starts with the lower case.

Example

```
<xs:attribute name="languageSupport" type=”xs:string”/>
```

5.4 Type

5.4.1 Named Types

Rule 5-9

All reusable, extendable, and restrictable types MUST be named. All such type names MUST be global in scope. Where reused, the new element MUST NOT have “name” attribute. Defining new elements for the same type SHOULD be avoided when RosettaNet has already defined an element for that type, and "ref" SHOULD be used to reuse an element. A new element MAY be declared when existing element name does not reflect a business term that is needed.

Example

```
<xs:complexType name="PartnerIdentificationType">
      <xs:sequence>
         <xs:element ref="PartnerIdentifier"/>
      </xs:sequence>
</xs:complexType>
<xs:element name="PartnerIdentifier" type="xs:string"/>
```

5.4.2 Naming Convention for Types

Rule 5-10

For type names, the Upper Camel Case ("UCC") convention MUST be used, i.e. the leading character of each word is capitalized. The complex type and simple type names MUST be written as component name (in UpperCamelCase) + Type, for example, TextualDescriptionType.

Example

```
<xs:simpleType name="MonetaryAmountType">
   <xs:restriction base="xs:nonNegativeInteger">
      <xs:totalDigits value="20"/>
   </xs:restriction>
</xs:simpleType>

<xs:complexType name="PhysicalAddressType">
   <xs:sequence>
      <xs:element name="AddressLine1" type="xs:string" minOccurs="0"/>
      <xs:element name="AddressLine2" type="xs:string" minOccurs="0"/>
      <xs:element name="AddressLine3" type="xs:string" minOccurs="0"/>
      <xs:element name="CityName" type="xs:string" minOccurs="0"/>
      <xs:element name="GlobalCountryCode" type="GlobalCountryCodeType" minOccurs="0"/>
      <xs:element name="NationalPostalCode" type="NationalPostalCodeType" minOccurs="0"/>
      <xs:element name="PostOfficeBoxIdentifier" type="xs:string" minOccurs="0"/>
      <xs:element name="RegionName" type="xs:string" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
```

5.5 Model Group

Rule 5-11

For model group names, the Upper Camel Case ("UCC") convention MUST be used, i.e. the leading character of each word is capitalized. The name MUST be written as group name (in UpperCamelCase) + Group, for example, TextualDescriptionGroup.

5.6 Namespace

Namespaces act as a mechanism to control and manage the extensible nature of the XML language. Namespaces resolve the problem of name collisions through a method of uniquely identifying Schema components with a prefix. This prefix is then associated to a Uniform Resource Name that truly guarantees unambiguous naming. More information on RosettaNet namespaces can be found in the namespace specification document [NSSM].

5.6.1 Namespace Convention

5.6.1.1 Namespace Prefix

Rule 5-12

Namespace prefix MAY be created by the first letters of the targetNamespace that appear between "specification" and "xml". If the abbreviation conflicts with other namespace prefixes, either integer suffices MAY be added (preferably based on version numbers), or additional letters MAY be added to make the namespace prefix unique within where it is used. The same namespace prefix SHOULD be reused in all the Schemas into which the Schema is imported.

Example

urn:rosettanet:specification:universal:ContactInformation:xsd:schema:2.0 may have a namespace prefix of "*uc".*

5.6.1.2 Relative URIs

Rule 5-13
Relative URI references MUST NOT be used in namespace declarations.

5.6.1.3 Uniform Resource Names

Rule 5-14
All reusable Schema components are considered RosettaNet Resources and MUST have a URN assigned to them.

Rule 5-15
Schema filename and targetNamespace URN MUST "canonically" match where for each targetNamespace there is one and only one file.

Rationale
Files are split when a single schema file contains multiple structures that may find independent use. This divergence in structures must be reflected in the namespace.

Rule 5-16
Schema targetNamespace URN SHOULD "canonically" match URN of one and only one of the Schema reusable types. This type is known as the "main type".

Rationale
If Schema contains only one reusable type definition then the name of that type is reflected in the namespace. If Schema contains more then one reusable type, but only one of them is used to define the root element of the instance document then the name of that type is reflected in the namespace.

Rule 5-17
Schema targetNamespace URN MAY "canonically" match URN of entities that convey logical grouping of resources.

Rationale
If Schema contains more then one reusable type definition then it is possible that those types are grouped logically based on some business or infrastructure classification. In that case the name of that classification group is reflected in the namespace.

Note
For further explanation of above rules consult the namespace specification document [NSSM].

5.6.1.4 Default Namespace

Rule 5-18
W3C XML Schema namespace MAY be the default namespace for any Schema.

Rule 5-19

xs:targetNamespace MAY be the default namespace for all Interchange Structure Schemas (E.g., PIP Schemas). Universal Structures and Domain Structure Schemas MUST NOT use xs:targetNamespace as the default namespace.

Rationale

Using default namespace provides better readability and more clarity for PIP Schemas. However, for Universal Structures and Domain Structure Schemas, the need to avoid accidental errors due to conflicting names in multiple namespaces takes priority, and therefore all elements are to be qualified with their namespace when used.

Example

For PIP Schemas

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://example.com"
      targetNamespace="http://example.com">
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="familyName" type="xs:string" />
        <xs:element name="firstName" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

For Universal Structures and Domain Structures

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:prod="http://example.com/prod"
      targetNamespace="http://example.com/prod">

  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="number" type="xs:integer" />
        <xs:element name="size" type="prod:SizeType" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

   <xs:simpleType name="SizeType">
   <!-- …  -->
   </xs:simpleType>
</xs:schema>
```

5.6.2 Namespace exposure

Rule 5-20

Namespaces of elements MUST be exposed in the XML instance files by setting elementFormDefault to “qualified” in the xs:schema. Namespaces of attributes MAY be exposed by setting attributeFormDefault attribute of the xs:schema element to “qualified”.

Example [Dare Obasanjo **(OBA)]**

This Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://example.com">
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="familyName" type="xs:string" />
        <xs:element name="firstName" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

validates the following document

```
<foo:person xmlns:foo="http://example.com">
  <familyName> KAWAGUCHI </familyName>
  <firstName> Kohsuke </firstName>
</foo:person>
```

which is unlikely what the Schema author intended. Altering the Schema to:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://example.com"
     elementFormDefault="qualified">
  <xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="familyName" type="xs:string" />
        <xs:element name="firstName" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

allows it to validate

