OTA

The OpenTravel™ Alliance

OpenTravel™ Alliance
XML Schema Design Best Practices

Version 3.0
5 December 2003

1 OTA XML Schema Design Best Practices

The IT Business world has long employed the principles of producing high quality products with a reduction of product development cost and faster "time-to-market" product delivery. In today's global, Internet-ready marketplace, these principles are as critical to the bottom line as ever. One way that corporations can apply these "increased earning potential principles" is by establishing a common set of best practice XML and XML Schema guidelines.

The current W3C XML specifications were created to satisfy a very wide range of diverse applications, which is why there may be no single set of "good" guidelines on how best to apply XML technology. However, when the application environment can be restricted by corporate direction or by a common domain, one can determine, by well-informed consensus, a set of effective guidelines that will lead to the best practice of using XML in that environment.

This document defines the OpenTravel™ Alliance (OTA) Best Practices Guidelines for all of the OTA XML data assets. OTA message specifications released prior to the 2002A Specification release may not follow the guidelines defined in this document.

2 XML Standard Specifications

Currently, there are several XML related specification recommendations produced by W3C (http://www.w3.org/Consortium/). This section refers to the W3C recommendations (http://www.w3.org/Consortium/Process-20010719/) and versions listed below:

- Extensible Markup Language (XML) 1.0 (Second Edition):
  - http://www.w3.org/TR/2000/REC-xml-20001006

- XML Schema Parts 0 - 2:
  - http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/
  -
  - http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/
  - http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

3 Best Practices

3.1 Scope

The OTA Best Practices Guidelines cover all of the OTA XML components (elements, attributes, tag names, and Schema definitions). This document defines guidelines for all OTA XML data assets.

The general OTA guideline approach is to maximize component (element/attribute) reuse for the highly diverse and yet closely related travel industry data. This is accomplished by building messages via context-driven component assembly. An example is the construction of a 'Flight Leg' segment from base objects such as 'Time,' 'Date,' and 'Location' (departure/arrival). The best mechanism that XML Schemas have to support this approach is by encapsulating lower level components (element and attribute objects) within named type definitions while using (and reusing) these base components to construct messages.

3.2 Schema Design Component Parts and Roles

The critical XML Schema guidelines that best support the OTA goal of a consistent set of reusable travel industry message content are listed below:

- Tag Naming conventions
- Root Element, Message, and File Naming Conventions
- Elements and Attributes
- Use of XML Schema
- Global vs. Local Element Types
- Namespaces
- Versioning XML Schemas
- Schema Markup and Annotations
- Enumerations vs. Code Lists
- Code Lists
- General

Each of the items above plays a unique role, supporting a common vocabulary, syntax, and semantic grammar for XML Schema and XML component (element and attribute) definitions.

4 OTA XML Schema Design Guidelines

The subsections below form the complete set of OTA XML Schema Design Best Practices Guidelines. Each guideline is presented as follows:

***Guideline*:** The base rule (or rules) that should be followed for compliance with OTA Best Practices.

***Rationale*:** OTA general consensus reasoning for the guideline.

Example:

```
An example (if applicable).
```

4.1 Tag Naming Conventions

4.1.1 Mixed Case

Guideline: Use mixed case tag names, with the leading character of each word in upper case and the remainder in lower case without the use of hyphens or spaces between words (a.k.a. Upper Camel Case (UCC) or "PascalCasing").

Rationale: This format increases readability and is consistent with common industry practices.

Example:

```
<FlightNumber>     <HotelCode>
```

4.1.2 Underscore

Guideline: Where the merger of tag name words and acronyms causes two upper case characters to be adjacent, separate them with an underscore ('_').

Rationale: This technique eliminates or reduces any uncertainty for tag name meaning.

Example:

```
<PO_Box>     <ID_Context>
```

4.1.3 Acronyms

Guideline: Acronyms are discouraged, but where needed, use all upper case.

Rationale: In some cases, common acronyms inhibit readability. This is especially true for internationally-targeted audiences. However, in practice, business requirements and/or physical limitations may require the need to use acronyms.

Example:

```
<AreaID>     <PassengerRPH>
```

4.1.4 Word Abbreviations

Guideline: Word abbreviations are discouraged. However, where needed, use UCC camel case.

