MedBiquitous XML Schema Design Guidelines

Version 1.3

25 October 2004
MedBiquitous Technical Steering Committee

Revision History

Date	Version	Description	Author
24 Apr 2003	0.1	Initial draft for Technical Steering Committee	Darin McBeath D.McBeath@elsevier.com
16 Jul 2003	1.0	Delimiter change	Joel Farrell joelf@us.ibm.com
15 Dec 2003	1.1	Namespace updates	Joel Farrell joelf@us.ibm.com
30 Apr 2004	1.2	Added XML tagging conventions to naming conventions, added license, updated format, put annotations inside type definitions.	Valerie Smothers valerie.smothers@medbiq.org
25 Oct 2004	1.3	Added guidelines for creating extensibility points.	Joel Farrell joelf@us.ibm.com Scott Hinkelman srh@us.ibm.com

MedBiquitous Consortium XML Public License and Terms of Use

Table of Contents

XML Schema Design Guidelines

1. Acknowledgements

These guidelines are based on a submission from Darin McBeath of Elsevier. Joel Farrell and Scott Hinkelman of IBM also contributed to this document.

2. Introduction

This document suggests best practices and guidelines that should be followed by MedBiquitous XML Schema designers when defining XML-based specifications.

2.1 Background

The MedBiquitous consortium is using XML and Web services as the basis for its specifications to promote the best possible interoperability between its members. Unfortunately, the W3C specification for XML Schema is large and complex leaving it nearly impossible for anyone to completely understand its breadth and depth. Furthermore, the W3C offers no guidance with respect to best practices or guidelines for implementing XML Schemas within the enterprise.

The purpose of this document is to establish high-level best practices and guidelines that should be followed by MedBiquitous to ensure consistency across the XML Schemas that are designed. These schemas describe data that can be exchanged between members. This document addresses many of the common features and issues pertaining to XML Schema; obscure aspects of XML Schema will not be addressed.

As the W3C XML Schema specification continues to evolve and mature, this document will be modified appropriately to keep pace with industry standard best practices and guidelines.

2.2 Resources

There are numerous resources available on the web that offers insight into best practices and guidelines for utilizing XML Schema. The following resources and personal experience were used while authoring this document.

www.xfront.com	Roger Costello of the Mitre Corporation maintains this site. He is well known within the XML community for his exhaustive tutorials on XML Schema.
www.xml.com	The publisher O'Reilly supports and maintains this informative site.
www.x12.org	The Accredited Standards Committee (ASC) X12 develops standards – in X12 and XML formats – for cross-industry electronic exchange of business information (EDI).

3. Namespaces

The following items are guidelines for defining namespaces within XML Schema definition files. After the guidelines are discussed, a partial XML Schema definition example is presented incorporating these concepts.

3.1 Target

The XML Schema definition file should define a target namespace. The namespace should be defined as a URL that uniquely qualifies this schema and its definitions.

Each MedBiquitous schema will have its own namespace. This provides a standard way to avoid name collisions between schemas that may be embedded in other XML schema definitions. The namespace string will be a URL, and will be constructed by means of a hierarchical organization that corresponds to the owning group within the

MedBiquitous consortium.

The root URL string will be http://ns.medbiq.org. This is followed by a working group qualifier (if necessary), schema qualifier, and a version specification. The version specification will be defined below.

Example:

```
targetNamespace="http://ns.medbiq.org/journals/v1/"
```

The namespace URL may provide a document that points to the schema and specifications locations, but this is not guaranteed.

3.2 Default

The XML Schema definition file should define a default namespace that is equal to the target namespace. In other words, XML Schema should not be the default namespace.

Example:

```
xmlns="http://ns.medbiq.org/journals/v1/"
```

3.3 xsd

Within the XML Schema definition file, the namespace prefix for XML Schema should be **xsd**. XML Schema should not be the default namespace.

Example:

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

3.4 Version

The default and target namespaces defined in the XML Schema definition file must include a version identification value. The value is string composed the character **v** followed by a number. Whenever the schema is changed, the version number must be incremented.

The initial public version of a schema will be version **v1**.

Example:

```
targetNamespace="http://ns.medbiq.org/journals/v1/"
```

3.5 Chameleon

Avoid creating XML Schema definition files with no target namespace ("chameleon"). Although chameleon schemas offer flexibility, validation performance is degraded since most parsers will not be able to cache components of the schema based on the namespace. In addition, care must be exercised to avoid symbol collisions when a chameleon schema is included within another schema.