```
<person xmlns="http://example.com">
  <familyName> KAWAGUCHI </familyName>
  <firstName> Kohsuke </firstName>
</person>
or
<foo:person xmlns:foo="http://example.com">
  <foo:familyName> KAWAGUCHI </foo:familyName>
  <foo:firstName> Kohsuke </foo:firstName>
</foo:person>
```

Rationale

Qualified attributeFormDefault is desirable when attributes from some other namespaces are also included. In other words, qualified attribute names are needed for those attributes that apply to a variety of elements in a variety of namespaces, such as xml:lang or xsi:type. For locally declared attributes, whose scope is only the type definition in which they appear, prefixes add extra text without any additional meaning.

The elementFormDefault and attributeFormDefault attributes determine whether to localize (hide) or expose the namespaces of elements and attributes within the XML instance documents.

Setting elementFormDefault and attributeFormDefault to "unqualified" ensures no namespace will be exposed within instance documents.

Setting elementFormDefault and attributeFormDefault to "qualified" ensures all namespaces will be exposed in instance documents.

5.6.3 Form Attribute

Form attribute can be used when control is required over whether an element or attribute should be qualified in instance documents.

Rule 5-21

RosettaNet Schema developers MUST NOT use the form attribute.

Rationale

The namespace exposure is determined by the global xs:elementFormDefault and xs:attributeFormDefault attributes for uniform look and feel of the XML Schemas.

6 Versioning

6.1 Versioning Philosophy

One basic rule for versioning is that any entity that has an independent lifecycle will have a version. Entities that are closely related and that are likely to have a lifecycle that is dependent on each other would have the same namespace and might be versioned together. The versioning scheme for files is closely aligned with PIP versioning scheme [PIP Development Guide (PIPDEV)]. The versioning for Schema components is described in namespace specification document [NSSM].

Rule 6-1
Schemas, namespaces and reusable types MUST have version numbers assigned to them.

6.1.1 Versioning Schemas

Schemas are versioned as all other entities. Version of a Schema is declared as explained in section 3.2.

Rule 6-2
The Schema version MUST match the version of the "main type" if the "main type" exists inside the Schema.

Note
For an explanation of the "main type" see Rule 5-16

6.1.2 Versioning namespaces

Sometimes namespaces contain multiple types that may change from one version of the namespace to another. If we want to identify the change, from one version to another, the contents need to be versioned and be independently identifiable. This allows faster change verification.

6.1.3 Relationship between Schema versions and namespace versions

Rule 6-3
The targetNamespace of a Schema MUST include the same number that matches the value of the built-in xs:schema "version" attribute.

Note
For an explanation of the "version" attribute see Rule 3-8

Example

```
urn:rosettanet:specification:universal:ContactInformation:xsd:schema:1.2
```

Rule 6-4
Major Schema version number MUST be changed when existing instance documents that validate against the current Schema cannot validate against the new Schema. Minor Schema version number MUST be changed when existing instance documents validate against the new Schema while new instance documents cannot validate against the existing Schema.

Rationale

This approach invalidates the instance documents when any change to Schema is introduced, which provides schema-based validation aid when old instances are incompatible with new schemas.

6.1.4 Versioning reusable types

Reusable types (simple types and complex types) are versioned independently. Versioning of types is independent of versioning of namespaces and versioning of Schemas.

Rule 6-5

"TypeVersion" element MUST be included under xs:appinfo element that annotates the reusable type.

Note

Usage of the "Type Version" element is explained in Component Documentation, Codelist Documentation and under the Rule 10-4.

Rule 6-6

"schemaVersion" attribute of the "token" type MUST be declared as an optional attribute for all reusable types.

Rationale

Reusable types and elements have unique identifiers within a namespace so that they can be referred to uniquely. This approach also indicates the fact that versioning of reusable types is independent of versioning of the Schema in which they reside. For further explanation of the application of this approach see Referencing Schemas from PIP Messages.

Note

More information on namespace versioning can be found in the namespace specification document. [NSSM] More information on packaging and versioning of Schemas can be found in the PIP Development Guide document. [PIPDEV]

7 Schema Construction Guidelines

XML Schema definition language gives many ways to express the same content in XML instance. The following sections give general guidelines regarding popular language constructs.

7.1 Use of XSD Built-In Types

Rule 7-1
The built-in types outlined in W3C XML Schema Datatypes [XSDD] SHOULD be used in designing Schemas as much as possible.

Rationale
The built-in types are well defined by the W3C Schema Datatypes specification [XSDD] and therefore unanimously understood by application developers. Creating RosettaNet types where W3C defined types can be used leads to confusion and misinterpretation during processing of data received in form of XML message.

7.2 Use of Element versus Attributes

The following characteristics of elements and attributes SHOULD be used to decide what is better as an attribute and what is better as an element.

1) attributes SHOULD only be used to specify meta-data. Meta-data provides context and facilitates processing of data. An example of meta-data is language (xml:lang)
2) attributes MUST NOT be used where further extensions of the attributes is required.
3) ordering is implementable only in elements and not in attributes
4) attributes need not be persistent.
5) attributes are less verbose. When values are lengthy, elements tend to be more readable than attributes. [Priscilla Walmsley (WAL)]
6) elements can be repeated [WAL]
7) elements can be used in substitution group [WAL]
8) elements can have nil values [WAL]
9) elements with all optional content SHOULD be avoided

7.3 Use of Content Model: sequence, choice, all.

The order and structure of the children of a complex type is known as its content model.

Rule 7-2
While composing groups of elements xs:sequence SHOULD be the preferred compositor, the use of xs:all is NOT RECOMMENDED. The xs:choice SHOULD be used if needed.

Rationale
The biggest disadvantage of xs:all is that it cannot be repeated any further. This limits the use of xs:all to the first occurrence of its set of elements. If a content model requires an element that occurs more than once then xs:all cannot be used.

Example XML Schema

```
   <xs:complexType name="ContactInformationType">
      <xs:sequence>
         <xs:element name="ContactName" type="xs:string"/>
         <xs:element name="EmailAddress" type="EmailAddressType" minOccurs="0"/>
         <xs:element name="FacsimileNumber" type="CommunicationsNumberType" minOccurs="0"/>
         <xs:element name="TelephoneNumber" type="CommunicationsNumberType" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
```

7.4 Reuse of Both Elements and Types

Rule 7-3
Schemas MUST define named global types (simpleType or complexType). Corresponding to the named global types, named global elements MUST be declared in all Schemas.

Rule 7-4
More then one global type definition and more then one global element declaration MAY be present in a Schema.

Note
This is a mixed approach of using Venetian Blind Design [MIT] and Garden of Eden [Universal Business Language Schema (UBLS)].

The Venetian Blind Design allows for maximum reuse of type definitions. Types are much easier to store in repository and reuse than elements.

The Garden of Eden allows declaration of reusable elements along with reusable types. The advantage of using reusable element is to avoid inconsistency in naming the elements of the same type. This will ensure uniform usage of element names corresponding to a particular type and will curb any misuse (for example, Order is of Company Type). There are some instances in PIP specifications where a structure (which is not a universal structure) is reused across PIPs, for example, PartnerProductForecast is used in PIPs 4A1, 4A2 and 4A3. It is useful in this situation to have a reusable element declared in the Domain Structure and reuse it instead of declaring three different element names corresponding to same complex type.

Example XML Schema

```
  <xs:element name="LocationIdentification" type="LocationIdentificationType"/>
  <xs:complexType name="LocationIdentificationType">
    <xs:sequence>
      <xs:element name="LocationIdentifier" type="LocationIdentifierType"/>
      <xs:element name="IdentifierAuthorityCode" type="IdentifierAuthorityCodeType"

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

7.5 Representing relationships

7.5.1 Use of Named Model Groups

Rule 7-5
The xs:group MAY be used when there is a need to reuse a set of elements when application design requires presentation to be structured. xs:group provides code reuse whereas type definitions provide definition reuse. xs:group SHOULD only be created when you need to group logically related content models.

Rationale

Schemas allow for grouping of elements and attributes. Grouping is performed using the xs:group element. Groups represent a set of element declarations or attribute declarations so that they can be incorporated as a group into complex type definitions. xs:group must be defined globally in order to be reused within a Schema. This might not be acceptable in terms of the overall design.