Rationale: Abbreviations may inhibit readability. This is especially true for internationally-targeted audiences. However, in practice, business requirements and/or physical limitations may require the need to use abbreviations.

Example:


```
<FormattedInd>      <AcctType>
```

4.1.5 Tag Length

Guideline: Element and attribute names should not exceed 25 characters. Tag names should be spelled out except where they exceed 25 characters, when standardized abbreviations should be applied.

Rationale: This approach can reduce the overall size of a message significantly and limit impact to any bandwidth constraints.

Example:

```
The tag:  <ShareSynchronizationIndicator> can be reduced to:  <ShareSyncInd>
```

4.1.6 Complex Type Tag Names

Guideline: Complex type tag names should be suffixed with the word "Type"

Rationale: This approach allows for complex types to be easily recognized, which encourages reuse.

Example:

```
<CurrencyAmountType>  <ParagraphType>
```

4.1.7 Simple Type Tag Names 1

Guideline: OTA data type simpleType tag names should clearly indicate the pattern that is used to define the simple type.

Rationale: This approach supports meaningful tag names.

Example:

```
<Numeric0to4>
```

4.1.8 Simple Type Tag Names 2

Guideline: All other OTA simpleType tag names should clearly indicate the usage of that type and should be suffixed with the word "Type".

Rationale: This approach supports meaningful tag names.

Example:

```
<RPH_Type>
```

4.1.9 Naming of Elements Based on Simple or Complex Types

Guideline: Elements that are based on complex or simple types must not be suffixed by "ComplexType," "SimpleType," or "Type."

Rationale: This technique reserves the "Type" suffix for complex and simple types, which allows for easy identification and reuse of types.

Example:

```
<Profiles> of type ProfilesType <RequestorID> of type UniqueID_Type
```

4.1.10 Naming of Attributes Based on Simple Types

Guideline: Attributes that are based on simple types must not be suffixed by "SimpleType" or "Type"

Rationale: This technique reserves the "Type" suffix for complex and simple types, which allows for easy identification and reuse of types.

Example:

```
<ID>of type StringLength1to32
<AirportCode> of type UpperCaseAlphaNumericLength3to5
```

4.1.11 Common Suffixes

Guideline: Use common tag name suffixes for elements defined by similar or common XML Schema type definitions.

Rationale: This approach supports a consistent syntax and semantic meaning for elements and attributes.

Example:

```
<OriginLocation> <DestinationLocation>  <ConnectionLocation>
```

4.1.12 Standard Suffixes

Guideline: The OTA XML Schema type definitions (includes simpleTypes, complexTypes, attributeGroups and groups) should incorporate the following list of suffixes for naming type labels. However, if a user-defined 'simpleType' definition is identical to a built-in XML Schema type, the built-in type definition should be used.

Primary *Representation Term*	**Definition**	**Secondary *Representation Term(s)***
Amount	A number of monetary units specified in a currency where the unit of currency is explicit or implied	
Binary Object	A set of finite-length sequences of binary octets. [Note: This *Representation Term* shall also be used for *Data Types* representing graphics (i.e., diagram, graph, mathematical curves, or similar representation), pictures (visual representation of a person, object, or scene), sound, video, etc.]	Graphic, Picture, Sound, Video
Code	A character string (letters, figures, or symbols) that for brevity and / or language independence may be used to represent or replace a definitive value or text of a *Property*. [Note: The term 'Code' should not be used if the character string identifies an instance of an *Object Class* or an object in the real world, in which case the *Representation Term* identifier should be used.]	
Date Time	A particular point in the progression of time (ISO 8601). [Note: This *Representation Term* shall also be used for *Data Types* only representing a Date or a Time.]. Examples: (CCYY-MM-DD); (hh:mm:ss[.ssss…[Z | +/-hh:mm]]); (CCYY-MM-DD [**T**hh:mm:ss [.ssss…[Z | +/-hh:mm]]])	Date, Time

