StanForD 2010 – Naming and design rules

Tapio Räsänen
Juha-Antti Sorsa

Metsäteho Oy

VERSION HISTORY

Date	Version	Description	Authors
24.11.2008	0.5	First released draft, missing lots of text and examples	Juha-Antti Sorsa, Metsäteho
31.12.2008	1.0	First released finalized version	Juha-Antti Sorsa, Metsäteho
28.1.2009	1.1	Few updates and misspelling corrections	Juha-Antti Sorsa, Metsäteho
5.1.2010	1.2	Added rules 7.6 and 7.7 and correct the corresponding topics in chapter 9 to follow those rules. Updated the rule 12.4 to allow empty elements in special cases. Added abbreviations MIN and MAX	Juha-Antti Sorsa, Metsäteho
23.6.2010	1.3	Added abbreviation MTH	Juha-Antti Sorsa, Metsäteho
13.2.2011	1.4	Updated the rule 7.4 to allow codelist namespace Updated the rule 10.1 to allow extension elements in many places in messages.	Juha-Antti Sorsa, Metsäteho
11.10.2011	1.5	Updated rule 12.3 to exclude order of attributes.	John Arlinger, Skogforsk
20.8.2012	1.6	Updated Rule 7.1 to exclude version information from namespace name. Updated also all texts and examples to follow the new rule. This change is obeyed from standard version 2.0.	Juha-Antti Sorsa, Metsäteho
4.3.2013	1.7	Corrected errors in examples relate to namespace naming.	Juha-Antti Sorsa, Metsäteho

Table of Contents

1 INTRODUCTION

1.1 Background

The data structure and format of StanForD has not been changed since the standard was established in 1986-1987. Discussions about updating the standard towards better structures and a new, more flexible format were started in the early years of 2000. First versions of StanForD XML-files were made then. Finally, the decision to upgrade the standard was made 2006-08-25 when StanForD 2010 –project was started.

One of the reasons to update the standard is that its technical data format has been considered too limited and complex for the present data management needs. The problem of the format is that it is unique, requires a lot of work to understand and special programming for implementing. For example, it does not support use of web technologies or communication with databases. Therefore it was decided, together with users and machine manufacturers, that one of the objectives of developing a new standard is to achieve a common and general format with open interface, making it easier to implement the standard in new wood supply applications.

XML has been chosen for the presentation technology of StanForD 2010. It was seen as the best solution because of its leading position as presentation technology of structured data and documents, especially in web-based applications. XML is general, open and free to use as well as well-supported by software developers and suppliers. Different XML technologies are being developed strongly all over the world and there are agreements on standardization of certain parts of XML. XML is utilized today in various data management systems of the forest companies and forest machine manufacturers. XML is used also in papiNet standard which is being extended for wood supply purposes, especially for data and message exchange between business partners of logistic chains. The goal is that StanForD and papiNet will remain independent standards but that they could support each other.

One of the main tasks of StanForD 2010 –project was to define the requirements of the data format in the new standard and to study how different solutions and ways to use XML suit those. This report presents the main issues of the study and gives the proposals concerning the format and the use of XML.

1.2 Audience

The main audience for this StanForD 2010 – Naming and Design Rules is the members of StanForD standardization group, especially those who are responsible for creating and maintaining StanForD 2010 schema definitions. Additionally all the representative of the organizations that are using StanForD messages might find this document useful for their system and software development tasks.

1.3 Terminology and notion

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this specification, are to be interpreted as described in Internet Engineering Task Force (IETF) Request For Comments (RFC) 2119.1. Non-capitalized forms of these words are used in the regular English sense.

Example – A representation of a definition or a rule. Examples are informative.

[Rule c.n] – Identification of a rule that requires conformance. Rules are normative. All the characters in rule text are in italics font. Rules are numbered as following:

c = chapter number
n = an increasing sequence of integers inside chapter starting from 1

When defining rules, the following annotations are used:

[ ] = optional
< > = variable
| = choice

Additionally the text backgrounds of the rules have light grey color.

`Courier font` – All XML code. Additionally different colors for different lexical elements increase readability.

1.4 Main resources

There are lots of resources available on the web that gives us best practices and guidelines for using XML Schema Language. The following resources were the most influential while writing this document.

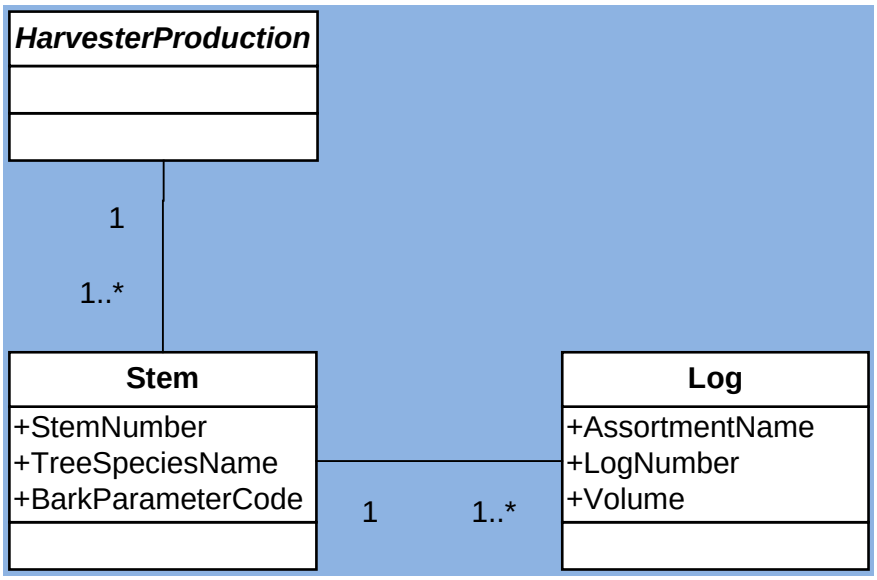
First of all there are two working drafts that gather most used practices and guidelines in schema development area. The first one is "Universal Business Language (UBL) Naming and Design Rules 2.0" from OASIS Universal Business Language Technical Committee (http://docs.oasis-open.org/ubl/prd-UBL-NDR-2.0.pdf) and the second more recent one "XML Naming and Design Rules V3.0" from UN/CEFACT (United Nation / Centre for Trade Facilitation and Electronic Business) (http://unstandards.org:8080/download/attachments/20381763/Specification...XML+Naming+and+Design+Rules+V3.0+1st+Public+Review.pdf)

Additionally resources are the following

http://www.xfront.com	Roger L. Costello's web pages have good collection of XML Schema Best Practices that were developed collaboratively by members of the xml-dev list group.
http://www.xml.com	The publisher O'Reilly supports and maintains this informative site.
http://www.w3.org/	The home page of the World Wide Web Consortium (W3C) is main resource for XML specifications.

1.5 Example used in this document

In this document all XML code examples are based on instance document scenario that is represented in the following UML class diagram.