Example XML Schema

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

   <xs:element name =  "Customer">
      <xs:complexType>
         <xs:group ref =  "NameGroup"/>
      </xs:complexType>
   </xs:element>

   <xs:group name = "NameGroup">
      <xs:sequence>
         <xs:element name =  "FirstName" type = "xs:string" />
         <xs:element name =  "MiddleInitial" type = "xs:string" />
         <xs:element name =  "LastName" type = "xs:string" />
      </xs:sequence>
   </xs:group>

</xs:schema>
```

7.5.2 Extensibility

Rule 7-6

Extensibility SHOULD be implemented using XML Schema extension and restriction. Element substitution MAY be used carefully when required for this purpose.

Rule 7-7

For extensibility of RosettaNet Schemas, a Schema change request MUST be submitted to RosettaNet.

7.5.2.1 Inheritance via Extension

Rule 7-8

Complex type extension SHOULD be used. It is not possible to extend the value space of a simple type using extension.

7.5.2.2 Inheritance via Restriction

Rule 7-9

Simple type restriction SHOULD be used. Use of complex type restriction is discouraged, as it is complex. [OBA]

Example XML Schema

```
   <xs:simpleType name="MonetaryAmountType">
      <xs:restriction base="xs:nonNegativeInteger">
         <xs:totalDigits value="20"/>
      </xs:restriction>
   </xs:simpleType>
```

7.5.3 Use of abstract type and substitution groups

Both element declarations and complex type definitions can be made abstract. An abstract element declaration cannot be used to validate an element in an XML instance document and can only appear in content models via substitution. An abstract complex type definition similarly cannot be used to validate an element in an XML instance document; but it can be used as the abstract parent of an element's derived type or in cases where the element's type is overridden in the instance using xsi:type. [OBA]

Rule 7-10
The abstract complex type definitions MAY be used in RosettaNet Schemas as needed.

The following example from [MIT] illustrates the use of abstract complex type.

Example XML Schema

```
<xs:complexType name="PublicationType" abstract="true">
    ...
</xs:complexType>
<xs:complexType name="BookType">
        <xs:complexContent>
            <xs:extension base="PublicationType" >
                ...
            </xs:extension>
        </xs:complexContent>
</xs:complexType>
<xs:complexType name="MagazineType">
        <xs:complexContent>
            <xs:restriction base="PublicationType">
                ...
            </xs:restriction>
        </xs:complexContent>
 </xs:complexType>
 <xs:element name="Catalogue">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Publication" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
 </xs:element>
```

Example XML Instance

```
<?xml version="1.0" encoding="UTF-16"?>
<Catalogue xmlns="http://www.catalogue.org"
                   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:schemaLocation="http://www.catalogue.org Catalogue.xsd">
        <Publication xsi:type="BookType">
                <Title>Illusions The Adventures of a Reluctant Messiah</Title>
                <Author>Richard Bach</Author>
                <Date>1977</Date>
                <ISBN>0-440-34319-4</ISBN>
                <Publisher>Dell Publishing Co.</Publisher>
        </Publication>
        <Publication xsi:type="MagazineType">
                <Title>Natural Health</Title>
                <Date>1999</Date>
        </Publication>
        <Publication xsi:type="BookType">
                <Title>The First and Last Freedom</Title>
                <Author>J. Krishnamurti</Author>
                <Date>1954</Date>
                <ISBN>0-06-064831-7</ISBN>
                <Publisher>Harper Row</Publisher>
        </Publication>
</Catalogue>
```

<u>Use of substitution group</u>

Substitution groups are a flexible way to designate element declarations as substitutes for other element declarations from other Schemas or other namespaces without changing the original content model. Substitution groups are useful for simplifying content models, making choice groups more flexible, and allowing more descriptive elements to be used, including localized names. The members of substitution group must have types that are either the same as the type of the head, or derived from it by either extension or restriction. They can be directly derived from it, or derived indirectly through multiple levels of restriction or extension. Only global element declarations can serve as heads of the substitution groups.

Substitution groups are a powerful tool and one may want to control their use using attributes xs:block and xs:final. The xs:final attribute can be used to prevent other people from defining Schemas that use your element declaration as the head of a substitution group. The xs:block attribute limits the use of substituted elements in instances. [WAL]

Substitution groups make content models more flexible and allow extensibility in directions the Schema author may not have anticipated. This flexibility is a two-edged sword: although it allows greater extensibility, it makes processing documents based on such Schemas more difficult. [OBA] Another complication is that members of a substitution group can be of a type derived from the substitution group's head when the type derivation can be both extension and restriction. The restriction of substitution groups is not recommended, since it may lead to interoperability issues between the Schema processors due to the fuzzy definition in the recommendations. [Eric van der Vlist (VLIS)]

<u>Rule 7-11</u>
The abstract element declarations and substitution group definitions MAY be used with caution. The use of block and final attributes SHOULD be used sparingly as and when needed.

A RosettaNet example for substitution group is as follows:

```
<xs:element name="TelephoneNumberType" type ="xs:string" abstract="true"/>
<xs:element name="WorkNumberType" type ="xs:string" substitutionGroup="TelephoneNumberType"/>
<xs:element name="FaxNumberType" type ="xs:string" substitutionGroup="TelephoneNumberType"/>
```

7.6 Use of Content

There are four types of content for complex types: simple, element-only, mixed and empty. [WAL]

<u>Rule 7-12</u>
Complex type with simple content SHOULD be used wherever needed.

```
   <xs:complexType name="SizeType">
      <xs:simpleContent>
         <xs:extension base="xs:integer">
            <xs:attribute name="system" type="xs:token"/>
         </xs:extension>
```

```
     </xs:simpleContent>
   </xs:complexType>
   <xs:element name="Size" type="SizeType"/>
```

```
   <Size system=”US-DRESS”>10</Size>
```

Rationale

Simple content allows character data only, with no children. Generally, the only thing that distinguishes a simple type from a complex type with simple content is that the latter may have attributes.

Rule 7-13

Complex type with element-only content SHOULD be used as needed.

```
   <xs:complexType name="ProductType">
     <xs:sequence>
        <xs:element name="Number" type="ProdNumType"/>
        <xs:element name="Name" type="xs:string"/>
        <xs:element ref="Size"/>
        <xs:element ref="Color"/>
     </xs:sequence>
    </xs:complexType>
    <xs:element name="Product" type="ProductType"/>
```

```
<Product>
   <Number>4566</Number>
   <Name>Long Skirt</Name>
   <Size system=”US-DRESS”>10 </Size>
   <Color value=”blue”/>
</Product>
```

Rationale

The element-only content allows children elements only, with no character data content.

Rule 7-14

Mixed content MUST NOT be used, as the character data in mixed content is completely unrestricted.

Rationale

Mixed content allows character data as well as child elements.

Rule 7-15

Complex type with empty content SHOULD be used as needed. Example of Empty content is <br/> element in XHTML. [XHTML]

Rationale

Empty content allows neither character data nor child elements. Elements with empty content may or may not have values in attributes.

7.6.1 Use of Default Values

Rule 7-16

The use of default values and fixed values is discouraged. The default values and fixed values SHOULD NOT be used. All the attribute and element values SHOULD be explicitly indicated.

Rationale

Default values of both attributes and elements are declared using the default attribute, although this attribute has a slightly different consequence in each case. Default attribute values apply when attributes are missing, and default element values apply when elements are empty. [XSDP]

The fixed attribute is used in both attribute and element declarations to ensure that the attributes and elements are set to particular values. This declaration means that the appearance of a fixed attribute in an instance document is optional, although if the attribute does appear, its value must be the same as in the corresponding declaration, and if the attribute does not appear, the Schema processor will provide a value from the corresponding declaration. Note that the concepts of a fixed value and a default value are mutually exclusive, and so it is an error for a declaration to contain both fixed and default attributes.

Rule 7-17

XML Schema built-in default values MUST be specified consistently.

Rationale

Having mixed approach when indicating XML Schema built-in default values, like sometimes indicating minOccurs=”1” and sometimes not, is often confusing for the human audience.

7.7 Use of Nillability

XML Schema provides a way of indicating nillability. By marking an element as “nil”, you are telling the processor “I know this element is empty, but I want it to be valid anyway.” The actual reason why this is empty and what the application should do, is entirely up to you. [WAL] It may indicate that the information is unknown, or not applicable, or the element may be absent for some other reason. Sometimes it is desirable to represent an unshipped item, unknown information, or inapplicable information *explicitly* with an element, rather than by an absent element. For example, it may be desirable to represent a "null" value being sent to or from a relational database with an element that is present. Such cases can be represented using XML Schema's nil mechanism, which enables an element to appear with or without a non-nil value. [XSDP]

XML Schema's nil mechanism involves an "out of band" nil signal. In other words, there is no actual nil value that appears as element content, instead there is an attribute to indicate that the element content is nil. [XSDP]

Example

