



Appendix A: XML Schema Best Practices

Table of Contents

1.1	OVERVIEW	2
1.2	DEFAULT NAMESPACE	2
1.2.1	<i>Best Practices Description</i>	2
1.2.2	<i>Best Practices Example</i>	2
1.2.3	<i>College Transcript Analysis and Recommendation</i>	4
1.3	COMPONENT DECLARATION: ELEMENT VS TYPE	4
1.3.1	<i>Best Practices Description</i>	4
1.3.2	<i>Best Practices Example</i>	4
1.3.3	<i>College Transcript Analysis and Recommendation</i>	4
1.4	NAMESPACE EXPOSURE	5
1.4.1	<i>Best Practices Description</i>	5
1.4.2	<i>Best Practices Example</i>	5
1.4.3	<i>College Transcript Analysis and Recommendation</i>	6
1.5	ACCESSIBILITY: GLOBALIZATION VS LOCALIZATION	6
1.5.1	<i>Best Practices Description</i>	6
1.5.2	<i>Best Practices Example</i>	7
1.5.3	<i>College Transcript Analysis and Recommendation</i>	9
1.6	NAMESPACE DESIGN	10
1.6.1	<i>Best Practices Description</i>	10
1.6.2	<i>Best Practices Example</i>	10
1.6.3	<i>College Transcript Analysis and Recommendation</i>	10
1.7	EXTENSIBLE CONTENT MODEL	11
1.7.1	<i>Best Practices Description</i>	11
1.7.2	<i>Best Practices Example</i>	11
1.7.3	<i>College Transcript Analysis and Recommendation</i>	11
1.8	VARIABLE CONTENT MODEL	12
1.8.1	<i>Best Practices Description</i>	12
1.8.2	<i>Best Practices Example</i>	12
1.8.3	<i>College Transcript Analysis and Recommendation</i>	15
1.9	REFERENCES	15



College Transcript Schema Analysis and Recommendations based on XML Schema Best Practices

1.1 Overview

This document is intended to provide a guideline for how the College Transcript can be modified in order to make it more extensible, dynamic and reusable. The College Transcript schema has been evaluated against several XML Schema Design Best Practices guidelines, particularly against those found on the MITRE Corporation's Schema Best Practices web site, xFront.com. The Best Practices areas are:

- Default Namespace
- Component Declaration
- Namespace Exposure
- Accessibility
- Namespace Design
- Extensible Content Model
- Variable Content Model

1.2 Default Namespace

1.2.1 Best Practices Description

XML namespaces provide a method for qualifying element and attribute names used in an XML document by associating them with namespaces defined by URI references. The `targetNamespace` defines the URI that all the components defined in the XML schema document will be associated with.

The default namespace specifies the namespace for which the schema components do not have to be namespace qualified. The best practices guidelines suggest that the `targetNamespace` should always be made the default namespace. The other options are making `XMLSchema` the default namespace or having no default namespace.

1.2.2 Best Practices Example

Approach 1 - Make the `targetNamespace` the default namespace.

This approach guarantees that components from any namespace other than the `targetNamespace` are always namespace qualified. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace=http://schemas.pescxml.org/0002/xsd/Core
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.pescxml.org/0002/xsd/Core">
  <xs:complexType name="BorrowerType">
    <xs:sequence>
      <xs:element name="Person" type="PersonType"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```



```
        <xs:element name="Employment" type="EmploymentType"/>
    </xs:sequence>
</xs:complexType>
</xs:schema>
```

PersonType and EmploymentType are not namespace qualified since they are part of the default namespace, in this case <http://schemas.pescxml.org/0002/xsd/Core>.

Approach 2 -Make XMLSchema the default namespace.