Instance documents are very simplified examples of harvester production data that is composed of stems and each stem has from one to many logs. Each stem has also three properties that are a stem number, a tree species name and a country specific bark parameter code. Each log has three properties that are an assortment name, a log number and a volume. One example of XML instance document that obeys the UML graph design can be seen in the following code.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<HarvesterProduction>
   <Stem stemNumber="1" treeSpeciesName="Pine" barkParameterCode="1">
      <Log logNumber="1" assortmentName="Saw log">
         <Volume>400.0</Volume>
      </Log>
      <Log logNumber="2" assortmentName="Pulp">
         <Volume>150.0</Volume>
      </Log>
   </Stem>
</HarvesterProduction>
```

The XML specific features of the instance document code are described in the chapters of this document. One must remember that this harvester production example is provided for this document only. Its data content varies in many ways in proportion to the actual harvester production message of the StanForD 2010 standard.

2 XML SCHEMAS

2.1 Overall schema rules

XML is intended to be a self-describing data format, allowing authors to define a set of element and attribute names that describe the content of a document. In addition to that, we need to be able to define what element and attribute names are allowed to appear in a conforming document in order to make that document useful. Furthermore, we need to be able to indicate what kind of content each element and attribute is allowed to contain. Only then can we have an agreement what is the meaning of the markup, be it for a human or for an application.

A schema defines the allowable contents of a class of XML documents. A class of documents refers to all possible permutations of structure in documents that will still conform to the rules of the schema.

There are numerous XML schema languages available today, all with their strengths and weaknesses. Because the W3C XML Schema Language is the schema definition language with the broadest adoption and the best tool and software support, it has been chosen the schema definition language for the StanForD 2010 standardization task.

[Rule 2.1]	*All StanForD 2010 schema design rules MUST be based on the W3C XML Schema Recommendations: XML Schema Part 1: Structures Second Edition and XML Schema 1.1 Part 2: Datatypes.*

The W3C is the most important source of XML specifications. Those specifications have various statuses during their development. It's important that only finally approved versions are used.

[Rule 2.2]	*All StanForD 2010 conformant XML instance documents MUST be based on the W3C specifications that have recommendation status.*

The contents of the schemas can be structured lexically different ways. However, one predefined consistent way to do it is essential.

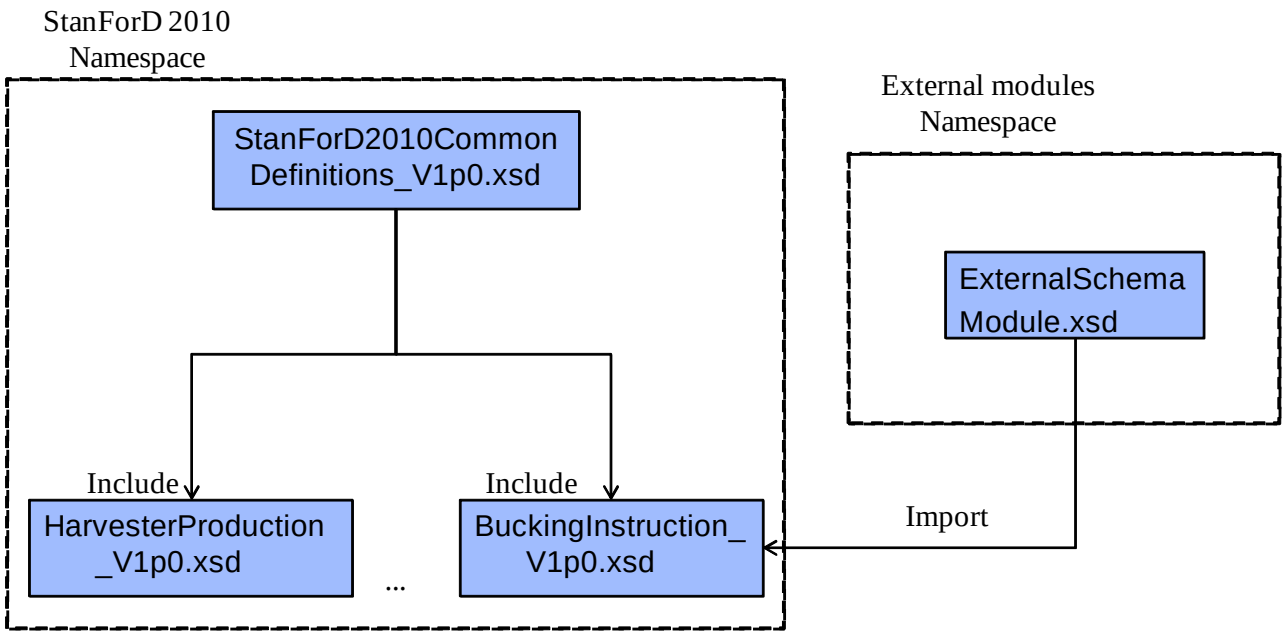
[Rule 2.3]	*All StanForD 2010 schemas MUST follow the standard structure defined in chapter 9.*

2.2 Schema modules

Modularity in schema design promotes reuse and provides significant management capabilities. Modules can be either unique in their functionality, or represent splitting of larger schema files for performance or manageability enhancement. A modularity model provides an efficient and effective mechanism for importing and including components as needed rather than dealing with complex, multi-focused schema.

StanFord 2010 schema module design is quite straightforward that can be seen in picture 2.1. Every StanFord 2010 message has its own schema module. Every schema module includes one common schema module "StanForDCommonDefinition_version.xsd" which contains all common global type definitions that are used in several message schemas. Additionally it's possible that StanForD

2010 schema modules could import additional schema modules from external sources. All StanForD 2010 schema modules belong to one common StanForD 2010 specific namespace (more detailed presentation will be in chapter 7).



Picture 2.1

StanForD 2010 message schema files are named formally according the following rule.

[Rule 2.4]	*The file name for StanForD 2010 messages schema modules MUST follow the form <MessageSchemaModuleName>_<Version>.xsd.*

In above rule <MessageSchemaModuleName> is predefined StanForD 2010 message name and <Version> is version number of the schema. Version conventions are presented in chapter 8.

3 NAMING CONSTRAINTS

Because StanForD is multinational standard, English has been chosen its official language. XML and XML development work could be done in principle using national languages. However XML tools and software are not necessarily supported by other languages than English. Hence XML development language should also be English.

[Rule 3.1] *Element, attribute and type names MUST be composed of words in the English language.*

The following set of naming rules is commonly used best practices. There are no reasons not to follow them in this standardization task also.

[Rule 3.2] *UpperCamelCase (UCC) MUST be used for naming elements and types.*

Example

```
<AssortmentName>Pulp</AssortmentName>
```

[Rule 3.3] *LowerCamelCase (LCC) MUST be used for naming attributes.*

Example

```
<Log logNumber="2" assortmentName="Pulp" volume="150.0"/>
```

[Rule 3.4] *Element, attribute and type names MUST be in singular form unless the concept itself is plural.*

[Rule 3.5] *Element, attribute and type names MUST be drawn from the following character set: a-z, A-Z and 0-9.*

[Rule 3.6] *XML element, attribute and type names MUST NOT use acronyms, abbreviations, or other word truncations, except those included in the appendix A. List of approved acronyms and abbreviations.*

[Rule 3.7] *The acronyms and abbreviations listed in the Appendix A MUST always be used in place of the word or phrase they represent.*

[Rule 3.8] *Acronyms MUST appear in all upper case except for when the acronym is the first set of characters of an attribute in which case they will be all lower case.*

Example