3.6 Example

The following XML fragment incorporates all of the guidelines discussed in this section.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://ns.medbiq.org/journals/v1/"
```

```
xmlns="http://ns.medbiq.org/journals/v1/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
...
```

4. Design Patterns

As with software design, there are design patterns associated with XML Schema design. The most popular XML Schema design patterns are Russian Doll, Salami, Bologna, Venetian Blind, and Garden of Eden. They each have different characteristics that will be analyzed (and basic advantages and disadvantages listed) before concluding with the recommended design pattern.

To understand the following design patterns, it is necessary to differentiate between a global component (element or type) and a local component (element or type). A global component is an immediate child of the `<schema>` element in the XML Schema definition file. A local component is not an immediate child of the `<schema>` element in the XML Schema definition file. Global components are associated with the target namespace of the schema and may be reused in other schema.

It is also important to understand that any element defined in the global namespace can be the root for a valid XML instance document adhering to the schema defined for that namespace.

4.1 Russian Doll

The Russian Doll design corresponds to having a single global element that nests local elements (that nest further local elements). Only one element, considered the root, is defined within the global namespace.

Example:

```
<xsd:schema>
   <xsd:element name="Couple">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Husband" type="xsd:string"/>
            <xsd:element name="Wife" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
       </xsd:element>
</xsd:schema>
```

Advantages:

- Since there is only one global element, there is only one valid XML document.
- Namespace complexity is *potentially*[1] localized since Husband and Wife are both defined within the local namespace.

Disadvantages:

- The reusability of the schema definition is limited to the single globally defined element.

4.2 Salami

The Salami design corresponds to having all of the elements defined within the global namespace and then referencing the elements.

[1] The namespace complexity is potentially localized depending on the value of the elementFormDefault attribute. This will be discussed further later in the document.

Example:

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

   <xsd:element name="Wife" type="xsd:string"/>

   <xsd:element name="Couple">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Husband"/>
            <xsd:element ref="Wife"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Advantages:

- All elements are defined in the global namespace and are therefore reusable.

Disadvantages:

- Since there are many global elements, there are many valid XML documents. Any element defined in the global namespace can be used as the root element for a valid XML document.
- Namespace complexity is exposed since Husband and Wife are both defined within the global namespace.

4.3 Bologna [2]

This is the Oscar Mayer of design patterns since it contains a little bit of everything. Essentially the person who uses this pattern has no idea what they are doing since they randomly use a combination of global and local elements for no obvious gains or benefits. This pattern should be avoided at all cost.

Example:

```
<xsd:schema>
   <xsd:element name="Husband" type="xsd:string"”/>

   <xsd:element name="Couple">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Husband"/>
            <xsd:element name="Wife" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
        </xsd:element>
</xsd:schema>
```

Advantages:

[2] One of the authors of this document coined this term based on his early attempts to design XML Schema definition files. He has since seen the light and now evangelizes the importance of good schema design patterns.

- None, other than it works … much like a computer program compiles.

Disadvantages:

- Too many to list.

4.4 Venetian Blind

Similar to the Russian Doll, the Venetian Blind design corresponds to having a single global element that nests local elements (that nest further local elements). Only one element, considered the root, is defined within the global namespace. However, the local elements use types (simple or complex) that are defined within the global namespace.

Example:

```
<xsd:schema>
   <xsd:simpleType name="Husband">
      <xsd:restriction base="xsd:string">
         <xsd:minLength value="1"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="Wife">
      <xsd:restriction base="xsd:string">
         <xsd:minLength value="1"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:element name="Couple">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Husband" type="Husband"/>
            <xsd:element name="Wife" type="Wife"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Advantages:

- The reusability of the schema definition is available for all types and the single root element defined in the global namespace.
- Namespace complexity is *potentially*[3] localized since Husband and Wife are both defined within the local namespace.
- Since there is only one global element, there is only one valid XML document.

Disadvantages:

- No major disadvantages.

[3] The namespace complexity is potentially localized depending on the value of the elementFormDefault attribute. This will be discussed further later in the document.

4.5 Garden of Eden

The Garden of Eden is a combination of the Venetian Blind and Salami. All elements and types are defined in the global namespace with the elements referenced as needed.