```
<xs:element name="shipDate" type="xs:date" nillable="true"/>
```

And to explicitly represent that shipDate has a nil value in the instance document, we set the nil attribute (from the W3C XML Schema namespace for instances) to true. [XSDP]

Example

```
<shipDate xsi:nil="true"></shipDate>
```

Rule 7-18

Nillability SHOULD not be used.

Rationale

The tool support for nillability is poor so this should be used with caution. The functionality for nillability can be achieved to some extent by using optional elements.

7.8 Use of Any Element and Any Attribute

Rule 7-19

“any” wildcard (for both attributes and element) MUST NOT be used as it is a loose form of extension. If there is a need for additional elements or attributes not mentioned in the RosettaNet provided Schemas, request MUST be submitted to RosettaNet for addition in the Schema definitions.

7.9 Message Constraint Representation

7.9.1 Data Type Constraints

Rule 7-20

User-defined data types MUST be based on built-in atomic types, i.e. exclusively use built-in xs:date for dates or types that are derived from xs:date. If any further formatting constraint is needed which cannot be expressed in XSD then it MUST be expressed as Schematron constraints in the “Constraint” child element inside the xs:appinfo child element of the xs:annotation element of the Schema. The processing of these Schematron constraints SHOULD be deferred to the application level. The format of indicating an instant of time in Schemas MUST conform to a built-in datatype, xs:dateTime. The xs:dateTime UTC (Coordinated Universal Time) format MUST be followed for representing date and time in international trade. For local trade the use of UTC format is up to the trading partners.

Example XML Schema

```
   <xs:simpleType name="DateTimeStampType">
      <xs:restriction base="xs:dateTime">
      -----
      -----
      </xs:restriction>
   </xs:simpleType>
```

Note

Schematron rules provide formatting and path/relationship based integrity constraints, that are not available in XSD. The following example is taken from [MIT].

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.demo.org"
        xmlns="http://www.demo.org"
        xmlns:sch="http://www.ascc.net/xml/Schematron”
        elementFormDefault="qualified">
    <xs:annotation>
        <xs:appinfo>
           <Constraint>
           <sch:title>Schematron validation</sch:title>
           <sch:ns prefix="d" uri="http://www.demo.org"/>
           </Constraint>
        </xs:appinfo>
    </xs:annotation>

    <xs:element name="Demo">
        <xs:annotation>
            <xs:appinfo>
              <Constraint>
                <sch:pattern name="Check A greater than B">
```

```
                    <sch:rule context="d:Demo">
                        <sch:assert test="d:A &gt; d:B" diagnostics="lessThan">A should be

                    </sch:rule>
                </sch:pattern>
                <sch:diagnostics>
                    <sch:diagnostic id="lessThan">
                        Error! A is less than B. A = <sch:value-of select="d:A"/>  B =

                    </sch:diagnostic>
                </sch:diagnostics>
              </Constraint>
            </xs:appinfo>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="A" type="xs:integer" />
                <xs:element name="B" type="xs:integer" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

8 Codelists

8.1 Internal Codelist

8.1.1 Creation of Codelist

Rule 8-1
An xs:simpleType with enumerations MUST be defined to contain the content of the codelist. Its base type SHOULD be xs:token. Its name SHOULD consist of the codelist name and a suffix "ContentType" and SHOULD not be used to define any element directly.

Rule 8-2
An xs:complexType MUST be defined as extension of the content type with three fixed value attributes: identifier, agency and version, whose types SHOULD be xs:token. Its name SHOULD consist of the codelist name and a suffix "Type".

Rule 8-3
An abstract element MUST be declared with the content type. Its name SHOULD consist of the codelist name and a suffix "A". The type of this element MUST match the content type defined under the Rule 8-1.

Rule 8-4
A default element MUST be declared with the type. Its name MUST be the same as the codelist name. Its substitution group MUST be the abstract element.

Example XML schema

```
<xs:element name="TransportEventA" type="TransportEventContentType"


<xs:element name="TransportEvent" type="TransportEventType"


<xs:simpleType name="TransportEventContentType">
   <xs:restriction base="xs:token">
      <xs:enumeration value="DOC"/>
      <xs:enumeration value="PIC"/>
      <xs:enumeration value="SHP"/>
   </xs:restriction>
</xs:simpleType>

<xs:complexType name="TransportEventType">
   <xs:simpleContent>
      <xs:extension base="TransportEventContentType">
         <xs:attribute name="identifier" type="xs:token" fixed="TransportEvent"/>
         <xs:attribute name="agency" type="xs:token" fixed="RosettaNet"/>
         <xs:attribute name="version" type="xs:token" fixed="1.0"/>
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
```

8.1.2 Extension of Codelist

Rule 8-5

An xs:simpleType MUST be defined as a union whose xs:memberTypes are the original content type and an anonymously defined xs:simpleType with new enumerations whose base type SHOULD be xs:token. Its name SHOULD consist of the codelist name and a suffix "ContentType" and SHOULD not be used to declare any element directly.

Rule 8-6

An xs:complexType MUST be defined as extension from the content type with three fixed value attributes: identifier, agency and version, whose types SHOULD be xs:token. Its name SHOULD consist the codelist name and a suffix "Type".

Rule 8-7

A default element MUST be declared with the type. Its name MUST be the same as the codelist name. Its substitution group MUST be the original abstract element.

Example XML schema

```
<xs:element name="ExtTransportEvent" type="TransportEventType"


<xs:simpleType name="ExtTransportEventContentType">
   <xs:union memberTypes="TransportEventContentType">
      <xs:simpleType>
         <xs:restriction base="xs:token">
            <xs:enumeration value="EXT"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:union>
</xs:simpleType>

<xs:complexType name="ExtTransportEventType">
   <xs:simpleContent>
      <xs:extension base="ExtTransportEventContentType">
         <xs:attribute name="identifier" type="xs:token" fixed="ExtTransportEvent"/>
         <xs:attribute name="agency" type="xs:token" fixed="RosettaNet"/>
         <xs:attribute name="version" type="xs:token" fixed="1.0"/>
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
```

8.1.3 Restriction of Codelist

Rule 8-8

An xs:simpleType MUST be defined as restriction of the original content type. Its name SHOULD consist of the codelist name and a suffix "ContentType" and SHOULD not be used to declare any element directly. The set of enumeration values in a restricted codelist MUST be a proper subset of the set of enumeration values in the original codelist.

Rule 8-9

An xs:complexType MUST be defined as extension from the content type with three fixed value attributes: identifier, agency and version, whose types SHOULD be xs:token. Its name SHOULD consist of the codelist name and a suffix “Type”.

Rule 8-10

A default element MUST be declared with the type. Its name MUST be the codelist name. Its substitution group MUST be the original abstract element.

Example XML schema

```
<xs:element name="ForecastTransportEvent" type="ForecastTransportEventType"


<xs:simpleType name="ForecastTransportEventContentType">
   <xs:restriction base="TransportEventContentType">
      <xs:enumeration value="DOC"/>
      <xs:enumeration value="PIC"/>
   </xs:restriction>
</xs:simpleType>