This approach ensures that components from any namespace other than XMLSchema are namespace qualified. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace=http://schemas.pescxml.org/0002/xsd/Core
xmlns:core="http://schemas.pescxml.org/0002/xsd/Core"
xmlns="http://www.w3.org/2001/XMLSchema">
    <complexType name="BorrowerType">
        <sequence>
            <element name="Person" type="core:PersonType"/>
            <element name="Employment" type="core:EmploymentType"/>
        </sequence>
    </complexType>
</schema>
```

In this example none of the XMLSchema components (element, sequence etc) need to be qualified since they belong to the default namespace. However, PersonType and EmploymentType need to be qualified in order for the processor to understand which namespace they belong to.

Approach 3 - No default namespace.

By following this approach all components in the schema must be namespace qualified. This provides a consistent way to represent all the components. For example:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace=http://schemas.pescxml.org/0002/xsd/Core
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:core="http://schemas.pescxml.org/0002/xsd/Core">
    <xs:complexType name="BorrowerType">
        <xs:sequence>
            <xs:element name="Person" type="core:PersonType"/>
            <xs:element name="Employment" type="core:EmploymentType"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

In this example there is no default namespace and all components are namespace qualified.



1.2.3 College Transcript Analysis and Recommendation

In the College Transcript schema there is no default namespace. This provides a consistent way of referring to components within the schema whereby they are all namespace-qualified. The only disadvantage to this is that the schema is cluttered and may be difficult to read. Having no default namespace is the best approach for the CommonLine schemas.

1.3 Component Declaration: Element Vs Type

1.3.1 Best Practices Description

Components within an XML schema can be declared as elements or types. Declaring components as types promotes reuse as the same type can be used to define multiple elements. The best practices guidelines suggest that whenever possible components should be declared as Types rather than Elements in order to promote reuse. The only scenario where a component must be declared as an element is if the item is substitutable for synonyms or aliases as in the case of a substitutionGroup.

1.3.2 Best Practices Example

Approach 1 - Declare component as an element

```
<xs:element name="Borrower">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Person" type="core:PersonType"/>
      <xs:element name="Employment" type="core:EmploymentType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In the example above, the Borrower element cannot be referenced by any other items.

Approach 2 - Declare components as a type

```
<xs:complexType name="BorrowerType">
  <xs:sequence>
    <xs:element name="Person" type="core:PersonType"/>
    <xs:element name="Employment" type="core:EmploymentType"/>
  </xs:sequence>
</xs:complexType>
```

In the above example, the type BorrowerType can be referenced by other items to define elements of this type.

1.3.3 College Transcript Analysis and Recommendation

The CommonLine schema has two substitutionGroups in CoreMain.xsd - LoanInformation and FinancialAward. These two components are defined as elements. All other components in the CommonLine Schema are declared as Types in accordance with the guidelines.



1.4 Namespace Exposure

1.4.1 Best Practices Description

XML Schemas have an attribute called `elementFormDefault` that controls whether component namespaces should be visible in the instance XML documents. This attribute can be set to “unqualified” whereby namespace complexities are localized to the schema and not propagated to the instance documents. Setting the attribute to “qualified” makes the namespaces of the elements visible in an instance document. The `elementFormDefault` switch can only control the exposure of local elements. All global elements must be namespace qualified. Therefore, schema designers should try to keep the number of global elements to a minimum.

The best practices guidelines suggest that there should be two copies of each XML Schema, one in which the `elementFormDefault` attribute is set to “qualified” and one in which it is set to “unqualified”. This allows the XML instance document writer to decide whether they want to expose namespaces and choose the corresponding schema to base their document on.

1.4.2 Best Practices Example

Approach 1 - elementFormDefault set to “qualified”

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace=http://schemas.pescxml.org/0002/xsd/Core
xmlns:core="http://schemas.pescxml.org/0002/xsd/Core"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="Borrower">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Person" type="core:PersonType"/>
        <xs:element name="Employment"
          type="core:EmploymentType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

A snippet of an XML instance document based on the above schema definition would look like this:

```
<core:Borrower>
  <core:Person>
    .....
  </core:Person>
  <core:Employment>
    .....
  </core:Employment>
</core:Borrower>
```



Approach 2: elementFormDefault set to “unqualified”

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace=http://schemas.pescxml.org/0002/xsd/Core
xmlns:core="http://schemas.pescxml.org/0002/xsd/Core"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified">
  <xs:element name="Borrower">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Person" type="core:PersonType"/>
        <xs:element name="Employment"
          type="core:EmploymentType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

A snippet of an XML instance document based on the above schema definition would look like this:

```
<core:Borrower>
  <Person>
    .....
  </Person>
  <Employment>
    .....
  </Employment>
</core:Borrower>
```

In the above example Borrower still has to be namespace qualified since it is a global element.

1.4.3 College Transcript Analysis and Recommendation

In the CommonLine schema the elementFormDefault attribute is set to “unqualified”. This allows the instance document to be more readable. It would be advisable to have a second copy of the schemas with the elementFormDefault set to “qualified”. This way the developer of the instance XML document has a choice as to whether the document will be namespace qualified and he/she can choose appropriate schema definition.