Example:

```
<xsd:schema>
   <xsd:simpleType name="Husband">
      <xsd:restriction base="xsd:string">
         <xsd:minLength value="1"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:element name="Husband" type=”Husband”>

   <xsd:simpleType name="Wife">
      <xsd:restriction base="xsd:string">
         <xsd:minLength value="1"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:element name="Wife" type=”Wife”>

   <xsd:element name="Couple">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref=”Husband”/>
            <xsd:element name=”Wife”/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Advantages:

- The reusability of the schema definition is available for all types and all elements defined in the global namespace.

Disadvantages:

- Since there are many global elements, there are many valid XML documents.
- Namespace complexity is exposed since Husband and Wife are both defined within the global namespace.

4.6 Recommendation

Venetian Blind is the preferred design pattern. The benefits of a reusable type definitions coupled with a single ‘root’ element in the global namespace provide MedBiquitous with the control and reusability necessary in the messaging interface definitions.

Of course, there are always exceptions. The common schema definition file will likely be just a collection of simple and complex types and no actual element definitions. Without the global root element, this schema does not explicitly adhere to the Venetian Blind design pattern. It is also possible (and likely) that most services will receive multiple request payloads and produce multiple response payloads. In this situation, multiple root elements would be defined in the global namespace for the request and response schemas (one for each type of request and response). Although this violates the purist definition of the Venetian Blind design pattern (more than one element in the global namespace), the practicality of this approach outweighs any disadvantages.

5. Qualified and Unqualified

When a schema definition file is created, there are two attributes (elementFormDefault and attributeFormDefault) of the schema element that should be specified. Although specified in the schema definition file, the impact of these settings is not obvious until an XML instance document is constructed.

5.1 elementFormDefault

Assume the following schema definition file. Note that elementFormDefault is set to **qualified**. This ‘switch’ implies that all schema instance files must qualify the namespace of all elements regardless of whether the element is defined in a global or local namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.darin.com/example/v1"
   xmlns=" http://www.darin.com/example/1"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">

   <xsd:simpleType name="Husband">
      <xsd:restriction base="xsd:string">
         <xsd:minLength value="1"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:simpleType name="Wife">
      <xsd:restriction base="xsd:string">
         <xsd:minLength value="1"/>
      </xsd:restriction>
   </xsd:simpleType>

   <xsd:element name="Couple">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Husband" type="Husband"/>
            <xsd:element name="Wife" type="Wife"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>

</xsd:schema>
```

A valid XML instance file for the above schema follows. A default namespace was not used (which is not recommended) to show the impact of the elementFormDefault setting.

```
<?xml version="1.0" encoding="UTF-8"?>
<ex:Couple xmlns:ex=" http://www.darin.com/example/v1"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.darin.com/example
   E:\Xquery\example.xsd">
   <ex:Husband>Darin</ex:Husband>
   <ex:Wife>Darby</ex:Wife>
</ex:Couple>
```

Now, assume the original schema definition file is altered so elementFormDefault now contains the value **unqualified**. A valid XML instance file for this change is listed below. Once again, a default namespace was not used to show the impact of the elementFormDefault setting. With this setting, only the elements defined in the global namespace must be namespace qualified.

```
<?xml version="1.0" encoding="UTF-8"?>
<ex:Couple xmlns:ex=" http://www.darin.com/example/v1"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.darin.com/example
   E:\Xquery\example.xsd">
   <Husband>Darin</Husband>
   <Wife>Darby</Wife>
</ex:Couple>
```

Industry best practices recommend using the first option where the elementFormDefault value is set to **qualified**. Since the default namespace for the schema instance should normally be specified, the resulting schema will not be as verbose as the example listed above. Furthermore, by fully qualifying the schema, it is obvious what namespace to associate with a given element.

Another justification for qualifying the namespace is to improve performance. Often, when processing an instance document, the namespace is required to determine how an element should be processed. If the namespace is hidden (or unqualified), then the application must lookup the element in the schema definition file for the element, which results in slower performance.

5.2 attributeFormDefault

This attribute setting should always be set to unqualified. The time to justify the setting of this value is not worth any incremental value in the understanding of this concept. Industry best practices recommend always setting this attribute value to unqualified.

6. Elements and Attributes

The following guidelines relate to elements and attributes within XML Schema definition files.

6.1 Naming Conventions

Naming conventions are based on the XML tagging guidelines from the Open Travel Association Group (OTA) (http://www.opentravel.org/) and from the ebXML (http://www.oasis-open.org/).

A key part of the XML grammar is consistent naming conventions for tags that represent the infrastructure and business-related elements. Tag name writers MUST follow these rules unless business requirements require other naming conventions.