<xs:complexType name="ForecastTransportEventType">
   <xs:simpleContent>
      <xs:extension base="ForecastTransportEventContentType">
         <xs:attribute name="identifier" type="xs:token" fixed="ForecastTransportEvent"/>
         <xs:attribute name="agency" type="xs:token" fixed="RosettaNet"/>
         <xs:attribute name="version" type="xs:token" fixed="1.0"/>
      </xs:extension>
   </xs:simpleContent>
</xs:complexType>
```

8.2 External Codelist

Rule 8-11

The targetNamespace SHOULD be used to denote the external source.

Rule 8-12

Creation procedure of an external codelist MUST be the same as of internal ones except that there is no need to declare enumerations in its content type since they are declared externally.

Rule 8-13

Extension procedure of an external codelist MUST be the same as of internal ones.

Rule 8-14

Restriction procedure of an external codelist MUST be the same as of internal ones.

Example XML Schema

```
<xs:schema targetNamespace="http://www.iso.ch/iso/en/prods-services/iso3166ma/02iso-3166-



   <xs:element name="CountryA" type="CountryContentType" abstract="true"></xs:element>

   <xs:element name="Country" type="CountryType" substitutionGroup="CountryA"></xs:element>

   <xs:simpleType name="CountryContentType">
      <xs:restriction base="xs:token"></xs:restriction>
   </xs:simpleType>
   <xs:complexType name="CountryType">
      <xs:simpleContent>
         <xs:extension base="CountryContentType">
            <xs:attribute name="identifier" type="xs:token" fixed="Country"/>
            <xs:attribute name="agency" type="xs:token" fixed="ISO"/>
            <xs:attribute name="version" type="xs:token" fixed="1.0"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
</xs:schema>
```

9 Schema File Naming Conventions and Packaging

9.1 Schema Packaging Conventions

Schemas will be packaged in the following way (for further explanation see PIP Development Guide) [PIPDEV]:

- XML
    - Domain
        - xxxDomain
            - Codelist
    - Interchange
    - System
    - Universal
        - Codelist

9.2 Schema File Naming Conventions

Definition: System Structure Schemas
The Schema definitions of System Structure types and elements are called System Structure Schemas – as these are reused in order to construct all other Schemas defined below.

Definition: Universal Structure Schemas
The Schema definitions of Universal Structure types and elements are called Universal Structure Schemas – as these are reused in order to construct more complex data structures in all PIPs.

Definition: Domain Structure Schemas
The Schema definitions of Domain types and elements are called Domain Structure Schemas – as these are reused in order to construct more complex data structures to create specific PIPs.

Definition: Interchange Structure Schemas
The Schema definitions of Interchange types and elements are called Interchange Structure Schemas – as these are used to construct PIP Messages to be exchanged between partners.

Rule 9-1
Schema file naming SHOULD be in UpperCamelCase, i.e. the leading character of each word is capitalized and file extension SHOULD be xsd.

Rule 9-2
For each codelist there MUST be one and only one Schema. Codelist Schema filename MUST include prefix that denotes the codelist provider.

Example

```
ISO_CountrySubdivision.xsd
```

9.3 System Structure Schemas

System structures contain the basic reusable building blocks to be used across all other Schemas. System structures include reusable elements, attributes and complex types.

Rule 9-3

System Structures Schemas MAY contain reusable definitions / declarations of several system structures.

Rule 9-4

Reusable system structures SHOULD be defined / declared in separate Schemas for better readability and differential namespace treatment.

Rule 9-5

Naming Convention for the files storing System Structure Schemas is:

	Subfield Name	Subfield Format	Subfield Value
Subfield 1	System Structure Name	Unabbreviated Alphanumeric string	
Subfield 2	File Extension	3 characters	xsd

Example

StandardDocumentHeader.xsd

9.4 Universal Structure Schemas

Universal structures contain the basic reusable building blocks to be used across all the PIPs. Universal structures include reusable elements, complex types, simple types and codelists.

Note

Codelist Schemas are fully explained in Codelists.

Rule 9-6

Universal Structure Schemas SHOULD contain reusable definitions / declarations of several universal structures.

Rule 9-7

Reusable universal structures SHOULD be defined / declared in separate Schemas for better readability and differential namespace treatment.

Rule 9-8

All reusable universal structures and data types MUST be grouped by categories. There SHOULD be one Schema for each category.

Rule 9-9

Universal Structure Schemas MUST NOT belong to the same namespace. The reusable elements and the types that are required for the definition of those elements and only for those elements MUST be in the same file and namespace.

Rule 9-10

File names of Universal Structure Schemas MUST include the category name.

Rule 9-11

Naming Convention for the files storing Universal Structure Schemas is:

	Subfield Name	Subfield Format	Subfield Value
Subfield 1	Category Name	Unabbreviated	

		Alphanumeric string	
Subfield 2	File Extension	3 characters	xsd

Example

ContactInformation.xsd

9.5 Domain Structure Schemas

Domain Structure Schemas contain the basic reusable building blocks specific to a particular domain. Domain Structure Schemas include reusable elements, complex types and codelists.

Note

Codelist Schemas are fully explained in section 8.

Rule 9-12

Domain Structure Schemas SHOULD contain reusable definitions / declarations of several domain structures.

Rule 9-13

Reusable domain structures SHOULD be defined / declared in separate Schemas for better readability and differential namespace treatment.

Rule 9-14

All reusable domain structures and data types MUST be grouped by domains. There SHOULD be one Schema for each domain.

Rule 9-15

File names of Domain Structure Schemas MUST include the domain name.

Rule 9-16

Naming Convention for the files storing Domain Structure Schemas is:

	Subfield Name	**Subfield Format**	**Subfield Value**
Subfield 1	Domain Name	Unabbreviated Alphanumeric String	
Subfield 2	File Extension	3 characters	xsd

Example

CollaborativeForecasting.xsd

9.6 Interchange Structure Schemas

Rule 9-17

There MUST be only one Schema per PIP Action Message.

Rule 9-18

The Interchange Structure Schemas SHOULD declare only one named global element.

Rule 9-19

File naming convention for Interchange Structure Schemas SHOULD follow the PIP naming convention explained in PIP Development Guide. [PIPDEV]

Naming Convention for the files storing Interchange Structure Schemas is:

	Subfield Name	Subfield Format	Subfield Value
Subfield 1	Interchange Structure term	3 characters	PIP
Subfield 2	Interchange Structure code (Segment, Cluster, Number)	3 characters	
Subfield 3	Business Document Name (Action Message name)	Full name – as many characters	
Subfield 4	File Extension	3 characters	xsd

Example