1.5 *Accessibility: Globalization Vs Localization*

1.5.1 Best Practices Description

There are three main methods of controlling the accessibility of elements within an XML schema document. These are – the Russian Doll Design, the Salami Slice Design and the Venetian Blind Design. These methods will be demonstrated through the following examples.



The best practices guidelines suggest the Venetian Blind Design as the best way to format schemas.

1.5.2 Best Practices Example

Approach 1 - The Russian Doll Design

In this design format, components are nested within other components. It is demonstrated by the following example:

```
<xs:element name="Borrower">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Person">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Birth">
              .....
            </xs:element>
            <xs:element name="Name">
              .....
            </xs:element>
            .....
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Employment">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Employer">
              .....
            </xs:element>
            <xs:element name="PositionTitle">
              .....
            </xs:element>
            .....
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The Russian Doll Design allows components to be bundled into a single tidy unit. The negative impact is that all components are localized and cannot be reused.



Approach 2 - The Salami Slice Design

In this design format, all the components are divided into individual element declarations and then assembled together. It is demonstrated by the following example:

```
<xs:element name="Borrower">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="core:Person"/>
      <xs:element ref="core:Employment"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Person">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Birth"/>
      <xs:element ref="Name"/>
      .....
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Employment">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Employer"/>
      <xs:element ref="PositionTitle"/>
      .....
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Birth">
  .....
</xs:element>
<xs:element name="Name">
  .....
</xs:element>
<xs:element name="Employer">
  .....
</xs:element>
<xs:element name="PositionTitle">
  .....
</xs:element>
```

The Salami Slice Design represents all the components in a global scope. While this allows reusability, it also does not allow namespace hiding.



Approach 3 - The Venetian Blind Design

In this design format, the components are laid out as individual type definitions that are then used to form element definitions. It is demonstrated by the following example:

```
<xs:complexType name="BorrowerType">
  <xs:sequence>
    <xs:element name="Person" type="PersonType"/>
    <xs:element name="Employment" type="EmploymentType"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="PersonType">
  <xs:sequence>
    <xs:element name="Birth" type="BirthType"/>
    <xs:element name="Name" type="NameType"/>
    .....
  </xs:sequence>
</xs:complexType>
<xs:complexType name="EmploymentType">
  <xs:sequence>
    <xs:element name="Employer" type="EmployerType"/>
    <xs:element name="PositionTitle" type="PositionTitleType"/>
    .....
  </xs:sequence>
</xs:complexType>
<xs:complexType name="BirthType">
  .....
</xs:complexType>
<xs:complexType name="NameType">
  .....
</xs:complexType>

<xs:complexType name="EmployerType">
  .....
</xs:complexType>

<xs:complexType name="PositionTitleType">
  .....
</xs:complexType>
```

The main advantage of the Venetian Blind Design format is that components are globally defined and are therefore reusable. At the same time it is easy to control namespace exposure since elements are nested within type definitions.

1.5.3 College Transcript Analysis and Recommendation

The CommonLine schema is based on the Venetian Blind Design format. The only exceptions are LoanInformation and FinancialAward in CoreMain.xsd that are declared as global elements since they are part of a substitutionGroup. The removal of substitutionGroups will be discussed in detail at a later point.



1.6 Namespace Design

1.6.1 Best Practices Description

The best practices guidelines presents three namespace designs for determining the targetNamespace for multiple schemas – Heterogeneous, Homogeneous and Chameleon. There is no one best solution for this case, but there are guidelines for when one approach is better than the other. The best practices guidelines also suggest that irrespective of the namespace design, all schema components should have an “id” attribute in order to provide a finer level of granularity.

1.6.2 Best Practices Example

Approach 1 - Heterogeneous Namespace Design

In this namespace design method, each schema has a different targetNamespace. This approach is best when there are multiple elements with the same name or there is a need to visually identify the lineage of each element.

Approach 2 - Homogeneous Namespace Design

In this namespace design method, all the schemas have the same targetNamespace. This approach is best when all the schemas are conceptually related and there is no need to identify the lineage of each element.

Approach 3 - Chameleon Design

In this namespace design method, the “integrating schema” has a targetNamespace, and the “supporting schemas” have no targetNamespaces. The no namespace components of the “supporting schemas” take on the targetNamespace of the “integrating schema”. This approach is best when the components contained in the “supporting schemas” have no inherent semantics; rather it is only within the context of the “integrating schema” that they take on semantics.

1.6.3 College Transcript Analysis and Recommendation

The CommonLine Schema follows the Heterogeneous approach, whereby each schema has a different targetNamespace. This is in accordance with the guidelines since this helps to avoid name collisions and also provides a visual indication of the origin/lineage of each component. The main disadvantage of this approach is that components defined in one schema cannot be redefined in a different schema. The best practices guidelines also suggest that all schema components have an “id” attribute in order to provide a finer level of granularity. However, this is not necessary for the CommonLine Schema since this level of programmatic categorization is not required at the moment.



1.7 Extensible Content Model

1.7.1 Best Practices Description

An element has an extensible content model if in instance documents that element can contain elements and attributes above and beyond what was specified by the schema. The best practices guidelines suggest that content models be made extensible since it is impossible for the schema designer to anticipate the varieties of data that an instance document author might need to use in creating an instance document. This can be done through the use of an <any> tag.

1.7.2 Best Practices Example

This is the current definition for BorrowerType.