Naming Conventions

Rule	Description	Example
Element and Type Case	Elements and types should be defined using upper camel case.	`<PostalCode>`
Attribute Case	Attributes should be defined using lower camel case.	`<Degree discipline="Chemistry">`
Acronyms	Acronyms are discouraged, but where needed, use all upper case.	`<UserID>`
Illegal Characters	Illegal characters cannot be used (e.g.: forward slash, etc.). Recommended characters in a tag name are basically limited to letters and underscores.	NOT allowed: `<Date/Time>` Allowed: `<DateTime>`

Similar Names	Use the similar tag names with elements in a similar child structure.	`<ContactAddress>`
		`<HomeAddress>`
		`<WorkAddress>`

Plural Names	Use plural tag names only for collections.	`<CreditCards>`
		`  <CreditCard>`

Name Size	Element and attribute name size have no limitation. The names must be meaningful.	`<CustomerRelationshipInformation>`

Suffixes	Element and attribute names should incorporate suffixes from the proposed list of representation types (adapted from ebXML) when appropriate. See the Tag Suffixes table for the proposed representation types.	`<StartDate>`
		`<BilledAmount>`

Tag Suffixes

Representation Type	Description
Amount	A number of monetary units specified in a currency where the unit of currency is explicit or it may be implied.
Code	A character string that represents a member of a set of values.
Boolean	An enumerated list of two, and only two, values which indicates a Condition such as on/off; true/false etc. (It was the general consensus to use 'Flag' as a term

Representation Type	Description
	to indicate a Boolean value.)
Date	A day within a particular calendar year. Note: Reference ISO 8601.
Time	The time within any day in public use locally, independent of a particular day. Reference ISO 8601:1988.
DateTime	A particular point in the progression of time. Note: This may incorporate dependent on the level of precision, the concept of date.
Identifier	(standard abbreviation ID, meaning a unique identifier) A character string used to identify and distinguish uniquely, one instance of an object within an identification scheme.
Name	A word or phrase that constitutes the distinctive designation of a person, object, place, event, concept etc.
Quantity	A number of non-monetary units. It is normally associated with a unit of measure.
Number	A numeric value which is often used to imply a sequence or a member of a series.
Rate	A ratio of two measures.
Text	A character string generally in the form of words.
Measure	A numeric value that is always associated with a unit of measure.

6.2 Usage

In general, use elements for data that will be produced or consumed by an application and attributes for metadata. A good rule of thumb is to use elements for *nouns* and attributes for *adjectives*. In the early days of Web services, there was a push to avoid attributes in schema definitions of messaging interfaces because of SOAP encoding issues. However, MedBiquitous has adopted a literal encoding[4] approach (doc-literal) that alleviates this issue.

6.3 Global and Local

With the adoption of the Venetian Blind design pattern, only the root element should be defined in the global namespace. All other elements should be defined in the local namespace and use named types that are defined in the global namespace. Since attributes are associated with an element, they should normally be defined locally within the context of the element.

[4] Literal encoding is also endorsed by Web Services Interoperability (WS-I).

7. Types

The following guidelines relate to simple and complex types within XML Schema definition files. As a general rule, types should always be defined in the global namespace and then used by the local elements. This approach supports the Venetian Blind design strategy. Consequently, named types should be used instead of anonymous types.

7.1 Simple Types

When appropriate, simple data types defined in the XML Schema data model should be used (and potentially restricted or extended) rather than creating a user defined complex data type. Restriction of a simple type reduces the possible values of the type while extension allows one to create a complex type with simple content that has attributes.

Example:

If a date value is needed, use

```
<xsd:element name="Date" type="xsd:date"/>
```

instead of

```
<xsd:complexType name="Date">
   <xsd:sequence>
      <xsd:element name="Month" type="xsd:integer"/>
      <xsd:element name="Day" type="xsd:integer"/>
      <xsd:element name="Year" type="xsd:integer"/>
   </xsd:sequence>
</xsd:complexType>
```

The correct simple type defined in the XML Schema data model should also be used.

Example:

If a date value is needed, use

```
<xsd:element name="Date" type="xsd:date"/>
```

instead of

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

7.2 Complex Types

Complex types should be extended but not restricted. Extension involves adding extra attributes or elements to a derived type. Derivation by restriction of complex types should be avoided because the WXS specification is complex in this area and implementations are error prone.