```
PIP4A3ThresholdReleaseForecastNotification.xsd
```

10 XML instance documents (PIP Action Messages)

Note
This section is intended for an audience that is different then the audience for the previous sections. Also, it addresses only a subset (incomplete list) of all aspects related to composition of XML instance documents that conform to Schema constrains explained in the rest of this document. Because of these two facts, it is possible that this section will be considerably larger as this documents is being revised or it might be promoted into different document(s).

10.1 XML & XSD

Both XSD and XML instance documents use the same syntax – therefore XML and XSD coding conventions and document structure will be largely the same for both XML and XSD documents. XSDs capture the syntax and semantics for a particular class of XML documents in W3C XML Schema language and provide the means for XML Schema processors to validate the corresponding XML instance documents.

10.2 Naming conventions for XML Documents

Documentation, Naming conventions, and component ordering of XML instance documents are the same as that of Schemas.

10.3 Referencing Schemas from PIP Messages

Rule 10-1
PIP XML Action Message documents MUST NOT have the absolute path defined in xsi:schemaLocation attribute. The xsi:schemaLocation attribute MAY contain the relative paths with respect to the location where the current Schema is stored.

Rationale
The xsi:schemaLocation attribute provides a hint to the processor as to where to find a Schema that declares components for that namespace. The path of the root should be specified in the packaging. The reason behind this decision is security concerns as well as ease of processing. Though desirable that xsi:schemaLocation contains relative path, the tool support is not sufficiently good at this time.

Rule 10-2
PIP XML Action Message documents SHOULD set the value of the “schemaVersion” attribute. The “schemaVersion” attribute MAY contain more then one value of the Schema versions that the PIP XML Action Message instance is compatible with.

Rule 10-3
PIP XML Action Message documents MUST set the value of the “pipVersion” element inside the “Service Header” to match the “PIP Umbrella Version”.

Rule 10-4
PIP XML Action Message documents MUST set the value of the “TypeVersion” element inside the “Standard Document Header” to match the “PIP Umbrella Version”.

Note

“PIP Umbrella Version” denotes the PIP version (e.g. R11.01) of the whole PIP Package. For further explanation of “PIP Umbrella Version” see [PIPDEV]. For explanation of “pipVersion” see [RNIF]. For explanation of “Standard Document Header” see [SBDH].

Rationale

This approach allows gradual transitioning to new Schemas. It can also support future needs of correlating a given PIP XML Action Message fragment to the type definitions in a particular namespace. In some cases the PIP XML Action Message fragment might become extracted from the source PIP XML Action Message document in which it was originally sent so “schemaVersion” could be used by destination processing application in order to take appropriate action(s).

Example XML Instance