Primary *Representation Term*	**Definition**	**Secondary *Representation Term(s)***
Identifier	A character string used to establish the identity of, and distinguish uniquely, one instance of an object within an identification scheme from all other objects within the same scheme.	
Indicator	A list of exactly two mutually exclusive Boolean values that expresses the only possible states of a *Property*. [Note: Indicated by a Boolean data type.]	
Measure	A numeric value determined by measuring an object. Measures are specified with a unit of measure. The applicable unit of measure is taken from UN/ECE Rec. 20. [Note: This *Representation Term* shall also be used for measured coefficients (e.g., m/s).]	
Numeric	Numeric information that is assigned or is determined by calculation, counting, or sequencing. It does not require a unit of quantity or a unit of measure. [Note: This *Representation Term* shall also be used for *Data Types* representing Ratios (rates where the two units are not included or where they are the same), Percentages, etc.]	Value, Rate, Percent
Quantity	A counted number of non-monetary units. Quantities need to be specified with a unit of quantity. [Note: This *Representation Term* shall also be used for counted coefficients (e.g., flowers/m²).]	
Text	A character string (i.e., a finite set of characters) generally in the form of words of a language. [Note: This *Representation Term* shall also be used for names (i.e., word or phrase that constitutes the distinctive designation of a person, place, thing, or concept).]	Name

Rationale: This approach supports a consistent syntax and semantic meaning for XML Schema definitions and does not affect the naming of element and attribute tags in an instance document.

4.2 Root Element, Message, and File Naming Conventions

4.2.1 Root Element Naming

Guideline: The format of root elements for messages shall be "OTA_" + Vertical name or area of focus + function + RQ or RS.

Rationale: This format allows for easy identification of message, Vertical, and function.

Example:

```
<OTA_HotelAvailRQ>     <OTA_InsuranceBookRS>
```

4.2.2 Use of Notif in Root Element Name

Guideline: The word "Notif" in a message name indicates that this message does not follow the normal requirements of a Request/Response transaction. This type of message provides (pushes) information from the originator to the recipient in support of a trading partner agreement.

Rationale: This technique allows for quick and easy identification of push messages.

Example:

```
<OTA_HotelResNotifRQ>   <OTA_HotelResNotifRS>
```

4.2.3 Message XML Schema File Naming

Guideline: The .xsd file is given the same name as the root element of the XML Schema.

Rationale: Easily identifies the contents of the .xsd file.

Example:

```
Root element: <OTA_AirFlifoRQ>
File name: OTA_AirFlifoRQ.xsd
```

4.2.4 File naming for collections of Attribute Groups, Simple, and Complex Types

Guideline: CommonType and SimpleType XML Schema files are used to house attribute groups, simple types, and complex types that are used among multiple messages. Items that apply to a specific Vertical are housed in a common file that includes the Vertical name.

Rationale: This approach easily identifies reusable components.

Example:

```
<OTA_SimpleTypes>     <OTA_CommonTypes>     <OTA_AirCommonTypes>
```

4.2.5 Naming of XML Schema Files that Contain Common Components

Guideline: Schema files that are not used as messages by themselves, but contain components for use in messages, should not contain RQ or RS in the Schema name. These files are primarily used for maintaining consistency between common message structures, usually in an RQ/RS set and its Notif counterparts.

Rationale: This approach allows for easy differentiation between messages and message components.

Example:

```
<OTA_Profile>   <OTA_HotelReservation>
```

4.3 Use of Elements and Attributes

4.3.1 Elements vs. Attributes

Guideline: For a given OTA data element, the preferred method is to represent that data-element as an attribute. The data-element is represented as an element if and only if:

- it is not atomic (i.e., it has attributes or child elements of its own) OR

- the anticipated length of the attribute value is greater than 64 characters[1] OR
- the presence or absence of the attribute represents a semantic 'choice' or branch within the Schema OR
- it is likely that the data element in question will be extended in the future

Rationale: The intention is to create a consistent OTA message design approach and to reduce the overall message size as well as avoid the potential of tag name collisions.

Example:

```
Element:
<LocationDescription>Five miles South of highway 85 and Main St. intersection next to
Town Square Mall</LocationDescription>

Attribute:
<ArrivalAirport LocationCode=”MIA” />
```

4.3.2 Number of Attributes per Element

Guideline: Element tags should not be overloaded with too many attributes (no more than 10 as a rule of thumb); instead, encapsulate attributes within child elements that are more closely related (or more granular). This should be done for those attributes that are likely to be extended by OTA or by specific trading partners.

Rationale: This approach maintains the built-in extensibility that XML provides with elements and is necessary to provide forward compatibility as the specification evolves. It also provides a consistent guide to the level of granularity used to compose OTA Schema objects (or fragments).