```
MachineID
```

[Rule 3.9] *Element and attribute names SHOULD include suffixes from the table of representation types found in the Appendix B (adapted from ebXML) when appropriate.*

4 ELEMENTS AND ATTRIBUTES

4.1 Naming conventions

Proper naming conventions increase the readability and understandability of the StanForD 2010 message documents.

[Rule 4.1]	*Element and attribute names SHOULD be chosen so that they identify what those elements and attributes contain.*

4.2 Using philosophy

Actual data in XML documents can locate in elements or attributes. For example, we could technically compose `Log` element using only elements as in the following example.

Example

```
<Log>
   <LogNumber>2</LogNumber>
   <AssortmentName>Pulp</AssortmentName>
   <Volume>150.0</Volume>
</Log>
```

Actual data values are the same as in example in chapter 2.6 but here we have implemented it using only a nested element structure. On the other hand, we could do following transformation to our example

Example

```
<Log logNumber="2" assortmentName="Pulp" volume="150.0"/>
```

Now `Log` element's content is empty and all values are stored in attributes.

There has been recurring mild debate about when one should use attributes and when one should use elements. There are no clear rules how we should divide our data between elements and attributes. Technically attributes are more compact way to store data but they can be only used for non-structured data i.e. they must be simple. So if there is a need to implement hierarchical structures in XML documents elements have to be used for that. In addition if we want to have values in some specific order we have to implement those using elements because attributes have no order.

The most used rule of thumb in this topic is to prefer to use attributes for metadata (i.e. data that describes actual data) as opposed to the data itself. However, this is quite rough rule because what is data and what is metadata depends heavily on who is reading your document for what purpose. In StanForD 2010 standard work this rule of thumb is followed because it improves the readability and understandability of messages. The example XML instance document in chapter 1.6 follows this rule of thumb.

[Rule 4.2]	*In general attributes SHOULD be used to store metadata and elements for actual data.*

5 TYPE DEFINITIONS

One of the most important properties of XML Schema Language is possibility to build new user defined and named data types. This enables us to design and implement reusable and customizable schema components. Hence schema development process is quite similar as designing class structures in object-oriented languages.

5.1 Overall type definition rules

The following rules relate both to simple and complex types in schema definition files. Some of these rules will become more understandable in the chapter 6.

[Rule 5.1] *If the user defined type is used in many schema modules it MUST be defined in StanForD2010CommonDefinitions schema module. If it is used only in one schema module it should be defined in that. However, if there is a vision that type could be used in future in other schema modules also it SHOULD be defined in StanForD2010CommonDefinitions schema module.*

User defined types should always be named using same kind of layout so that we achieve clear distinction between element/attribute names and type names and so our schema definitions become more readable and understandable.

[Rule 5.2] *All user defined types MUST be named using upper camel case as already presented in the rule 3.2 and additionally all the names MUST have suffix* `Type`*.*

Example

```
<xsd:complexType name="HarvesterProductionType">
   <xsd:sequence>
     <xsd:element name="Stem" type="StemType" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

[Rule 5.3] *Anonym types MUST NOT be used.*

This means that it is not possible to declare elements so that the type definition of the element is included as a part of elements declaration. So the following example element declaration is not allowed.

Example

```
<xsd:element name="Log" maxOccurs="unbounded">
  <xsd:complexType">
    <xsd:sequence>
      <xsd:element name="Volume" type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="logNumber" type="xsd:integer" use="required"/>
    <xsd:attribute name="assortmentName" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:element>
```

[Rule 5.4] *All user defined types MUST be defined globally in the schema module.*

All the global type definitions are possible structures for reuse and customization.

The XML Schema Language has predefined type `xsd:anyType.` Using this type we can have following kinds of element declarations.

Example

```
<xsd:element name="Anything" type="xsd:anyType"/>
```

The content of the element declared in this way is unconstrained, so the element value may be 150, but it may be any other sequence of characters as well, or indeed a mixture of characters and elements. This kind of flexibility is not an objective of StanForD 2010.

[Rule 5.5] *The predefined XML Schema Language type `xsd:anyType` MUST NOT be used.*

It is also possible to declare elements without types.

Example

```
<xsd:element name="Anything"/>
```

In this case also the type for element `Anything` is `xsd:anyType` because it is a default type if type attribute is missing.

[Rule 5.6] *Element declaration MUST always specify type.*

5.2 Simple types

Simple types are strings that don't contain any child elements, but might be constrained to be numeric or otherwise specially-formatted. For example attribute values are always simple types. Simple types can be either primitive or derived. Primitive types cannot themselves be defined from any smaller components. Derived types are defined in terms of an existing type.

XML Schema specification provides a set of predefined simple types, known as built-in types. Altogether there are 44 built-in types and 19 of those are primitive ones and 25 are derived ones.

Whenever it is suitable in StanForD 2010 schema development we should use built-in types for simple types. However, because the set of the predefined types is quite big and there are sometimes many alternatives for certain kind of types (e.g. integers), we should define proper subset built-in types that should be used in StanForD 2010.

[Rule 5.7] *StanForD 2010 Schema definitions SHOULD use only the built-in types found in the table 1*

Built-in type	Description	Examples (delimited with commas)
string	a character string of any length	this is a string
boolean	a two-state “true” or “false” flag	true, false, 1, 0
decimal	a decimal number of arbitrary precision	12.456, -12.0
dateTime	a specific instance in time	2008-10-01T11:30:23
integer	any integer number	12, 0, -400
negativeInteger	any integer number with a value < 0	-12, -400
positiveInteger	any integer number with a value > 0	12, 400
nonPositiveInteger	any integer number with a value <= 0	-400, -12, 0
nonNegativeInteger	any integer number with a value >= 0	0, 12, 400

Table 1

If there is a need for simple types that are not found from built-in types, they must be derived from existing type (base type) either by extension or restriction. The following example defines user named simple type `TreeSpeciesNameType`. Elements or attributes that are declared to be this type can have a value that is `Pine`, `Spruce` or `Birch` and nothing else.

Example

```
<xsd:simpleType name="TreeSpeciesNameType">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="Pine"/>
  <xsd:enumeration value="Spruce"/>
  <xsd:enumeration value="Birch"/>
 </xsd:restriction>
</xsd:simpleType>
```

The other example defines a new simple type for decimal numbers that are not negative i.e. they are greater or equal than zero.

Example

```
<xsd:simpleType name="NonNegativeDecimalType">
 <xsd:restriction base="xsd:decimal">
  <xsd:minInclusive value="0"/>
 </xsd:restriction>
</xsd:simpleType>
```

[Rule 5.8] *User defined simple data types SHOULD be used when there are needs to increase the validation power of the schema*

5.3 Complex types

Elements are considered complex types if they contain child elements or attributes. The simplest form of complex type definition is maybe the case when simple content element has one attribute. The following complex type example defines `VolumeType` and it has one attribute `measurementUnit` for defining what the unit of the volume element's value is.

Example

```
<xsd:complexType name="VolumeType">
 <xsd:simpleContent>
  <xsd:extension base="xsd:float">
   <xsd:attribute name="measurementUnit" type="MeasurementUnitType"/>
  </xsd:extension>
 </xsd:simpleContent>
</xsd:complexType>
```

The following Log example is using the new volume structure defined above.