```
<Thing xmlns="urn:rosettanet:specification:domain:ThingType:xsd:schema:0.3"
```

11 References

Source	Description
[MIT]	Author: xml-dev list group Title*: "XML Schemas: Best Practices."* The MITRE Corporation Retrieved October 20, 2003 from: http://www.xfront.com/BestPracticesHomepage.html
[NAM]	Editors: Tim Bray, Dave Hollander, Andrew Layman Title: "*Namespaces in XML"* World Wide Web Consortium Retrieved October 20, 2003 from: http://www.w3.org/TR/1999/REC-xml-names-19990114
[NSSM]	RosettaNet Namespace Specification and Management October 2003.
[OBA]	Author: Dare Obasanjo Title: "*W3C XML Schema Design Patterns: Avoiding Complexity"* O'REILLY xml.com Retrieved October 20, 2003 from: http://www.xml.com/pub/a/2002/11/20/schemas.html
[PIPDEV]	PIP Development Guide
[RFC2119]	Author: Scott Bradner Title: *"Key words for use in RFCs to Indicate Requirement Levels"* The Internet Engineering Task Force Retrieved October 20, 2003 from: http://www.ietf.org/rfc/rfc2119.txt
[RNIF]	Title: *"RosettaNet Implementation Framework"* RosettaNet Consortium Retrieved October 20, 2003 from: http://www.rosettanet.org/rnif
[SBDH]	Title: *"UN/CEFACT Standard Business Document Header"* Revision 2.1 UN/CEFACT Retrieved October 20, 2003 from: http://webster.disa.org/cefact-groups/atg/downloads/Generic_Header_TS_rev2.1.zip
[STRON]	Author: Rick Jelliffe Title: *"The Schematron - An XML Structure Validation Language using Patterns in Trees"* Retrieved October 20, 2003 from: http://www.ascc.net/xml/resource/schematron/schematron.html
[UBLS]	Author: Eve Maler Title: *"Schema Design Rules for UBL and May be for You"* Retrieved October 20, 2003 from: http://www.idealliance.org/papers/xml02/dx_xml02/papers/05-01-02/05-01-02.html
[VLIS]	Author: Eric van der Vlist (2002) Title: *"XML Schema"* O'Reilly Publications
[WAL]	Author: Priscilla Walmsley (2002) Title: *"Definitive XML Schema"* The Charles Goldfarb definitive XML Series.
[XHTML]	Authors : Members of the W3C HTML Working Group Title : *"XHTML™ 1.0 The Extensible HyperText Markup Language"*

	World Wide Web Consortium Retrieved October 20, 2003 from: http://www.w3.org/TR/xhtml1/
[XML]	Editors : Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eva Maler Title : *"Extensible Markup Language (XML) 1.0"* W3C Recommendation 6th October 2000. World Wide Web Consortium Retrieved October 20, 2003 from: http://www.w3.org/TR/2000/REC-xml-20001006
[XSD]	Editors : Ashok Malhotra, Murray Maloney Title : *"XML Schema Requirements"* World Wide Web Consortium Retrieved October 20, 2003 from: http://www.w3.org/TR/NOTE-xml-schema-req
[XSDD]	Editors : Paul V. Biron, Ashok Malhotra Title : *"XML Schema Part 2: Datatypes"* World Wide Web Consortium Retrieved October 20, 2003 from: http://www.w3.org/TR/xmlschema-2/
[XSDP]	Editor : David C. Fallside Title : *"XML Schema Part 0: Primer"* World Wide Web Consortium Retrieved October 20, 2003 from: http://www.w3.org/TR/xmlschema-0/
[XSDS]	Editors : Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn Title : *"XML Schema Part 1: Structures"* World Wide Web Consortium Retrieved October 20, 2003 from: http://www.w3.org/TR/xmlschema-1/

1788 # 12 Glossary

Term	Definition
Abstract types	Allow use of complex types in such a way that a single element name can be used to represent various types in an XML document instance.
Annotation	Information for human and/or mechanical consumers. The interpretation of such information is not defined in the XML Schema specifications. The annotation element can contain one or more <documentation> or <appinfo> elements.
Attribute	A name=”value” field within an XML element, providing information associated with that XML element.
Attribute Group	A set of attribute declarations, enabling re-use of the same set in several complex type definitions.
Attribute Group Definition	An attribute group definition is an association between a name and a set of attribute declarations, enabling re-use of the same set in several complex type definitions.
Built-in Datatypes	Datatypes that are defined either in the XML Schema specification (as primitive types) or in this specification, and can be either primitive or derived.
Character set	The encoding method for the data values of the document, based on Unicode format.
Complex Type	An XML element type that allows nested elements in their content and may carry attributes.
Complex Type Definition	A complex type definition is a set of attribute declarations and a content type, applicable to the attributes and children of an element information item respectively. The content type may require the children to contain neither element nor character information items (that is, to be empty), to be a string that belongs to a particular simple type or to contain a sequence of element information items that conforms to a particular model group, with or without character information items as well.
Complex type extension	Extension adds attributes, and adds elements to the end of the content model of the base type.
Complex type restriction	Restriction limits a base type to a more restrictive set of valid values.
component	Component means a basic building block of the Schema like named type, named element, named group etc.
Datatype	A datatype is a 3-tuple, consisting of a) a set of distinct values, called its value space, b) a set of lexical representations, called its lexical space, and c) a set of facets that characterize properties of the value space, individual values or lexical items.
Default attribute values	Data values that imply a default value if they do not explicitly appear in the XML instance document.
Derived Data Types	Derived datatypes are those that are defined in terms of other datatypes. A datatype is said to be derived by **restriction** from another datatype when values for zero or more constraining facets are specified that serve to constrain its value space and/or its lexical space to a subset of those of its base type. Every datatype that is derived by **restriction** is defined in terms of an existing datatype, referred to as its **base type**. **Base types** can be either primitive or derived.
Element	A fundamental unit of XML information, which has an element name,

	optional attributes, optional data value, and an associated type definition. Elements may be nested, one inside another.
Element Declaration	An element declaration is an association of a name with a type definition, either simple or complex, an (optional) default value and a (possibly empty) set of identity-constraint definitions.
Facet	A **facet** is a single defining aspect of a value space. Generally speaking, each facet characterizes a value space along independent axes or dimensions.
Fixed attribute values	An attribute value that always has the same value.
Globally defined attributes	Attribute definitions that are defined at the highest level in the XML Schema document, so that the definitions can be reused.
Globally defined elements	Element definitions that are defined at the highest level in the XML Schema document, so that the definitions can be reused.
Groups	XML Schema allows fragments of content models to be named and referenced from multiple complex types.
Main type	A reusable type that is used to define the root element of the XML instance document (PIP Action Message). In case when Schema contains only one reusable type definition than that type is by default the Schema main type.
Message Guidelines	The Message guidelines are the semantic documentation of the PIPs, which cannot be captured in Schemas.
Mixed Content	A combination of child elements and character data nested within an element.
Name	Represents names in XML. A **Name** is a token that begins with a letter, underscore, or colon and continues with name characters (letters, digits, and other characters). This data type is derived from **token**.
NCName	Represents noncolonized names. This data type is the same as **Name**, except it cannot begin with a colon. This data type is derived from **Name**.
Named Types	Named types may be defined once and used many times.
Namespaces	An XML namespace is a collection of names identified by a URI reference, which are used in XML documents as element types and attribute names.
normalizedString	Represents white space normalized strings. This data type is derived from **string.**
Simple Type	Simple types cannot have element content and cannot carry attributes.
Simple Type Definition	A simple type definition is a set of constraints on strings and information about the values they encode, applicable to the normalized value of an attribute information item or of an element information item with no element children. Informally, it applies to the values of attributes and the text-only content of elements.
Substitution groups	An element can be declared to be a substitute for another element, the "head" element, allowing the new element to appear anywhere the head element may appear.
targetNamespace	The namespace of an instance document.
token	Represents tokenized strings. This data type is derived from **normalizedString**.
Type Derivation	XML Schema allows a type to be derived from another type (its base type), either by extension or restriction.
Type Redefinition	XML Schema allows a Schema author to redefine the types or groups of another Schema document.

Type Substitution	Allows a base type to be substituted by any derived type.
PIP Umbrella Version	The PIP version (e.g. R11.01) of the whole PIP Package.
Union types	The union operation is supported by XML Schema for element types. For example, a codelist may be defined as the union of two other codelists.
User-derived Datatypes	User-derived datatypes are those derived datatypes that are defined by individual Schema designers.
Value Space	A value space is the set of values for a given datatype. Each value in the value space of a datatype is denoted by one or more literals in its lexical space.
XML Schema	An XML document that defines the allowable content of a class of XML documents. A class of documents refers to all possible permutations of structure in documents that will still confirm to the rules of the Schema.

13 Appendix

13.1 Rules Appendix

This section summarizes the rules for a quick review. For complete information regarding particular aspect refer to the appropriate section.

Prologue and Encoding	Rule 3-1 MUST specify XML prologue at the beginning of each schema Rule 3-2 Either "UTF-8" or "UTF-16" MUST be used as the value for character set and encoding type
xs:schema element	Rule 3-3 "xs" or "xsd" namespace prefix MAY be used to indicate the usage of W3C XML Schema namespace Rule 3-4 The attribute xs:targetNamespace of xs:schema MUST be specified for all Schema documents Rule 3-5 "tns" namespace prefix SHOULD be used to indicate xs:targetNamespace when targetNamespace is not the same as the default namespace of the Schema Rule 3-6 Default namespace MAY be specified as an attribute of xs:schema element Rule 3-7 The xs:elementFormDefault attribute of xs:schema MUST have the value "qualified" and the xs:attributeFormDefault attribute MAY have the value of either "qualified" or "unqualified". Rule 3-8 The xs:version attribute of xs:schema MUST be present Rule 3-9 Order of xs:schema attributes MUST be as follows: targetNamespace declaration, declaration binding "xs" namespace prefix, default namespace declaration, declaration binding "tns" prefix, any other declarations binding prefixes to other namespaces, elementFormDefault declaration, attributeFormDefault declaration and version declaration.
Documentation	Rule 3-10 The xs:schema root element and all reusable components in the Schema MUST have xs:annotation defined. Rule 3-11 All Schema annotations MUST be in English and within the xs:annotation element Rule 3-12 The documentation for a Schema component SHOULD be placed as close to the component as possible Rule 3-13 Any constraints relevant to either the whole Schema or to an individual Schema component MUST be expressed in Schematron syntax