4.3.3 Encapsulating Element

Guideline: XML element containers must be used for repeating elements if the XML Schema 'maxOcc' attribute exceeds 5 repetitions. The encapsulating element container is optional if the XML Schema 'maxOcc' attribute is less-than or equal to 5. However, a single XML <element> container can be used for "simpleType" repeating content (via the XML Schema "list" construct).

Rationale: This technique provides consistency for repeating data fields.

Example:

```
    complexTypes -

      <States>
        <State country="US">NY</State>
        <State country="US">FL</State>
        <State country="US">CA</State>
      </States>

    simpleTypes -

        <States>NY FL CA</States>
or
        <Location RegionStates = "NY FL CA"/>
```

4.4 Use of XML Schema

4.4.1 OTA Specification Uses XML Schema

[1] URLs are considered to be less than 64 characters.

Guideline: The XML Schema recommendations from W3C should be used to define all XML message documents.

Rationale:

- Schemas are written in XML syntax, rather than complex SGML regular expression syntax.
- Because XML Schemas are themselves well-formed XML documents, they can be programmatically generated and validated using a meta-schema – a Schema used to define other Schema models.
- XML Schemas have built-in data types and an extensible data-typing mechanism. (DTDs understand only markup and character data.)
- Using XML syntax to define data model requirements allows for more constraints, strong data typing, etc.
- XML Schemas provide for a consistent Data Repository syntax.

4.5 Global vs. Local Element Types

4.5.1 Simple and Complex Types

Guideline: Define XML Schema element types globally in the namespace for the elements that are likely to be reused (instead of defining the type anonymously in the Element declaration). This applies to both simpleType and complexType element type definitions.

Rationale: This approach supports a domain library or repository of reusable XML Schema components. Also, since simpleType and complexType names are not contained in XML instance documents, they can be verbose to avoid element type name collisions.

4.5.2 Type Attribute vs. Ref Attribute

Guideline: Define XML Schema elements as nested elements via the 'type' attribute or an inline type definition ('simpleType' or 'complexType') instead of the 'ref' attribute that references a global element.

Rationale: This approach for local element naming reduces the possibility of tag name collisions and allows the creation of short tag names. Globally-defined elements should be reserved only for travel domain elements with well-defined meanings; such global names should be constructed with sufficient roots and modifiers to identify their domain of use and avoid tag-name collisions.

Example:

```
<xs:complexType name="AddressType">
 <xs:sequence>
   <xs:element name="StreetNmbr" type=" xs:string" minOccurs="0"/>
   <xs:element name="BldgRoom" type="PlaceID_Type"
               minOccurs="0"maxOccurs="unbounded"/>
   <xs:element name="AddressLine" type="AddressLineType"
               minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="CityName" minOccurs="0">
     <xs:complexType>
       <xs:simpleContent>
         <xs:extension base="xs:string">
           <xs:attribute name="PostalCode" type=" PostalCodeType"/>
         </xs:extension>
       </xs:simpleContent>
     </xs:complexType>
   </xs:element>
```



```
   <xs:element name="StateProv" type=”StateProvinceType” minOccurs="0"/>
   <xs:element name="CountryName" name="CountryNameType” minOccurs="0"/>
   <xs:element name="PrivacyDetails" type="Privacy"/>
 </xs:sequence>
</xs:complexType>
```

4.5.3 Attribute Groups

Guideline: Define common attribute parameters globally as a reusable component via the XML Schema ‘attributeGroup’ element definition.

Rationale: This approach supports a domain library or repository of reusable XML Schema components. Also, since the names used for the XML Schema ‘attributeGroup’ components are not contained in XML instance documents, they can be verbose to avoid name collisions with other ‘attributeGroup’ definitions.

Example:

```
<xs:attributeGroup name="OTA_PayloadStdAttributes">
  <xs:attribute name="EchoToken" type="OTA_TokenType"/>
  <xs:attribute name="TimeStamp" type="xs:dateTime"/>
  <xs:attribute name="Target" default="Production">
    <xs:simpleType>
      <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="Test"/>
        <xs:enumeration value="Production"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="Version" type="OTA_VersionType"
                       use=”required”/>
  <xs:attribute name="SequenceNmbr" type="xs:integer"/>
</xs:attributeGroup>
```

4.6 Namespaces

4.6.1 OTA Namespace

Guideline: All OTA message Schemas are declared in one targetNamespace, which is http://www.opentravel.org/OTA/2003/05. However, during the specification review period, the domain name will include an extension of alpha or beta corresponding to member review and public review respectively. If additional releases are necessary, they would continue with gamma, delta, etc.