Example

```
<Log logNumber="1" assortmentName="Saw log">
   <Volume measurementUnit="cubic meter">400.0</Volume>
</Log>
```

In the same way as in simple types new complex types can be derived from existing complex types either using extension or restriction. However, derivation by restriction in complex types is generally considered to be its most complex parts of XML Schema Language. This kind of derivation does not map to concepts in either object oriented programming or relational database concepts, which are the primary producers and consumers of XML data. This is the exact opposite of the situation with derivation by extension of complex types.

[Rule 5.9]	*Complex type extension MAY be used where appropriate. Complex type restriction MUST NOT be used.*

5.4 Attribute groups

In XML attributes are way to store information when the information is not structured (vs. elements) and there is need for specific order between attributes. Because these limitations there are not same way possible to define "complex types" for attributes as we can do with elements. However, XML Schema Language provides attribute group concept for improving reusability in schema development.

Attribute group is defined in schema using `xsd:attributeGroup` element. The purpose of that element is to group a set of attribute declarations so that they can be incorporated as a group into complex type definitions.

We again use Log part of the HarvesterProduction message as an example to clarify the attribute group concept. The Log is maybe not the best example for explaining attribute groups when we think about the real StanForD 2010 messages, but because our HarvesterProduction message example in this document is so simple we are satisfied with it. First the actual `Log` element

```
<Log logNumber="2" assortmentName="Pulp">
   <Volume>150.0</Volume>
</Log>
```

and then the corresponding schema definition for `Log`

```
<xsd:complexType name="LogType">
  <xsd:sequence>
    <xsd:element name="Volume" type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="logNumber" type="xsd:integer" use="required"/>
  <xsd:attribute name="assortmentName" type="xsd:string" use="required"/>
</xsd:complexType>
```

If we want to group the attribute declarations of the `LogType` into attribute group, schema definition for that is

```
<xsd:attributeGroup name="LogAttributeGroup">
  <xsd:attribute name="logNumber" type="xsd:integer" use="required"/>
  <xsd:attribute name="assortmentName" type="xsd:string" use="required"/>
<xsd:attributeGroup>
```

When `LogAttributeGroup` is used in schema definition of `LogType` it looks like this

```
<xsd:complexType name="LogType">
  <xsd:sequence>
    <xsd:element name="Volume" type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="LogAttributeGroup"/>
</xsd:complexType>
```

We must remember, firstly, that the order of the attribute declarations in attribute group have no meaning for attributes in actual log instances because in XML attributes do not have order relation. Secondly, this is only concept that helps our schema development work and so the actual instances look exactly identical if we are using attribute groups or not.

Then some rules for using attribute groups in StanForD 2010.

[Rule 5.10]	*User defined attribute groups SHOULD be used when there is possibility to reuse attribute groups in schema definitions. Attribute groups MUST be defined in StanForD2010CommonDefinitions schema module if there is possibility that they are used in multiple schemas.*

[Rule 5.11]	*All user defined attribute groups MUST be named using upper camel and additionally all the names MUST have suffix `AttributeGroup`.*

6 XML SCHEMA DESIGN PATTERNS

It is possible to provide schema definitions for one instance document i.e. StanForD 2010 message in many alternative ways when XML Schema Language is used as a schema definition language. The main goals we try to achieve when deciding what alternative should be used are reusability, customization possibility and easiness to develop instance documents.

The following XML instance document is simplified example of the original example that was represented in chapter 1.6. All the attributes have been removed so that the following different kinds of schema definitions are easier to understand.

```
<?xml version="1.0" encoding="UTF-8"?>
<HarvesterProduction>
   <Stem>
      <Log>
         <Volume>400.0</Volume>
      </Log>
      <Log>
         <Volume>150.0</Volume>
      </Log>
   </Stem>
</HarvesterProduction>
```

In addition namespaces and versioning issues are represented in the chapters 7 and 8 so those things are also omitted from these schema examples. In the following chapters four well known XML schema design patterns are represented. All of them validate XML instance document seen above.

To understand differences of the schema design patterns it is important to understand what is the difference between local and global definitions of schema components (elements or types). A global schema component (element or type) is the one that is declared immediately under the root element `<xsd:schema>` of the schema definition. Local components instead are not declared immediately under the root element. The global components are important because those are the ones that can be reused in the same schema module and also all the other schema modules.

6.1 Russian Doll

The first schema design pattern is called Russian Doll. It has exactly one global element declaration. All the other element declarations are local and nested the global one. In addition all the type definitions are unnamed for each corresponding element declaration. The schema code for Russian Doll is on the next page.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="HarvesterProduction">              ← Global element
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Stem" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Log" maxOccurs="unbounded">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="Volume" type="xsd:decimal"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Because Russian Doll design pattern does not support reusability and type customization, it is not proper choice for StanForD 2010 schema design.

6.2 Salami Slice

In Salami Slice design pattern all element declarations are global. In this case all nested element references are implemented using `ref`-attribute and the globally declared element name.

Example

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

  <xsd:element name="HarvesterProduction">          ← Global element
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Stem" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="Stem">                         ← Global element
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Log" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="Log">                          ← Global element
    <xsd:complexType>
```

```
    <xsd:sequence>
      <xsd:element ref="Volume"/>
    </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="Volume" type="xsd:decimal"/>    ← Global element

</xsd:schema>
```

All global elements are reusable both in the same schema module and other schema modules. However customization goal is not achieved because there are no globally named type definitions. All type definitions are anonymous as a part of element declarations. In addition this design pattern provides us a valid instance document for each global element name. One valid document could be for example:

```
 <Volume>400.0</Volume>
```

However, this is not a desirable feature because StanForD 2010 is not a standard for general reusable components.

6.3 Venetian Blind

The main idea in Venetian Blind schema design pattern is that we have only one global element declaration but many global type definitions. All the global type definitions are possible structures for reuse in the same and other schema modules. In addition those type definitions are also customization structures. It is possible to provide new type definitions using the existing ones either restricting or increasing them.

Example

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

 <xsd:element name="HarvesterProduction" type="HarvesterProductionType"/>
   ← Global element

 <xsd:complexType name="HarvesterProductionType">          ← Global type
   <xsd:sequence>
     <xsd:element name="Stem" type="StemType" maxOccurs="unbounded"/>
   </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="StemType">                         ← Global type
   <xsd:sequence>
     <xsd:element name="Log" type="LogType" maxOccurs="unbounded"/>
   </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="LogType">                          ← Global type
   <xsd:sequence>
     <xsd:element name="Volume" type="xsd:decimal"/>
   </xsd:sequence>
 </xsd:complexType>

</xsd:schema>
```

6.4 Garden of Eden

The Garden of Eden schema design pattern is a combination of Salami Slice and Venetian Blind because both element declarations and type definitions are global.