Schema Documentation	Rule 3-14 Any human readable information relevant to the whole Schema MUST be contained in an xs:documentation element Rule 3-15 Any application related information relevant to the whole Schema MUST be contained in an xs:appinfo element
Component Documentation	Rule 3-16 Any human readable information relevant to reusable types MUST be contained in an xs:documentation element Rule 3-17 Any application related information relevant to reusable types MUST be contained in an xs:appinfo element Rule 3-18 Only when the name of a reusable element is different than its default name (i.e. type name without the suffix) the reusable element SHOULD have its own documentation Rule 3-19 nent documentation for any lower level element SHOULD be defined only by "definition"
CodeList Documentation	Rule 3-20 Any human readable information relevant to codelists MUST be contained in an xs:documentation element

	Rule3-21 Any application related information relevant to codelists MUST be contained in an xs:appinfo element
Component Ordering	Rule 3-22 Schemas MUST follow consistent structuring rules Rule 3-23 Placement of various Schema components (follow link for details) Rule 3-24 Within the type definition, the sequences, choice, groups and sub-content models SHOULD be ordered in alphabetical order. Also within each content model (like sequence, choice, groups etc) elements SHOULD be sorted in alphabetical order
Reusing Schemas	Rule 4-1 The xs:import attribute MUST contain the schemaLocation attribute that points to the imported schema(s) via relative paths Rule 4-2 Import SHOULD be used where needed. Circular imports MUST be avoided. Duplicate imports SHOULD be avoided. Rule 4-3 xs:include is allowed and MAY be used where needed. Rule 4-4 xs:redefine MUST NOT be used
Internationalization Features	Rule 5-1 The name of an XML Schema component MUST be an NCName (XML Name minus the ":") Rule 5-2 All names MUST be composed of alphanumeric characters only Rule 5-3 The name of an XML Schema component MUST be taken out of the UML model Rule 5-4 All Schema names and values created and maintained by RosettaNet SHOULD be understandable by an English speaking audience
Acronyms	Rule 5-5 Acronyms SHOULD be written using uppercase
Element	Rule 5-6 For element names, the Upper Camel Case ("UCC") convention MUST be used Rule 5-7 While creating names for inner elements, concatenating the name of the inner element to the name of the outer element SHOULD be avoided
Attribute	Rule 5-8 For attribute names, the Lower Camel Case ("LCC") convention MUST be used
Type	Rule 5-9 All reusable, extendable, and restrictable types MUST be named. All such type names MUST be global in scope Rule 5-10 For type names, the Upper Camel Case ("UCC") convention MUST be used
Model Group	Rule 5-11 For model group names, the Upper Camel Case ("UCC") convention MUST be used
Namespace Convention	Rule 5-12 Name space prefix MAY be created by the first letters of the targetNamespace that appear between "specification" and "xml". Rule 5-13 Relative URI references MUST NOT be used in namespace declarations Rule5-14 All reusable Schema components are considered RosettaNet Resources and MUST have a URN assigned to them Rule5-15 Schema filename and targetNamespace URN MUST "canonically" match where for each targetNamespace there is one and only one file Rule5-16 Schema targetNamespace URN SHOULD "canonically" match URN of one and only one of the Schema reusable types Rule5-17 Schema targetNamespace URN MAY "canonically" match URN of entities that convey logical grouping of resources Rule5-18 W3C XML Schema namespace MAY be the default namespace for any Schema Rule5-19 xs:targetNamespace MAY be the default namespace for all Interchange Structure Schemas
Namespace exposure	Rule 5-20 Namespaces of elements MUST be exposed in the XML instance files by setting elementFormDefault to "qualified" in the xs:schema
Form Attribute	Rule 5-21 RosettaNet Schema developers MUST NOT use the form

	attribute
Versioning Philosophy	Rule 6-1 Schemas, namespaces and reusable types MUST have version numbers assigned to them
Versioning Schemas	Rule 6-2 The Schema version MUST match the version of the "main type" if the "main type" exists inside the Schema Rule 6-3 The targetNamespace of a Schema MUST include the same number that matches the major number of the value of the built-in xs:schema "version" attribute Rule 6-4 Major Schema version number MUST be changed when existing instance documents that validate against the current Schema cannot validate against the new Schema
Versioning reusable types	Rule 6-5 "TypeVersion" element MUST be included under xs:appinfo element that annotates the reusable type Rule 6-6 "schemaVersion" attribute of the "token" type MUST be declared as an optional attribute for all reusable types
Use of XSD Built-In Types	Rule 7-1 The built-in types outlined in W3C XML Schema Datatypes [XSDD] SHOULD be used in designing Schemas as much as possible
Use of Content Model: sequence, choice, all.	Rule 7-2 While composing groups of elements xs:sequence SHOULD be the preferred compositor, the use of xs:all is NOT RECOMMENDED Rule 7-3 Schemas MUST define named global types (simpleType or complexType) Rule7-4 More then one global type definition and more then one global element declaration MAY be present in a Schema
Use of Named Model Groups	Rule7-5 The xs:group MAY be used when there is a need to reuse a set of elements when application design requires presentation to be structured
Use of Named Model Groups	Rule7-6 Extensibility SHOULD be implemented using XML Schema extension and restriction
Extensibility	Rule7-6 Extensibility SHOULD be implemented using XML Schema extension and restriction Rule7-7 Extensibility of RosettaNet Schemas by Trading Partners that use Schemas is allowed only for Codelists
Inheritance via Extension	Rule 7-8 Complex type extension SHOULD be used
Inheritance via Restriction	Rule 7-9 Simple type restriction SHOULD be used. Use of complex type restriction is discouraged, as it is complex
Use of abstract type and substitution groups	Rule 7-10 The abstract complex type definitions MAY be used in RosettaNet Schemas as needed Rule 7-11 The abstract element declarations and substitution group definitions MAY be used with caution
Use of Content	Rule 7-12 Complex type with simple content SHOULD be used wherever needed Rule 7-13 Complex type with element-only content SHOULD be used as needed Rule 7-14 Mixed content MUST NOT be used, as the character data in mixed content is completely unrestricted Rule 7-15 Complex type with empty content SHOULD be used as needed. Example of Empty content is <br/> element in XHTML
Use of Default Values	Rule 7-16 The use of default values and fixed values is discouraged Rule 7-17 XML Schema built-in default values MUST be specified consistently
Use of Nillability	Rule 7-18 Nillability SHOULD not be used
Use of Any Element and Any Attribute	Rule 7-19 "any" wildcard (for both attributes and element) MUST NOT be used as it is a loose form of extension
Data Type	Rule 7-20 User-defined data types MUST be based on built-in atomic types

Constraints	
Creation of Codelist	Rule 8-1 A simpleType (the content type) with enumerations MUST be defined to contain the content of the code list Rule 8-2 A complexType (the type) MUST be defined as extension of the content type with three attributes: identifier, agency and version, whose types SHOULD be xs:token and with fixed values Rule 8-3 An abstract element MUST be declared with the content type. Its name SHOULD consist of the codelist name and a suffix "A". Rule 8-4 A default element MUST be declared with the type
Extension of Codelist	Rule 8-5 A simpleType (the content type) MUST be defined as a union whose memberTypes is the original content type and an anonymously defined xs:simpleType with new enumerations whose base type SHOULD be xs:token Rule 8-6 An xs:complexType MUST be defined as extension from the content type with three fixed value attributes: identifier, agency and version, whose types SHOULD be xs:token Rule 8-7 A default element MUST be declared with the type. Its name MUST be the same as the codelist name
Restriction of Codelist	Rule 8-8 A simpleType (the content type) MUST be defined as restriction of the original content type Rule 8-9 An xs:complexType MUST be defined as extension from the content type with three fixed value attributes: identifier, agency and version, whose types SHOULD be xs:token Rule 8-10 A default element MUST be declared with the type. Its name MUST be the codelist name
External Codelist	Rule 8-11 The targetNamespace SHOULD be used to denote the external source Rule 8-12 Creation procedure of an external codelist MUST be the same as of internal ones except that there is no need to declare enumerations in its content type since they are declared externally Rule 8-13 Extension of an external code list MUST be the same as internal ones Rule 8-14 Restriction of an external code list MUST be the same as internal ones
Schema File Naming Conventions	Rule 9-1 Schema file naming SHOULD be in UpperCamelCase Rule 9-2 For each codelist there MUST be one and only one Schema. Codelist Schema filename MUST include prefix that denotes the codelist provider
System Structures Schemas	Rule 9-3 System Structures Schemas MAY contain reusable definitions / declarations of several system structures Rule 9-4 Reusable system structures SHOULD be defined / declared in separate Schemas for better readability and differential namespace treatment Rule 9-5 Naming Convention for the files storing System Structure Schemas (Follow link for details)
Universal Structures Schemas	Rule 9-6 Universal Structures Schemas SHOULD contain reusable definitions / declarations of several universal structures Rule 9-7 Reusable universal structures SHOULD be defined / declared in separate Schemas for better readability and differential namespace treatment Rule 9-8 All reusable universal structures and data types MUST be grouped by categories. There SHOULD be one Schema for each category Rule 9-9 Universal structures Schemas MUST NOT belong to the same namespace

	Rule 9-10 File names of Universal Structures Schemas MUST include the category name Rule 9-11 Naming Convention for the files storing Universal Structure Schemas (follow link for details)
Domain Structure Schemas	Rule 9-12 Domain structures Schemas SHOULD contain reusable definitions / declarations of several domain structures Rule 9-13 Reusable domain structures SHOULD be defined / declared in separate Schemas for better readability and differential namespace treatment Rule 9-14 All reusable domain structures and data types MUST be grouped by domains. There SHOULD be one Schema for each domain Rule 9-15 File names of Domain Structure Schemas MUST include the domain name Rule 9-16 Naming Convention for the files storing Domain Structure Schemas (follow link for details)
Interchange Structure Schemas	Rule 9-17 There MUST be only one Schema per PIP Action Message Rule 9-18 The Interchange Structure Schemas SHOULD declare only one named global element Rule 9-19 File naming convention for Interchange Structure Schemas SHOULD follow the PIP naming convention explained in PIP Development Guide
Referencing Schemas from PIP Messages	Rule 10-1 PIP XML Action Message documents MUST NOT have the absolute path defined in xsi:schemaLocation attribute Rule 10-2 PIP XML Action Message documents SHOULD set the value of the "schemaVersion" attribute Rule 10-3 PIP XML Action Message documents MUST set the value of the "pipVersion" element inside the "Service Header" to match the "PIP Umbrella Version" Rule 10-4 PIP XML Action Message documents MUST set the value of the "TypeVersion" element inside the "Standard Document Header" to match the "PIP Umbrella Version"