```
<xs:complexType name="BorrowerType">
  <xs:sequence>
    <xs:element name="Person" type="PersonType"/>
    <xs:element name="Employment" type="EmploymentType"/>
  </xs:sequence>
</xs:complexType>
```

This definition can be extended by using the <any> tag.

```
<xs:complexType name="BorrowerType">
  <xs:sequence>
    <xs:element name="Person" type="PersonType"/>
    <xs:element name="Employment" type="EmploymentType"/>
    <xs:any namespace="##other" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

The instance document designer is now able to extend BorrowerType to include any data he/she requires. The use of the "namespace=##other" signifies that any element taking the place of the <any> element must come from a namespace different from the current one. This ensures that there is no ambiguity in determining the path taken in the schema document.

1.7.3 College Transcript Analysis and Recommendation

The CommonLine Schema follows the suggestion of using the <any> tag to extend an element and does this for the UserDefinedExtenType in CoreMain.xsd to provide an extensible, deterministic schema model.



1.8 Variable Content Model

1.8.1 Best Practices Description

There are a number of ways to design an element that can be comprised of variable content. These include, Abstract element and element substitution, Abstract type and type substitution, and using a dangling type.

Using abstract element and element substitution has the following disadvantages:

1. **No Independent Elements** - The variable content container (in the case of the CommonLine schema, the substitutionGroups) cannot contain elements whose type does not derive from the abstract element's type.
2. **Structural variability** - There is limited structural variability for the types derived from the base type.
3. **Non-linear processing** - Each time a change is made to the variable container (i.e., an element is added or removed from the substitutionGroup), changes to need to be made to any processors (e.g., a stylesheet) handling the instance document.
4. **Namespace Exposure** - The elements in the substitution groups are declared globally. Thus, they must always be namespace qualified. This fails the guideline that namespace exposure should be controllable through elementFormDefault.

Abstract types and type substitution can be used to eliminate the third and fourth disadvantage. However, the problems of independent types and structural variability still remain. These problems can be eliminated by using the following "Dangling Type" method.

1.8.2 Best Practices Example

Approach 1 - Abstract element and element substitution

This approach can be demonstrated by the current definition of LoanInformation.

```
<xs:element name="LoanInformation" type="core:LoanInformationType"
abstract="true"/>
<xs:element name="DLLoanInformation" type="core:DLLoanInformationType"
substitutionGroup="core:LoanInformation"/>
<xs:element name="FFELLOanInformation" type="core:FFELLOanInformationType"
substitutionGroup="core:LoanInformation">
  <xs:annotation>
    <xs:documentation>NOT USED</xs:documentation>
  </xs:annotation>
</xs:element>
```

Approach 2 - Abstract type and type substitution

The definition of LoanInformationType itself can be changed whereby it is made abstract.

```
<xs:complexType name="LoanInformationType" abstract="true">
  <xs:sequence>
```



```
<xs:element name="StudentLevelCode"
type="core:StudentLevelCodeType" minOccurs="0"/>
<xs:element name="FinancialAwardBeginDate" type="xs:date"
nillable="true" minOccurs="0"/>
<xs:element name="FinancialAwardEndDate" type="xs:date"
nillable="true" minOccurs="0"/>
</xs:sequence>
<xs:attribute name="LoanKey" type="xs:integer" use="required"/>
</xs:complexType>
```

Any element that is of type LoanInformationType can now have the contents of any of the types derived from LoanInformationType.

