OTA

The OpenTravel™ Alliance

OpenTravel™ Alliance Best Practices

OpenTravel Alliance, Inc.
333 John Carlyle Street, Suite 600
Alexandria, Va. 22314 USA
+1 703-548-7005
Fax +1 703-548-1264
opentravel@disa.org
http://www.opentravel.org/

Prepared in partnership with Data Interchange Standards Association (http://www.disa.org)

Table of Contents

1 OTA XML 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 Corporations can apply these “increased earning potential principles” is by establishing a common set of best practice XML and XML Schema guidelines.

This document defines the OpenTravel Alliance’s Best Practices Guidelines for all of OTA’s XML data assets. OTA approved message specifications released prior to version base 2001C may not follow the guidelines defined in this document. However, the approval of any OTA specification to be released with version 2001C (or beyond) will be based on how well it complies with OTA’s Best Practice Guidelines.

1.1 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/

1.2 Best Practices

1.2.1 Scope

The OTA Best Practices Guidelines cover all of OTA's XML components (elements, attributes, tag names and Schema definitions).

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

1.2.2 XML Component Parts and Roles

The critical XML components that best support OTA's goal of a consistent set of reusable travel industry message content are listed below:

- Tag Naming conventions
- Elements vs. Attributes
- DTD vs. XML Schema
- Global vs Local Element Types and Elements/Attributes
- Namespaces
- Versioning XML Schemas
- XML Markup - General
- OTA General

Each of the eight items above plays a unique role, supporting a common vocabulary, syntax, and semantic grammar for XML Schema and XML component (element and attribute) definitions. Also, each of the guidelines details its specific role in the rationale section. This document defines OTA guidelines for all XML data assets.

1.3 OTA XML Guidelines

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

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

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

Example: An example (if applicable).

1.3.1 Tag Naming Conventions (I)

1.3.1.1 XML Tag Names (I-1)

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 between words (a.k.a. "UCC camel case" or "PascalCasing").

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

Example: <WorkAddress> <PostalCode>

1.3.1.2 XML Tag Names (I-2)

Guideline: Acronym abbreviations 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: <BusinessURL> <HomeUSA>

1.3.1.3 XML Tag Names (I-3)

Guideline: Word abbreviations are discouraged, however where needed, word abbreviations should 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 acronyms.

Example: <ProductInfo> <BldgPermit>

1.3.1.4 XML Tag Names (I-4)

Guideline: Element and attribute names should not exceed 25 characters. Tag names should be spelled out except where they exceed 25 characters, then 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>

1.3.1.5 XML Tag Names (I-5)

Guideline: Where the merger of tag name words and acronyms cause 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>, <UDDI_Keys>

1.3.1.6 XML Tag Names (I-6)

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: <ContactAddress> <HomeAddress> <WorkAddress>

1.3.1.7 XML Tag Names (I-7)

Guideline: The OTA approved, defined or derived 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.

Simple datatype	Description
Amount	A number of monetary units specified in a currency
Code	A character string that represents a member of a set of values.
Date	A day within a particular calendar year. Note: Reference ISO 8601 (CCYY-MM-DD).
Time	The time within any day in public use locally, independent of a particular day. Reference ISO 8601: 1988 (hh:mm:ss[.ssss…[Z | +/-hh:mm]])
DateTime	A particular point in the progression of time. (CCYY-MM-DD [**T**hh:mm:ss [.ssss…[Z | +/-hh:mm]]]). Reference ISO 8601.
Boolean	Examples: true, false
Identifier	A character string used to identify and distinguish uniquely, one instance of an object within an identification scheme (standard abbreviation ID).
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.
Type	An enumerated list of values from which only one value can be chosen at a time.

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.