Starting with release 2003A, the year and month on this targetNamespace is set to the initial publication of the 2003A OTA specification (the baseline specification). This value will not be changed in the subsequent releases, and the same namespace will also be used for new messages. The only reason to change the namespace would be to deprecate the 2003 baseline specification. This value would change to support the new OTA baseline specification, an action which should occur only on a 3- or 4-year cycle.

Rationale: This approach supports a consistent way to manage and identify OTA XML-based transaction assets both internally and externally (via trading partners and global e-business repositories such as UDDI). It also avoids the need for explicit prefixes on both XML Schema and XML instance documents.

Example:

```
      http://www.opentravel.org/OTA/2003/05
  or
      http://www.opentravel.org/OTA/2003/05/alpha
```



```
Usage:

    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://www.opentravel.org/OTA/2003/05"
       xmlns="http://www.opentravel.org/OTA/2003/05"
       version="1.0"
        id="OTA2003A">
```

4.6.2 No Namespace for Common XML Schema Files

Guideline: There will be no namespace for any common OTA data type .xsd Schema file.

Rationale: Common data type Schema files (i.e., type definitions only) are version independent from message Schemas that may include them, and this content may be applied to multiple versions of a message.

Example:

```
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        version="4.123"
        id="OTA2003A2003B">
```

4.6.3 Use of OTA Namespace in Instance Documents

Guideline: Each XML instance document produced by the 'OTA' namespace Schemas should specify a default namespace and that should be the 'OTA' namespace defined above. Also, a namespace prefix of "ota:" is to be reserved for the 'OTA' namespace and used where 'OTA' is required not to be a default namespace, to satisfy unique business needs.

Rationale: This approach provides a standard way for "OTA" namespace content to be merged with other Industry or Trading Partner namespace content.

4.7 Versioning XML Schemas

4.7.1 Version Attribute in XML Schema

Guideline: The <xs:schema> root element of XML Schemas will contain a 'version' attribute whose value will identify both the major version base and a minor version sequence value.

Rationale: This approach enables easy identification of the two basic ways that an XML Schema and conforming instance documents may change:

A) Extensions to a Schema via adding new content, which does not invalidate the previous version (i.e., minor version change).

B) Structural content or data type changes where the previous content would not validate against the new Schema (i.e., major version change).

Example:

```
A)  Multiple minor version messages of a particular base message Schema (or major
version) will all validate against the latest base Schema version (e.g., forward
compatibility: message versions '2.012', '2.037' and '2.050' all validate against Schema
version '2.050').

B) Version values for major changes '2.000', '3.000', '4.000',…
```


4.7.2 Version Attribute in Common XML Schema Files

Guideline: The 'version' attribute in the <xs:schema> root element of OTA common data type Schema files (e.g., OTA_CommonTypes.xsd) will contain an independent self-describing version value (e.g., version="19.127", where '19' is the major version and '127' is the minor version).

Rationale: Common data type Schema files (i.e., type definitions only) are version independent from message Schemas that may include them, and this content may be applied to multiple versions of a message.

4.7.3 Version Attribute in XML Instance Documents

Guideline: XML instance documents being validated against an OTA message Schema will contain a 'Version' attribute on the root element. The value of this attribute should map directly to the value of the 'Version' attribute on the root 'Schema' element of the message Schema being used for validation.

Rationale: This approach provides version correlation between XML instance message and the corresponding XML Schema.

Example:

```
Schema value:
       version="1.050"

matches instance value:
       Version="1.050"
```

4.7.4 ID Attribute in Message and Common XML Schema

Guideline: The 'id' attribute in the <xs:schema> root element of OTA XML Schemas will contain the release. The 'id' attribute in the <xs:schema> root element of OTA common data type XML Schemas will contain the range release. The OTA specification manager will update the 'id' attribute of all schemas to the current release prior to publishing the schemas. The 'id' attribute is only found in the XML Schema, it is not used in the instance.