Example

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

  <xsd:element name="HarvesterProduction">                    ← Global element

  <xsd:complexType name="HarvesterProductionType">            ← Global type
    <xsd:sequence>
      <xsd:element ref="Stem" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="Stem" type="StemType"/>                  ← Global element

  <xsd:complexType name="StemType">                           ← Global type
    <xsd:sequence>
      <xsd:element ref="Log" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="Log" type="LogType"/>                    ← Global element

  <xsd:complexType name="LogType">                            ← Global type
    <xsd:sequence>
      <xsd:element ref="Volume"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="Volume" type="xsd:decimal"/>             ← Global element

</xsd:schema>
```

Garden of Eden schema design pattern support both type reusability and customization. However, because all elements are globally declared they are all also valid instance documents which is always desirable feature.

6.5 Schema design pattern for StanForD 2010

StanForD 2010 schemas should support reusability and customization. Hence the decision which schema design pattern development work should follow must be made between Venetian Blind and Garden of Eden. StanForD 2010 is the standard of a set of well defined messages. It is not meant to be a general library of standard reusable and customizable components, which might need globally declared elements. Therefore Venetian Blind schema design pattern is enough for StanForD 2010 schema development process.

[Rule 6.1]	*Schema development work in StanForD 2010 SHOULD follow Venetian Blind schema design pattern.*

7 NAMESPACES

A namespace is a uniquely named collection of names for elements, attributes and types that can unambiguously distinguish the collection from other collection of names in another namespace. XML Namespace is W3C recommendation and it is the mechanism to solve name conflicts that might occur if information from different sources is merged. The negative thing is that using namespaces increases both syntactical and semantical complexity in XML schemas and instance documents. However, using proper techniques it is possible to hide additional complexity in most cases.

7.1 Namespace Uniform Resource Identifiers

XML Namespace recommendation specifies that an XML namespace is identified i.e. named by a URI (Uniform Resource Identifier, RFC 2396 – http://www.ietf.org/rfc/rfc2396.txt) reference. There are two syntactically different alternatives for URIs: URN or URL.

URN (Uniform Resource Name) is a subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable and for that reason URNs are easy to conceptualize as a name and not a location.

URL (Uniform Resource Locator) refers to a subset of URI that identify resources via representation of their primary access mechanism rather that identifying the resource by name of that resource. URLs are fundamental to the World Wide Web, because all web-addresses are valid URLs.

Namespace names have two desirable properties. They should be persistent and resolvable. URNs are persistent but they are not resolvable, whereas URLs are resolvable but they are not persistent.

We see that persistence is more important property for namespace than resolvability so StanForD 2010 namespace is named according to the following URN syntax.

[Rule 7.1] *The namespace names for StanForD 2010 schemas follow URN specification and MUST be of the form:*

urn:skogforsk:stanford2010

7.2 Declaring namespaces in schemas

In general namespaces are declared as an attribute of an element. Namespaces in StanForD 2010 are declared only in attributes of the root element of the XML schema module. A namespace is declared generally in `xsd:schema` element as follows:

Example

```
<xsd:schema xmlns:sfd="urn:skogforsk:stanford2010">
```

In the attribute `xmlns:sfd`, `xmlns` is like a reserved word, which is used only to declare a namespace and `sfd` is a abbreviation prefix for usually quite long namespace name. So, the above example is read as binding the prefix "sfd" with the namespace "`urn:skogforsk:stanford2010`"

The StanForD 2010 namespace is called target namespace where the newly defined elements and attributes will reside. Declaring target namespace has a little bit different syntax than the general one.

[Rule 7.2] *Every StanForD 2010 defined schema module, MUST have a namespace declared using the `xsd:targetNamespace` attribute.*

Example

```
targetNamespace="urn:skogforsk:stanford2010"
```

In general we have to use namespace prefix to qualify our element and attribute names in schema files. If we declare our StanForD 2010 namespace as default then any element within the scope of the declaration will be qualified implicitly and no prefixes are needed. However, there could be only one default namespace in existence, so all names from other namespaces have to be qualified. For example all the names from the schema namespace `http://www.w3.org/2001/XMLSchema` must be qualified using `xsd` prefix.

[Rule 7.3] *Every StanForD 2010 defined schema module, MUST have default namespace declared using the `xsd:xmlns` attribute.*

Example

```
xmlns="urn:skogforsk:stanford2010"
```

In StanForD 2010 we have decided to have only one target namespace for all schema modules. Other option could have been to have different namespace for each message schema module. That solution would have made possible to have different version for every message independently of each other. However because most of the schema "stuff" is put into StanForD2010CommonDefinitions schema module, there is no need for finer versioning politics.

[Rule 7.4]	*Every StanForD 2010 defined schema module MUST have common unique namespace. However there is one exception: StanForD2010CodeList schema module has its own namespace "urn:skogforsk:stanford2010:codelist"*

All the possible other schemas that are used in constructing StanForD 2010 messages belong to other namespaces than StanForD 2010 namespace.

[Rule 7.5]	*StanForD 2010 namespaces MUST only contain StanForD 2010 developed schema modules.*

Because we have defined that there is always default namespace defined in schema modules there is normally no need to add namespace names (or prefixes) in element and attribute names. However there is one exception when we have to use them. If we define unique- and key/keyref -constraints in our schemas we have to include targetnamespace prefix for all element names that are used in selector and field parts of the constraint definition. Therefore we have to declare that namespace prefix in all our schema modules.

[Rule 7.6]	*Every StanForD 2010 defined schema module, MUST define a prefix* `sfd` *as an abbreviation to the target namespace.*

Example

```
xmlns:sfd="urn:skogforsk:stanford2010"
```

Because we add documentation text inside our schema definitions (in chapter 11.2), it is desirable to separate it clearly from actual data definitions. This is achieved by defining all the documentation text inside own namespace reserved for that purpose only.

[Rule 7.7]	*Every StanForD 2010 defined schema module, MUST define a namespace xmlns:doc="urn:skogforsk:stanford2010:doc"*

Example

```
xmlns:doc="urn:skogforsk:stanford2010:doc"
```

8 VERSIONING

StanForD 2010 has adopted a three-layer versioning scheme: major, minor and status. Major, minor and status version information is captured within the xsd: version attribute of the xsd:schema element. Additionally all the schema file names have major and minor version information.

8.1 Overall versioning rules

XML Schema Language have predefined place to store versioning information. This version attribute is included in the xsd:schema element. It is natural to store the versioning information there because it is possible that some XML tools could use it from there.

[Rule 8.1] *The xsd:schema version attribute MUST always be declared.*

The versioning information stored in version attribute must be detailed versioning information.

[Rule 8.2] *The xsd:schema version attribute MUST use the following template:*

<xsd:schema ... version=”draft” |” release”_ <major>.<minor> >

Where:

draft | release – is used based upon the status.
<major> - sequential number of the major version.
<minor> - sequential number of the minor version.

Example