Example:

```
<xs:complexType name="TravelEventDateTime">
  <xs:simpleContent>
    <xs:extension base="xs:dateTime">
      <xs:attribute name="type">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="depart"/>
            <xs:enumeration value="arrive"/>
            <xs:enumeration value="pick-up"/>
            <xs:enumeration value="drop-off"/>
            <xs:enumeration value="checkin"/>
            <xs:enumeration value="checkout"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

1.3.2 Elements vs. Attributes (II)

1.3.2.1 Elements vs. Attributes (II-1)

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
- presence or absence of the attribute represents a semantic 'choice' or branch within the schema OR
- an element should also be used where 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 to avoid the potential of tag naming collisions.

Example:

Element:

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

Attribute:

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

1.3.2.2 Elements vs. Attributes (II-2)

Guideline: Do not overload element tags with too many attributes (no more than 10 as a rule of thumb) by encapsulating 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.

[1] URLs are considered less than 64 characters

Rationale: Maintains the built-in extensibility XML provides with elements and is necessary to provide backward compatibility as the specification evolves. It also provides a consistent guide to the level of granularity used to compose OTA's schema objects (or fragments).

1.3.2.3 Elements vs. Attributes (II-3)

Guideline: Multiple XML element containers must be used for repeating complexType elements if the XML Schema 'maxOccurs' attribute exceeds 6 repititions. The encapsulating element container is optional if the XML Schema 'maxOccurs' attribute is less-than or equal to 6. However, a single XML <element> container can be used for "simpleType" repeating content (via the XML Schema "list" construct).

Rationale: Provides consistency for OTA approved 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" "GA"/>
```

1.3.3 DTD vs. XML Schema (III)

1.3.3.1 DTD vs. XML Schema (III-1)

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 datatypes and an extensible data-typing mechanism. (DTDs only understand markup and character data.) Using an XML syntax to define data model requirements allows for more constraints, strong datatyping, etc and provides for a consistent Data Repository syntax.

1.3.4 Global vs Local Element Types and Elements/Attributes (IV)

1.3.4.1 Global vs Local Element Types and Elements/Attributes (IV-1)

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 Schema type names are not contained in XML instance documents, they can be verbose to avoid Schema element type naming collisions.

1.3.4.2 Global vs Local Element Types and Elements/Attributes (IV-2)

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 naming collisions.

Example:

```
	<xs:complexType name="AddressType">
		<xs:sequence>
			<xs:element name="StreetNmbr" type="StreetNmbrType" minOccurs="0">
				<xs:annotation>
					<xs:documentation xml:lang="en">Street Name and Number within the address</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="BldgRoom" type="StringLength64" minOccurs="0">
				<xs:annotation>
					<xs:documentation xml:lang="en">Building name, room, apartment, or suite
number.</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="AddressLine" type="StringLength64" minOccurs="0" maxOccurs="5"/>
			<xs:element name="CityName" type="StringLength64" minOccurs="0">
				<xs:annotation>
					<xs:documentation xml:lang="en">City name eg. Dublin</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="PostalCode" type="StringLength16" minOccurs="0">
				<xs:annotation>
					<xs:documentation xml:lang="en">Post Office Code number.</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="County" type="StringLength32" minOccurs="0">
				<xs:annotation>
					<xs:documentation xml:lang="en">County Name eg. Fairfax</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="StateProv" type="StateProvType" minOccurs="0">
				<xs:annotation>
					<xs:documentation xml:lang="en">State name eg. Texas</xs:documentation>
				</xs:annotation>
			</xs:element>
			<xs:element name="CountryName" type="CountryNameType" minOccurs="0">
				<xs:annotation>
					<xs:documentation xml:lang="en">Country name eg. Ireland</xs:documentation>
				</xs:annotation>
			</xs:element>
		</xs:sequence>
		<xs:attributeGroup ref="FormattedInd"/>
		<xs:attributeGroup ref="PrivacyGroup"/>
		<xs:attribute name="Type" type="OTA_CodeType" use="optional">
			<xs:annotation>
				<xs:documentation>Defines the type of address such as home, business, other, etc.</xs:documentation>
			</xs:annotation>
		</xs:attribute>
	</xs:complexType>
```

1.3.4.3 Global vs Local Element Types and Elements/Attributes (IV-3)

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 naming collisions with other ‘attributeGroup’ definitions.