Rationale: This attribute indicates the release in which the XML Schema was published. It is important to note that the 'id' attribute does not indicate if a message format has changed between releases, this is determined by comparing the 'version' attribute in the <xs:schema> root element of the XML Schemas.

Example:

```
Message schema files:
       id="OTA2003A"

CommonType schema files
       id="OTA2003A2003B"
```

4.7.5 Use of schemaLocation Attribute

Guideline: The attribute schemaLocation is to be used on elements in instances to name the location of a retrievable Schema for that element associated with that namespace.

Rationale: This approach supports use of OTA XML Schemas.

Example:

```
Attribute:
xsi:schemaLocation="http://www.opentravel.org/OTA http://www.opentravel.org/OTA/2002A-
REC/VEH- availability/VehAvailRateRQ-23.xsd"
```

4.8 Schema Markup and Annotations

4.8.1 Use of Annotation and Document Elements

Guideline: OTA XML Schemas will use the <documentation> sub-element of the <annotation> element for Schema documentation.

Rationale: Schema comments "<!--… -->" are not part of the core information set of a document and may not be available or in a useful form.

Example:

```
<xs:annotation>
   <xs:documentation>Privacy sharing control attributes.
   </xs:documentation>
</xs:annotation>
```

4.8.2 Use of lang Attribute

Guideline: Documentation elements will include the xs:lang attribute. The initial value of the attribute will be set to "en".

Rationale: This approach allows for future inclusion of documentation in other languages.

Example:

```
xs:lang="en"
```

4.8.3 Meaningful Annotations

Guideline: OTA requires that all complex types, simple types, elements, attribute groups, attributes, and enumerations are meaningfully annotated.

- Complex type annotation: Describe the overall purpose of a complex type.
- Simple type annotation: Define the structure and its usage.
- Element annotation: Must describe the element in a meaningful manner so that the trading parties, who may not always have full understanding of the business context of the messages they are implementing, can understand the usage of the element.
- Attribute group: At the attribute group declaration, describe the overall functionality of the grouping. Within the element where the attribute group is referenced, include a description of the specific use of the attribute group.
- Attributes: Must include usage information.
- Enumerations: Provide an explanation of each value.

Rationale: These standards enable the readers of a Schema to understand the usage of each data item.

4.8.4 Annotation of Typed Elements

Guideline: Annotation of elements that are typed should reflect the specific usage of that complex or simple type at that location. If there is no additional specific usage information, then the global annotation found at the complex or simple type must be duplicated at the element level.

Rationale: This approach enables the readers of a Schema to understand the usage of a typed element in its specific context.

4.8.5 Annotations of Root Elements

Guideline: The root element of each RQ message shall include an overall description of the functionality of the message pair. If an RS message (e.g., OTA_ErrorRS) does not have a companion RQ message, then the full description of the message is to be included in the RS.

Rationale: This approach enables the readers of a Schema to understand the functionality of a message.

4.8.6 Use of "may be"

***Guideline*:** The term "may be" is used only to indicate a possible use of an element or attribute; it does not denote that the element or attribute is optional. Optionality is defined in the Minimum Occurrence (MinOcc) indicator of the element and the Use indicator of the attribute.

Rationale: Consistency in terminology helps eliminate confusion between usage and optionality.

Example:

```
 "May be used to give further detail on the code or to remove an obsolete item."
```

4.8.7 Reference to Code Tables

Guideline: When the OTA_CodeType is used, the following must be included "Refer to OTA Code List nnn nnn nnn (xxx)" where nnn is the name of an OTA Code List and xxx is its 3-character identifier.

Rationale: This reference enables the reader or implementer of a Schema to find the code values of the referenced OTA code table.

Example:

```
Refer to OTA Code List Room Amenity Type (RMA).
```

4.8.8 No Use of Processing Instructions

Guideline: OTA XML Schemas will avoid the use of Processing Instructions (PI) by replacing them with the <appinfo> sub-element of the <annotation> element that supplies this functionality.

Rationale: <`appinfo`> elements are available to users of the Schema. PIs require knowledge of their notation to be parsed correctly. Extensions to the XML Schema can be made using <`appinfo`>. An extension will not change the Schema-validity of the document.