```
<xsd:schema… version="draft_1.0">
```

Other natural and very often used place to store versioning information is the names of the schema files.

[Rule 8.3] *Version information in StanForD 2010 schema file names <MessageSchemaModuleName>_<Version>.xsd. (Rule 2.4 in chapter 2.2) MUST use the following format:*

<Version> => V<major>"p"<minor>

Where:

<major> - sequential number of the major version.
<minor> - sequential number of the minor version.

Example

HarvesterProduction_V1p0.xsd

Version information and history of the schemas must be managed carefully. It is clear that all the companies that are using StanForD 2010 are not using the exactly the same version of the standard.

[Rule 8.4] *StanForD 2010 web site MUST have*

- *all schema files that have status "release"*
- *version history of those schema files*
- *change history of those schema files*

This means that schema files that have status "standard" are never removed.

8.2 Major versions

A major version of a StanForD 2010 schema file is not backward compatible to previous major versions of the schema. If any StanForD 2010 message instance based on an older major version of schema is validated against a newer version, it should provide validation errors. A new major version will be produced when non-backward compatible changes occur. This would include the following changes:

- Removing or changing values in enumerations
- Changing of element names, type names and attribute names
- Changing the structures so as to break polymorphic processing capabilities
- Deleting or adding mandatory elements or attributes
- Changing cardinality from mandatory to optional

Major version numbers will be based on logical progressions to ensure semantic understanding of the approach and guarantee consistency in representation.

[Rule 8.5] *Every StanForD 2010 schema major version number MUST be a sequentially assigned incremental integer greater than zero.*

8.3 Minor versions

Within a major version of a StanForD 2010 schema file there can be a series of minor, or backward compatible, changes. The minor versioning of the schemas helps to identify backward and forward compatibility. Minor versions will only be increased when compatible changes occur, i.e.

- Adding values to enumerations
- Add optional elements or attributes

[Rule 8.6] *Minor versioning MUST be limited to declaring new optional XML content, extending existing XML content, or refinements of an optional nature.*

If the changes in some StanForD 2010 schema module means that previously valid instance documents become invalid the change has not been a minor one.

[Rule 8.7]	*StanForD 2010 schema module minor version changes MUST not break semantic compatibility with prior versions that have same major version number.*

Minor version numbering follows corresponding major one. Additionally minor version number zero is also allowed.

[Rule 8.8]	*Every StanForD 2010 schema minor version number MUST be a sequentially assigned incremental non-negative integer.*

9 STRUCTURE OF SCHEMA FILES

In the following chapters the physical layout of StanFord 2010 schema files is presented. Chapters 9.1 and 9.2 present the structure of the files as whole and in the following chapters most important parts are explained more detailed way.

9.1 Structure of StanForD 2010 message schema files

The following rule defines the structure of StanForD 2010 message schema files using pseudo code and after that there is a whole schema example of the HarvesterProduction message complies with the presented layout.

[Rule 9.1] *StanForD2010 message schema files, except "StanForD2010Common Definitions.xsd" schema module, MUST conform to the following physical layout as applicable:*

```
<!-- ======= XML Declaration======== -->
<?xml version="1.0" encoding="UTF-8"?>
<!-- ======= Schema Header ======= -->
Document Name:
Generated On:
Version number:
Schema agency:
Schema web address:
<!-- ===== xsd:schema Element With Namespaces Declarations ===== -
            ->
xsd:schema element to include version attribute and namespace dec-
            larations:
            xmlns:xsd
            Target namespace
            Default namespace
            Prefix definition for target namespace
            Documentation namespace
            All imported namespaces
            Attribute Declarations
               elementFormDefault="qualified"
               attributeFormDefault="unqualified"
            Version Attribute
<!-- ===== Includes ===== -->
StanForDCommonDefinitions schema module
<!-- ===== Imports ===== -->
Optional schema modules
<!-- ===== Root Element ===== -->
Root Element Declaration
<!-- ===== Type Definitions ===== -->
Root Element Type Definition
All other schema specific type definitions
```

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ********************************************************** -->
<!-- ***  HarvesterProduction XML Schema file                *** -->
<!-- ********************************************************** -->
<!--

   Document Name:      HarvesterProduction_V1p0.xsd
   Generated On:       16 December 2008
   Version number:     1.0 draft
   Schema agency:      Skogforsk
   Schema web address: http://www.skogforsk.se/stanford2010/schema/

-->
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
    targetNamespace="urn:skogforsk:stanford2010"
    xmlns="urn:skogforsk:stanford2010"
    xmlns:sfd="urn:skogforsk:stanford2010"
    xmlns:doc="urn:skogforsk:stanford2010:doc"
    elementFormDefault="qualified" attributeFormDefault="unqualified"
    version="draft_1.0">

  <xsd:include schemaLocation="StanForD2010CommonDefinitions_V1p0.xsd"/>

  <xsd:element name="HarvesterProduction" type="HarvesterProductionType"/>

  <xsd:complexType name="HarvesterProductionType">
    <xsd:sequence>
      <xsd:element name="Stem" type="StemType" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

9.2 Structure of "StanForD2010CommonDefinitions.xsd" schema module

The following rule defines the structure of StanForDCommonDefinitions.xsd file using pseudo code and after that there is a whole schema example complies with the presented layout.

[Rule 9.2] *"StanForD2010Common Definitions.xsd" schema module MUST conform to the following physical layout as applicable:*

```
<!-- ======= XML Declaration======== -->
<?xml version="1.0" encoding="UTF-8"?>
<!-- ======= Schema Header ======= -->
Document Name:
Generated On:
Version number:
Schema agency:
Schema web address:
<!-- ===== xsd:schema Element With Namespaces Declarations === -->
xsd:schema element to include version attribute and namespace dec-
            larations:
            xmlns:xsd
```

```
            Target namespace
            Default namespace
            Prefix definition for target namespace
            Documentation namespace
            All imported namespaces
            Attribute Declarations
              elementFormDefault="qualified"
              attributeFormDefault="unqualified"
            Version Attribute
<!-- ===== Imports ===== -->
Optional schema modules
<!-- ===== Type Definitions ===== -->
All common type definitions
All common attribute group definitions
```

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ********************************************************* -->
<!-- ***  StanForD2010CommonDefinitions XML Schema file      *** -->
<!-- ********************************************************* -->
<!--

   Document Name:      StanForD2010CommonDefinitions_V2p0.xsd
   Generated On:       16 June 2011
   Version number:     2.0 draft
   Schema agency:      Skogforsk
   Schema web address: http://www.skogforsk.se/stanford2010/schema/

-->
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
   targetNamespace="urn:skogforsk:stanford2010"
   xmlns="urn:skogforsk:stanford2010"
   xmlns:sfd="urn:skogforsk:stanford2010"
   xmlns:doc="urn:skogforsk:stanford2010:doc"
   elementFormDefault="qualified" attributeFormDefault="unqualified"
   version="draft_2.0">

  <xsd:complexType name="StemType">
    <xsd:sequence>
      <xsd:element name="Log" type="LogType" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="stemNumber" type="xsd:integer" use="required"/>
    <xsd:attribute name="treeSpeciesName" type="xsd:string" use="required"/>
    <xsd:attribute name="barkParameterCode" type="xsd:integer"/>
  </xsd:complexType>

  <xsd:complexType name="LogType">
    <xsd:sequence>
      <xsd:element name="Volume" type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="logNumber" type="xsd:integer" use="required"/>
    <xsd:attribute name="assortmentName" type="xsd:string" use="required"/>
  </xsd:complexType>

</xsd:schema>
```

9.3 XML declaration

All standard XML files have XML declaration as a first line of document.

[Rule 9.3] *All StanForD 2010 schema files MUST have following XML declaration:*

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

There is W3C Recommendation for XML version 1.1. However the main issue that version 1.1 adds to the version 1.0 is support to use "exotic" characters in element and attribute names. We have decided to use only English names in element and attribute names so there is no need to use version 1.1. Additionally and more importantly, there is no software and tool support for version 1.1.

The encoding attribute should be UTF-8 (Unicode) because it is the most used and best supported character set today. If we are using very exotic languages (from our point of view) there is possible that UTF-8 does not include all necessary characters, and then we have to use UTF-16 instead.

9.4 Schema header

Every schema definition file must have comment section after XML declaration that includes the information and layout defined in the following rule.