Through this approach of using abstract types rather than abstract elements, the following problems are eliminated:

1. **Nonlinear processing** - With this method it is not the container that contains the variable content but the element within the container. Thus, the contents can always be processed uniformly irrespective of the changes made to the content.
2. **Namespace Exposure** - By eliminating the substitutionGroups, the global element declarations are also eliminated. Thus, namespace exposure can be controlled through elementFormDefault.

Approach 3 - Dangling Types

The following changes can be made to LoanInformation in order to implement Dangling Types.

- 1) Move the declaration of LoanInformationType to a new schema called LoanInformationSchema.xsd.

```
<xs:element name="LoanInformation" type="core:LoanInformationType"/>
<xs:element name="DLLoanInformation" type="core:DLLoanInformationType"
substitutionGroup="core:LoanInformation"/>
<xs:element name="FFELLoanInformation" type="core:FFELLoanInformationType"
substitutionGroup="core:LoanInformation">
  <xs:annotation>
    <xs:documentation>NOT USED</xs:documentation>
  </xs:annotation>
</xs:element>
```

- 2) Move the DLLoanInformationType declaration to a new schema called DLLoanInformationSchema.xsd.

```
<xs:complexType name="DLLoanInformationType">
  <xs:complexContent>
    <xs:extension base="core:LoanInformationType">
      <xs:sequence>
        <xs:element name="OriginationFeePercent" nillable="true"
minOccurs="0">
          <xs:simpleType>
```



```
        <xs:restriction base="xs:decimal">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="100"/>
            <xs:totalDigits value="6"/>
            <xs:fractionDigits value="3"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
.....
<xs:element name="AcademicYearBeginDate" type="xs:date"
nillable="true" minOccurs="0"/>
<xs:element name="AcademicYearEndDate" type="xs:date"
nillable="true" minOccurs="0"/>
<xs:element name="Response" type="core:ResponseType"
minOccurs="0"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

3) Move FFELLoanInformationType to a new schema called FFELLoanInformation.xsd.

```
<xs:complexType name="FFELLoanInformationType">
    <xs:annotation>
        <xs:documentation>NOT USED other than by
        FFELLoanInformation above</xs:documentation>
    </xs:annotation>
    <xs:complexContent>
        <xs:extension base="core:LoanInformationType">
            <xs:sequence>
                <xs:element name="SignatureData"
                type="core:SignatureDataType" minOccurs="0"/>
                .....
                <xs:element name="Response" type="core:ResponseType"
                minOccurs="0"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Any schema that needs to have an element of type LoanInformationType must import the namespace associated with LoanInformationType. But the schema should not specify the schema location. For example, the Schema declaration could look like this:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://schemas.pescxml.org/0002/csd/Core"
xmlns:core="http://schemas.pescxml.org/0002/csd/Core"
xmlns:loan="http://schemas.pescxml.org/0002/csd/Loan">
```

Any element that is declared to be of type LoanInformationType should represent LoanInformationType as being part of a different namespace. For example, the declaration could look like this:



```
<xsd:element name="LoanInformation" type="loan:LoanInformationType"/>
```

The instance document must then identify a schema that implements LoanInformationType. Thus, at runtime the reference to LoanInformationType can be matched up with the implementation of LoanInformationType, which in this case is contained in DLLoanInformationType.xsd.

For example, the instance document could specify the following schema location:

```
xsi:schemaLocation="http://schemas.pescxml.org/0002/csd/Core CoreMain.xsd  
http://schemas.pescxml.org/0002/csd/Loan/DLLoanInformation.xsd"
```

1.8.3 College Transcript Analysis and Recommendation

In the CommonLine schema LoanInformation and FinancialAward in CoreMain.xsd can have variable content. The CommonLine Schema uses abstract elements and a substitutionGroups in order to achieve this. The LoanInformation definition can be changed to implement either Approach 2 or Approach 3, as demonstrated in the examples above, in order to remove the disadvantages that are cause by Approach 1. Similar changes can be made to the FinancialAwardType to allow variable content.

1.9 References

Roger Costello, The MITRE Corporation, August 22 2003,
<<http://www.xfront.com/BestPracticesHomepage.html>>

Paul Golick, Association for Retail Technology Standards, August 25 2003, <<http://www.nrf-arts.org/bestpractices.htm>>