4.9 Enumerations vs. Code Lists

4.9.1 Use of Enumerations

Guideline: Enumerations are used in the case where the list of values is static or there is little likelihood that additional values will be added.

Rationale: This method allows for the values to be validated.

Example:

```
<xs:attribute name="Gender" use="optional">
           <xs:simpleType>
               <xs:restriction base="xs:NMTOKEN">
                    <xs:enumeration value="Male"/>
                    <xs:enumeration value="Female"/>
                   <xs:enumeration value="Unknown"/>
               </xs:restriction>
           </xs:simpleType>
      </xs:attribute>
```

4.9.2 Use of Code Lists

Guideline: Code lists are used in the case where the list of values is dynamic or there is great likelihood that additional values will be added.

Rationale: This method allows for new codes to be added and used between releases.

Example:

```
Communication Location Type
1  Home
2  Business
3  Other
```

4.10 Code Lists

4.10.1 Name of Code List Table

Guideline: The name of a code list table should be the same or similar to the name of the attribute in XML Schema, but should be in plain English with spaces between the words.

Rationale: This approach provides the reader or implementer with better understanding of how the code values are used.

Example:

```
Code set name Coverage Type for <xs:attribute name="CoverageType"/>
Code set name Phone Technology Type for <xs:attribute name="PhoneTechType"/>
```

4.11 OTA General

4.11.1 Required Attributes of XML Instance Root Elements

Guideline: The root element of all OTA payload documents (XML instance messages), must contain the following attributes:

- xmlns="http://www.opentravel.org/OTA/2003A/05"
- Version="[current version here]"
- xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
- xsi:schemaLocation="http://www.opentravel.org/…"

Rationale: This format provides a standard way to identify OTA payload messages, message version, and the corresponding XML Schema.

Example:

```
<OTA_VehAvailRateRQ
           xmlns="http://www.opentravel.org/OTA"
           Version="1.001"
            Xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.opentravel.org/OTA
http://www.opentravel.org/OTA/2002A-REC/VEH-availability/VehAvailRateRQ.xsd">
         <!--  Payload content… -->
</OTA_VehAvailRateRQ>
```

4.11.2 Use of TPA_Extensions

Guideline: Trading partner-specific data can be included in an XML instance message within the <TPA_Extension> global element at OTA-sanctioned plug-in points defined in the XML Schema. This element may also contain the Boolean attribute 'mustProcess', which notifies that the message receiver must process the 'TPA_Extension' data.

Rationale: This approach provides a standard way for OTA to integrate and manage specific trading partner information.

Example: Schema fragment:

```
<xs:element name="TPA_Extension" type="xs:anyType">
```

Sample XML:

```
<OTA_VehResRQ  xmlns="http://www.opentravel.org/OTA/2003/05"
           Version="1.23"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.opentravel.org/OTA
http://www.opentravel.org/OTA/2002A-REC/VEH-booking/VehResRQ-Tze123.xsd">
   <POS>
       <Source PseudoCityCode="ABC123" AgentSine="123456789"/>
           <UniqueId URL="http://switch.com/OTAEngine/"
                           Type="VehResRQ" Id="123456"/>
           <BookingChannel Type="GDS"/>
       </Source>
       <TPA_Extension mustProcess="1">
           <NegotiatedService Type="TourGuideDriver"/>
       </TPA_Extension>
   </POS>
   <VehRequest>
       <!—OTA VehRequest content -->
   </VehRequest>
</OTA_VehResRQ>
```

4.11.3 Standard Simple Types vs. OTA Simple Types

Guideline: Wherever possible, OTA Schema data types should use the standard built-in simple types defined in the XML Schema specification.

Rationale: This approach simplifies OTA message implementation because validation tools support built-in XML Schema simple types.

4.11.4 New Data Types Based on Extending Existing Types

Guideline: Create new Schema data types by using or extending existing OTA type definitions or from built-in XML Schema types whenever possible.

Rationale: This technique maximizes reuse and avoids duplicating definitions.

4.11.5 Type Restrictions

Guideline: OTA XML Schemas should avoid rigid type restrictions unless the type is a common industry standard which is unlikely to change.

Rationale: This approach allows OTA messages to interoperate globally in a more seamless manner and allows any particular trading partner to locally restrict content values as needed for unique business requirements.