[Rule 9.4] *Each StanForD 201 schema file MUST have comment section with following layout and information:*

```
<!-- ************************************************** -->
<!-- ***  [Schema module name] XML Schema file          *** -->
<!-- ************************************************** -->

  Document Name:
  Generated On:
  Version number:
  Schema agency:
  Schema web address:

-->
```

Example

```
<!-- ************************************************** -->
<!-- ***  HarvesterProduction XML Schema file           *** -->
<!-- ************************************************** -->
<!--

   Document Name:      harvesterproduction_V2p0.xsd
   Generated On:       16 June 2008
   Version number:     2.0 draft
   Schema agency:      Skogforsk
   Schema web address: http://www.skogforsk.se/stanford2010/schema/

-->
```

NB! The www address in the example above is not necessarily correct one.

9.5 xsd:schema element

`xsd:schema` element have many common attribute declarations in each StanForD 2010 schema file.

Example

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
   targetNamespace="urn:skogforsk:stanford2010"
   xmlns="urn:skogforsk:stanford2010"
   xmlns:sfd="urn:skogforsk:stanford2010"
   xmlns:doc="urn:skogforsk:stanford2010:doc"
   elementFormDefault="qualified" attributeFormDefault="unqualified"
   version="draft_2.0">
```

The following rules define all the attributes and their values.

[Rule 9.5] *Each StanForD 2010 schema file MUST declare namespace for XML Schema Language and prefix `xsd` for it as follows:*

`xmlns:xsd=http://www.w3.org/2001/XMLSchema`

All XML Schema Language element and attribute names are prefixed using `xsd` prefix.

The rules for targetNamespace and default namespace declarations are already presented in chapter 7.2.

[Rule 9.6] *`elementFormDefault` attribute MUST have value `"qualified"` and `attributeFormDefault` MUST have value `"unqualified"`.*

The above rule defines values for two attributes: `elementFormDefault` and `attributeFormDefault.` The values of the attributes are most used best practices in situation when we are using Venetian Blind schema design pattern and we have one common namespace for all schema modules.

Rules for version attribute `version` are already presented in chapter 8.1.

9.6 xsd:include element

All StanForD 2010 message schema modules are using common type definitions from StanForD2010CommonDefintions schema module.

[Rule 9.7] *Each StanForD 2010 message schema module MUST have following declaration*

```
<xsd:include schemaLocation="StanForD2010CommonDefinitions_<version>.xsd"/>
```

where `<version>` *is version information as defined in chapter 8.1*

`xsd:include` is a mechanism to include other schema modules in to current one. All the included schema modules must belong to same target namespace. The value of `schemaLocation` attribute is defined to be URI and it is only a hint for processors and applications where to look the needed schema module. In most cases StanForD2010CommonDefininitions schema module should be in a same place (e.g. directory) as the schema module of the message so the pure file name is enough.

9.7 xsd:import element

If there is a need to use schema modules defined outside StanForD 2010 they are imported in message schema modules using `xsd:import` element.

[Rule 9.8] *Each StanForD 2010 schema modules that import other schemas MUST have following declaration*

```
<xsd:import namespace=<othernamespace>
            schemaLocation=<location>/>
```

where `<othernamespace>` *is a namespace URI of the imported schema and* `<location>` *is hint for a processor where to find imported schema module.*

Additionally the imported schema modules MUST be declared in `xsd:schema` *element with proper prefixes so that they can be used properly.*

10 EXTENSIBILITY

Information needs are changed every now and then and so must standards that define that information. In StanForD 2010 there are needs for both machine manufacturers and users to do tests that need information exchange that is not yet standardized and maybe never is to be standardized. It has to be possible in StanForD 2010 to include non-standardized information in standard messages. However, the way to do it must be well defined and clear. XML Schema Language defines a wildcard element `xsd:any` for this purpose. The basic idea behind that is in place where that wildcard exist could be any well-formed XML structure. The xsd:any elements structure is as follows:

```
<xsd:any namespace="##any" processContents="lax"
         minOccurs="0" maxOccurs="unbounded"/>
```

`xsd:any` element have four attributes that are important to understand. The first one `namespace` attribute can have values:

- `"##any"` Element and attribute names can be from any namespace or they might not have namespace
- `"##other"` Element and attribute names come from some other namespaces than our targetnamespace
- `"##targetnamespace"` Element and attribute names are from our targetnamespace

The second attribute `processContents` can have values:

- `lax` Validation is performed if possible.
- `skip` Validations not done.
- `strict` Validation is enforced.

`minOccurs` and `maxOccurs` attributes are limiting the number of top level elements in the place of the wildcard.

It is in principle possible to put the wildcard declaration multiple places in StanForD 2010 schema definitions. The most flexible way to do it is to put the wildcard declaration between all element declarations in schema definitions and so it is possible to add extensible content in every place of StanForD 2010 messages. However such flexibility is not desirable feature in StanFord 2010 standard which should provide well defined messages between wood production actors. Hence we define more strict way to provide extensibility in StanForD 2010. Firstly we encapsulate the wildcard declaration inside type definition `ExtensionType` as follows.

Example

```
<xsd:complexType name="ExtensionType">
  <xsd:sequence>
    <xsd:any namespace="##any" processContents="lax"
             minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
```

We choose attribute values so that they provide us as much flexibility as possible and if necessary validation of extent content might be possible. Secondly we declare `Extension` element in every StanForD 2010 message schema in the same place. `Extension` element has to be the last element in message type definition. Naturally `Extension` element is optional. The following Harvesterproduction schema definition has `Extension` element declaration

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  . . .
 <xsd:element name="HarvesterProduction" type="HarvesterProductionType"/>
 <xsd:complexType name="HarvesterProductionType">
  <xsd:sequence>
   <xsd:element name="Stem" type="StemType" maxOccurs="unbounded"/>
   <xsd:element name="Extension" type="ExtensionType" minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

Then we have example of Harvesterproduction message that have `Extension` element that includes both some elements with or without namespace.

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<sfd:HarvesterProduction
    xmlns:sfd ="urn:skogforsk:stanford2010"
    xmlns="urn:skogforsk:stanford2010"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ext="urn:extension"
    xsi:schemaLocation=
     "urn:skogforsk:stanford2010:1:draft HarvesterProduction_V1p0.xsd">
 <Stem stemNumber="1" treeSpeciesName="Pine" barkParameterCode="1">
  <Log logNumber="1" assortmentName="Saw log">
   <Volume>400.0</Volume>
  </Log>
  <Log logNumber="2" assortmentName="Pulp">
   <Volume>150.0</Volume>
  </Log>
 </Stem>
 <Extension>
  <ext:Element1>value1</ext:Element1>
  <ext:Element2>value2</ext:Element2>
  <NonamespaceElement>
     <NestedValue>value3</NestedValue>
  </NoNamespaceElement>
 </Extension>
</sfd:HarvesterProduction>
```

Finally we present the rule that defines the use of the extension in StanForD2010.

[Rule 10.1]	*In StanForD 2010 all not standardized data MUST be included in predefined Extension elements. Extension elements are optional in every StanForD 2010 mes-*

sages. Additionally `xsd:any` *or* `xsd:anyAttribute` *elements MUST NOT be used.*

11 MISCELLANEOUS XML SCHEMA RULES

11.1 Element content compositors

XML Schema Language provides three compositors for grouping nested element declarations inside an element. Those compositors are `sequence`, `choice` and `all`. All schema definition examples in this document are using `sequence` compositor. Each of these compositors has different rules governing how the elements declared in schema can appear in instance documents. The rules for each compositor are following

- `sequence` means that the elements declared inside it must appear in the same order in which they are declared.
- `choice` means that only one declared element inside it can occur in the instance document.
- `all` allows the declared elements to occur in any order. Additionally, they can occur only once or not at all.