Example:

Base Type

```
<xsd:complexType name=”BaseAddress”>
    <xsd:sequence>
        <xsd:element name=”State” type=”xsd:string”/>
    </xsd:sequence>
</xsd:complexType>
```

Derived Type

```
<xsd:complexType name=”NewAddress”>
    <xsd:extension base=”BaseAddress”>
        <xsd:sequence>
            <xsd:element name=”City” type=”xsd:string”/>
        </xsd:sequence>
    </xsd:extension>
</xsd:complexType>
```

7.3 Naming Conventions

Both complex and simple types should be defined using upper camel case.

Example:

```
<xsd:simpleType name="Wife">
   <xsd:restriction base="xsd:string">
      <xsd:minLength value="1"/>
   </xsd:restriction>
</xsd:simpleType>
```

8. Extensibility Points

MedBiquitous schemas may include an extensibility points to allow members to address situations where the standard format is insufficient to meet some special need. This is done using the XML Schema element xsd:any, that allows any element to be included. XML Schema design language provides up to five extension mechanisms. Wildcard extensions are the only approach where an extended instance document is valid against the original schema definition, and a core reason why this approach is recommended.

Working groups may choose whether to include a single point of extensibility or to allow extensibility at multiple points in the schema. Where to put extensibility points is highly dependent on the domain being modeled by the schema and the final decisions will have to be made by the experts in the working group. A few simple guidelines should be followed when making these decisions.

1. Schemas or sections of schemas that define well-established or fundamental content need not contain extensibility points.
2. Schemas or sections of schemas that attempt to codify a new or dynamic area of content should use extensibility points throughout.
3. If a schema is generally modeling stable, well-understood content, but some flexibility is desired for unforeseen cases, a single extensibility point can be defined.
4. Only elements from a namespace different from the document namespace should be allowed in the extension. This restriction is specified in XML Schema as:

   ```
   <xsd:any namespace="##other"/>
   ```

   A namespace constraint set to ##other avoids content collision and non-deterministic content models.

MedBiquitous also defines guidelines concerning how extension points should be used.

1. A extension should not repackage or subset existing information in the XML document. The extensions should be additional information added to the content model of the element being extended.
2. It is appropriate to require that the extensions be understood by a particular partner with which the extended instance documents will be exchanged. However, if the document is sent to a general MedBiquitous member with which no special agreements are in place (one that has no knowledge of the extension) the receiver must not be expected to process the extensions.

   MedBiquitous encourages members that extend the formal schemas to engage in review of such extensions within the MedBiquitous working groups after the extensions have been exercised in production for potential inclusion into the formal specifications if the workgroup sees fit to do so.

9. Other

The following guidelines are a collection of miscellaneous items that do not cleanly fit into any specific category.

9.1 Annotations

Annotations are a mechanism for documenting a schema definition file. A standard best practice is to document the complex types defined in the XML Schema definition file. Standard XML style comments should be avoided.

Example:

```
<xsd:simpleType name="Husband">
   <xsd:annotation>
      <xsd:documentation xml:lang=”en”>
      This is a complex type that defines the name of the husband
      </xsd:documentation>
   </xsd:annotation>
   <xsd:restriction base="xsd:string">
      <xsd:minLength value="1"/>
   </xsd:restriction>
</xsd:simpleType>
```

9.2 Default and Fixed Values

The main issue with default and fixed values is the dependency on the validation of the schema document. In other words, a schema with default or fixed values is incomplete if the schema instance is not validated against the schema definition file. Because it is not safe to depend on validation since there is no control over whether the consumer of the document will actually perform validation, default and fixed values should be avoided.

9.3 Substitution Groups and Choice

Since the Venetian Blind design pattern is recommended, the ability to utilize substitution groups is eliminated since substitution groups minimally depend on all elements (candidates for the substitution) be defined in the global namespace. In place of substitution groups, the choice option should be used. While this sacrifices extensibility to a certain extent, the gains in simplicity and maintainability offset this benefit. There is also a lack of support for substitution groups in the current JAXB specifications.[5]

9.4 any and anyAttribute

The any element and anyAttribute attribute are provided for schema extensibility. Rather than support schema extensibility through this approach, it is recommended that new schema versions be created as necessary and that old interfaces (versions) be supported for a period of time to prevent the need for a lock-step coordination with clients and services.

9.5 minOccurs and maxOccurs

The default value for both of these attributes is 1. Do not pollute the XML Schema definition file with these attributes if the default values are to be used.

[5] JAXB specifies a partial support of XML Schema making redefinitions, type substitutions, substitution groups, and derivation control attributes optional for JAXB implementations.