Example:[3]

```
<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>
```

1.3.5 Namespaces (V)

1.3.5.1 Namespaces (V-1)

Guideline: All message schemas specified as compliant with OTA's XML message specifications SHALL put the global names they declare in an OTA namespace which is of the form: http://www.opentravel.org/OTA/yyyy/mm. Simple atomic data type schema files and common complexType schema files SHOULD have either a generic fixed namespace or no namespace to allow common content to transcend multiple message schema versions.

Rationale: This approach supports a consistent way to manage and identify OTA’s 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 prefix qualifiers on both schema and instance docs.

1.3.5.2 Namespaces (V-2)

Guideline: Each XML instance document produced by the 'OTA' namespaced 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: The same rationale as V-1 above. Also, provides a standard way for “OTA” namespaced content to be merged with other Industry or Trading Partner namespace content.

1.3.5.3 Namespaces (V-3)

Guideline: Each XML schema document produced as 'OTA' namespaced message schema should specify a default namespace and a targetNamespace and both should be the 'OTA' namespace.

Rationale: The same rationale as V-1 above.

Example:

[3] For the complete definition of the attributeGroup OTA_PayloadStdAttributes see section 2.4.2

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.opentravel.org/OTA/yyyy/mm"
    xmlns="http://www.opentravel.org/OTA/yyyy/mm"
    version="2002A">
```

1.3.6 Versioning XML Schemas (VI)

1.3.6.1 Versioning XML Schemas (VI-1)

Guideline: The root tag for all XML payload instances should contain a ‘version’ attribute (obtained from the attributeGroup ‘OTA_PayloadStdAttributes’) whose value will mimic the OTA version release, plus OTA restricted extension meta-data (if extensions are present).[4] Additionally, the ‘schemaLocation’ attribute should contain a URI that corresponds to the location of the schema version (requester’s, receiver’s or common repository) defining this particular XML payload instance.

Rationale:This approach supports automated schema discovery and provides sufficient version meta-data for repository maintenance. It also provides a quick and simple way for human or machine users to identify XML message transaction versions.

Example: [5]

```

‘2002A’;  ‘2002B.23’;  ‘2003A.Tabc123’;  ‘2001C.Txyz456’
  where:
    ‘2002A’ –   is the bi-annual year of release (ie. 2002B, 2003A).

     ‘.23’  –   A numeric reference value ‘.nn’ to the base OTA schema
                version (ie. 2002A) which contains extensions derived
                from an XML Schema simpleType or from an existing
                OTA Schema type (simple or complex).

    ‘.Tabc123’ – This extension type flags the presence of the
                <TPA_Extension> element within an XML instance
                message.  The OTA based schema for this type of
                version extension can be any OTA version, base or
                extension.  The format for this extension is:  ‘T’ followed
                by a short string that user trading partners can recognize.
```

1.3.6.2 Versioning XML Schemas (VI-2)

Guideline: Each version of a schema produced under the 'OTA' namespace must have a unique value for the 'version' attribute of the <xs:schema> opening tag. The value must correspond to the OTA payload message root tag name as shown in the example following. Additonally, if ebXML is used as the transport medium, the version value MUST be duplicated within the ebXML Manifest as the value of its grandchild <eb:schema> element.

Rationale: Using a version mechanism that parallels the schema-discovery mechanism of validating XML parsers is desirable and is supported by many schema validation tools. Additionally, having a versioning scheme that mimics OTA's specification release methodology reduces the overall work effort of both schema publication and maintainability. Similarly, the 'domain_path' recommendations can greatly reduce the work required to maintain and XML Schemas and schema fragments. This feature can be further enhanced by supplying the 'domain_path' as schema meta-data in a repository tool.

Example:

```
    <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema"
    targetNamespace = "http://www.opentravel.org/OTA/yyyy/mm"
    xmlns = "http://www.opentravel.org/OTA/yyyy/mm"
    version = "2002A.123">
```

where:

[4] a numeric extension type is used only for OTA approved Use Cases where the unique added content satisfies an important need - however, is deemed not common enough to migrate in the base version (Controlled by OTA’s InterOperability committee). Alternatively, OTA allows an extension starting with letter ‘T’ for trading partners to add proprietary content via the <TPA_Extension> element specified in OTA XML Schemas at specific locations (see examples below; also see guideline ‘VIII-2’ for a <TPA_Extension> example).

[5] related to guideline example in section ‘VI – 2’.

'2002A.123' - bi-annual base version release (ie. 2002B, 2003A) and a possible extension (ex: ".123" or ".Tabc123"). Extensions starting with ".T" signify a TPA extension agreement between trading partners while extensions without a letter following the "." are approved OTA extension versions.

1.3.7 XML Markup – General (VII)

1.3.7.1 XML Markup - General (VII-1)

Guideline: The attribute schemaLocation replaces the DOCTYPE and can be used on elements in instances to name the location of a retrievable schema for that element associated with that namespace.

Rationale: Supports OTA's decision to use XML Schemas, which are not aware of this construct.

1.3.7.2 XML Markup - General (VII-2)

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

Rationale: Comments are not part of the core information set of a document and may not be available or in a useful form. However, <documentation> elements are available to users of the Schema.

Example:

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

1.3.7.3 XML Markup - General (VII-3)

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

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

1.3.8 OTA General (VIII)

1.3.8.1 OTA General (VIII-1)

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

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

Rationale: Provides a standard way to identify OTA payload messages, message version and the corresponding schema.

1.3.8.2 OTA General (VIII-2)

Guideline: Proprietary trading partner data can be included in an XML instance message within the <TPA_Extension> global element at OTA sanctioned plug-in points defined in the schema.

Rationale: This approach (along with the versioning Guideline of VI-2) provides a standard way for OTA to integrate and manage proprietary trading partner information.

Example: schema fragment:

```
	<xs:element name="TPA_Extensions" type="TPA_Extensions_Type">
		<xs:annotation>
			<xs:documentation>A place holder in the schema to allow for additional elements and attributes to be included
if required, per Trading Partner Agreement (TPA).</xs:documentation>
		</xs:annotation>
	</xs:element>
	<xs:complexType name="TPA_Extensions_Type">
		<xs:sequence>
			<xs:annotation>
				<xs:documentation xml:lang="en">Extensions
```

```
allowed to OTA specification per bilateral agreement between trading
partners.</xs:documentation>
                        </xs:annotation>
                        <xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
```

1.3.8.3 OTA General (VIII-3)

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

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

1.3.8.4 OTA General (VIII-4)

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

Rationale: Maximizes reuse and avoids definition duplication.

1.3.8.5 OTA General (VIII-5)

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 defined messages to inter-operate globally more seamlessly and allows any particular trading partner to locally restrict content values as needed for unique business requirements.

1.3.8.6 OTA General (VIII-6)

Guideline: When adding a preference level attribute qualifier to an OTA element, a complexType definition should be created which extends the base type of the element using the attribute group PreferLevel.

Rationale: This approach provides a standard way for creating and processing OTA element preferences.

Example: schema fragment:

```
 <xs:complexType name="FreeTextType>
  <xs:annotation>
   <xs:documentation>Provides textual information in regarding message
context
   </xs:documentation>
   <xs:simpleContent>
     <xs:restriction base="xs:string"/>
   </xs:simpleContent>
 </xs:complexType>

        extends to the following new type:

 <xs:complexType name="FreeTextPrefType>
  <xs:annotation>
   <xs:documentation>Provides textual information in regarding message
context including its preference level
   </xs:documentation>
   <xs:simpleContent>
     <xs:extension base="FreeTextType">
      <xs:attributeGroup ref="PreferLevelType">
   </xs:simpleContent>
 </xs:complexType>
```