Some rules that are restricting the use of certain compositors in StanForD 2010.

[Rule 11.1]	*The `xsd:all` compositor MUST NOT be used in StanForD 2010 schema definitions.*

[Rule 11.2]	*The `xsd:choice` compositor SHOULD NOT be used in StanForD 2010 schema definitions if there is need for customization and extensibility in structures where that compositor is used.*

Because StanFord 2010 is standard for data-centric information (vs. document-centric) there is no need for compositor that allows elements occur in any order.

11.2 Annotation and documentation

The schema definitions of StanForD 2010 are most important documents of the standard. However, without any other documents the meaning and the semantics of the standards structures and data content is not understandable enough.

XML Schema Language allows schema components to be annotated using the `<annotation>` element. The annotation element can contain one or more `<documentation>` or `<appinfo>` elements that can themselves have any attributes and contain any text or child elements.

[Rule 11.3]	*StanForD 2010 schemas MUST be documented using `annotation` and `documentation` elements. `appinfo` element MUST NOT be used. Additionally, the XML comment notation MUST NOT be used for documentation. The format and content of the documentation will be defined in StanForD 2010 schema definition working group.*

Although the schema annotation is necessary, its volume results in a considerable increase in the size of the StanForD 2010 schemas. To address this issue, two schemas will be developed for each

StanForD 2010 schema module. A normative, fully annotated schema will be provided to facilitate greater understanding of the schema module. A non-normative schema without annotation will also be provided that can be used for validation purposes especially at run-time.

[Rule 11.4] *StanFord 2010 MUST provide two schemas for each schema module. One normative schema shall be fully annotated. One non-normative schema shall be without annotation.*

11.3 Other rules

In XML instance documents it is possible that element have mixed content, in which case elements are allowed to have both child elements and textual content. Including mixed content in StanFord 2010 messages is undesirable because message transactions are based on exchange of discrete pieces of data that must be clearly unambiguous.

[Rule 11.5] *Mixed content MUST NOT be used.*

XML Schema Language has two alternatives to provide reference constraints between elements. `xsd:key/xsd:keyref` must be used because the use of `xsd:ID/xsd:IDREF` has limitations.

[Rule 11.6] *`xsd:key/xsd:keyref` MUST be used for element referencing `xsd:ID/xsd:IDREF` MUST NOT be used.*

xsd:key/xsd:keyref constraint is also unique constraint i.e. every key value is unique inside on instance document. If there is other values than keys that should also be unique `xsd:unique` constraint should be used.

[Rule 11.7] *`xsd:unique` SHOULD be used where appropriate.*

XML Schema Language has properties that allows fixed or default values to be specified for elements and attributes. Exploiting these properties require that schema must be present at the time the corresponding instance document is created otherwise those values are missing from the document. This is such serious restriction that these properties are not allowed in StanForD 2010 schemas.

[Rule 11.8] *`xsd:fixed` and `xsd:default` attributes MUST NOT be used.*

A best practice is to deny the use of `xsd:notation` attribute.

[Rule 11.9] *`xsd:notation` MUST NOT be used.*

12 STANFORD 2010 INSTANCE DOCUMENTS

In this chapter we present rules and issues that are specific to StanForD 2010 instance documents i.e. messages.

12.1 Validation

It is self-evident that all StanForD 2010 messages must be validated against schema. If they don't validate they are not StanForD 2010 conformant messages.

[Rule 11.1] *All StanForD 2010 messages MUST validate to a corresponding schema.*

12.2 Structure of StanForD 2010 messages

The following rule defines the common structure of StanForD 2010 message files using pseudo code and after that there is a whole message example complies with the presented layout.

[Rule 12.2] *StanForD 2010 messages MUST conform to the following physical layout as applicable:*

```
<!-- ======= XML Declaration======== -->
<?xml version="1.0" encoding="UTF-8"?>
<!-- ===== root element with namespaces declarations === -->
root element to include namespace declarations in the following
            order :
            StanForD 2010 namespace
            Default namespace
            XMLSchema-instance namespace
            Schema location
Content of the message
```

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<sfd:HarvesterProduction xmlns:sfd="urn:skogforsk:stanford2010"
       xmlns="urn:skogforsk:stanford2010"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="urn:skogforsk:stanford2010
harvesterProduction_V2p0.xsd">
  <Stem stemNumber="1" treeSpeciesName="Pine" barkParameterCode="1">
    <Log logNumber="1" assortmentName="Saw log">
      <Volume>400.0</Volume>
    </Log>
    <Log logNumber="2" assortmentName="Pulp">
      <Volume>150.0</Volume>
    </Log>
  </Stem>
</sfd:HarvesterProduction>
```

12.3 XML Declaration

XML declaration is completely similar than declaration in schema structure that was presented and detail explained in chapter 9.3.

12.4 Attributes in the root element of the messages

The root element of each message MUST have following declarations:

[Rule 12.3] *The root element of each StanForD 2010 message MUST have following namespace declarations and schema location attribute:*

```
xmlns:sfd="urn:skogforsk:stanford2010"
xmlns="urn:skogforsk:stanford2010"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:skogforsk:stanford2010
            <URI path to schema file>"
```

where

<version> is version information that format was presented in chapter 7.1
`<URI path to schema file>` *this is a hint for processor where to find corresponding schema file of the message.*

Prefix for StanForD 2010 namespace MUST always be `sfd.`

12.5 Empty content

In general, the absence of an element in an XML schema does not have any particular meaning - it may indicate that the information is unknown, or not applicable, or the element may be absent for some other reason. The XML Schema specification does provide a feature, the `xsd:nillable` attribute that defines that elements value could be missing i.e. it have element tags but no value. Use of empty elements within XML instance documents is a source of controversy for a variety of reasons. An empty element does not simply represent data that is missing.

[Rule 12.4] *Empty elements SHOULD NOT be used in StanForD 2010 messages. However if element have at least one mandatory and not empty attribute then the element may be empty.*

APPENDIX A. StanForD 2010 abbreviations and acronyms

Abbreviation/acronym	Actual word or phrase
ID	Identifier
GIS	Geographical Information System
DBH	Diameter Breast Height
MIN	Minimum
MAX	Maximum
MTH	Multi tree handle

APPENDIX B. Standard suffixes for certain representation types in StanForD 2010

Representation type	Description	Example
Code	A character string that represents a member of a set of values.	`AssortmentCode`
DateTime	A particular point in the progression of time.	`StartDate`
Identifier	A character string used to identify and distinguish uniquely, one instance of an object within an identification scheme. The standard abbreviation ID, meaning a unique identifier, should be used in actual names.	`MachineID`
Name	A word or phrase that constitutes the distinctive designation of a person, object, place, event, concept etc.	`AssortmentName`
Quantity	A number of non-monetary units. It is normally associated with a unit of measure.	`TotalQuantity`
Number	A numeric value which is often used to imply a sequence or a member of a series.	`LogNumber`
Text	A character string generally in the form of words.	`OptionalText`