OpenTravel™ Alliance
2001C Infrastructure Specification

Abstract

This document presents the specifications for the underlying infrastructure for the exchange of messages in the travel industry, covering travel services for airlines, car rentals, hotels, and travel insurance. It uses the Extensible Markup Language (XML) for the exchange of these messages transmitted under Internet protocols and includes a detailed mapping onto ebXML Message Services.

Section 1 - Introduces this OTA specification, its intended audience and conventions used throughout the document.

Section 2 – Best Practices for development of XML messages for use within OTA specifications.

Section 3 – Generic Messages and the Service/Action Model describe generic messages, defined within the infrastructure, that are used within various vertical specific message sets. The service/action model provides a web-services like model for the exchange of messages between trading partners.

Section 4 - Infrastructure, the underlying architectural model, and details of the on-the-wire data for OTA message exchange over an ebXML Messaging Service.

Section 5 – Detailed explanation and examples of the generation and processing of messages that use the OTA generic update method.

Section 6 – Definition of Service/Action mappings for all current OTA messages. Recommendations are also made for an optimum class of delivery.

Section 7 – Summary of infrastructure from previous OTA specifications.

Section 8 – Utilizing ebXML v2.0 in OTA Solutions.

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)

OpenTravel™ Alliance, Inc.

License Agreement

Authorization to Use Specifications and documentation

IMPORTANT: The OpenTravel™ Alliance ("OTA") Message Specifications ("Specifications"), whether in paper or electronic format, are made available subject to the terms stated below. Please read the following carefully as it constitutes a binding Agreement, based on mutual consideration, on you and your company as licensee ("You").

1. **Documentation.** OTA provides the Specifications for voluntary use by individuals, partnerships, companies, corporations, organizations, and other entities at their own risk. The Specifications and any OTA supplied supporting information, data, or software in whatever medium in connection with the Specifications are referred to collectively as the "Documentation."

2. **License Granted.**

   2.1. OTA holds all rights, including copyright, in and to the Documentation. OTA grants to You this perpetual, non-exclusive license to use the Documentation, subject to the conditions stated below. All use by You of the Documentation is subject to this Agreement.

   2.2. You may copy, download, and distribute the Documentation and may modify the Documentation solely to allow for implementation in particular contexts. You may bundle the Documentation with individual or proprietary software and/or sublicense it in such configurations.

   2.3. You must reference, in a commercially reasonable location, the fact that the OTA Documentation is used in connection with any of your products or services, in part or in whole, whether modified or not, and You may include truthful and accurate statements about Your relationship with OTA or other use of the Documentation.

   2.4. However, you may not change or modify the Specification itself, develop a new standard or specification from the Documentation, or state or imply that any works based on the Documentation are endorsed or approved by OTA.

   2.5. You must include the OTA copyright notice in connection with any use of the Documentation. Any uses of the OTA name and trademarks are subject to the terms of this Agreement and to prior review and approval by OTA.

   2.6. Nothing in this Agreement shall be interpreted as conferring on You or any other party any other interest in or right to the Documentation. Nothing in this Agreement shall be interpreted as in any way reducing or limiting OTA's rights in the Documentation.

3. **LIABILITY LIMITATIONS.** *THIS AGREEMENT IS SUBJECT TO THE FOLLOWING LIABILITY LIMITATIONS*:

3.1. ANY DOCUMENTATION PROVIDED PURSUANT TO THIS NON-EXCLUSIVE LICENSE AGREEMENT IS PROVIDED "AS IS" AND NEITHER OTA NOR ANY PERSON WHO HAS CONTRIBUTED TO THE CREATION, REVISION, OR MAINTENANCE OF THE DOCUMENTATION MAKES ANY REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE.

3.1. Neither OTA nor any person who has contributed to the creation, revision or maintenance of the documentation makes any representations or warranties, express or implied, that the use of the documentation or software will not infringe any third party copyright, patent, patent application, trademark, trademark application, or other right.

3.2. Neither OTA nor any person who has contributed to the creation, revision, or maintenance of the documentation shall be liable for any direct, indirect, special or consequential damages or other liability arising from any use of the documentation or software. You agree not to file a lawsuit, make a claim, or take any other formal or informal action against OTA based upon Your acquisition, use, duplication, distribution, or exploitation of the Documentation.

3.3. The foregoing liability limitations shall apply to and be for the benefit of OTA, any person who has contributed to the creation, revision or maintenance of the documentation, and any member of the board of directors, officer, employee, independent contractor, agent, partner, or joint venturer of OTA or such person.

4. **No Update Obligation.** Nothing in this Agreement shall be interpreted as requiring OTA to provide You with updates, revisions or information about any development or action affecting the Documentation.
5. **No Third Party Beneficiary Rights.** This Agreement shall not create any third party beneficiary rights.
6. **Application to Successors and Assignees.** This Agreement shall apply to the use of the Documentation by any successor or assignee of the Licensee.
7. **Term.** The term of this license is perpetual subject to Your compliance with the terms of this Agreement, but OTA may terminate this License Agreement immediately upon your breach of this Agreement and, upon such termination you will cease all use duplication, distribution, and/or exploitation of the Documentation in any manner.
8. **Interpretation and Choice of Forum.** The law of the Commonwealth of Virginia and any applicable Federal law shall govern this Agreement. Any disputes arising from or relating to this Agreement shall be resolved in the courts of the Commonwealth of Virginia, including Federal courts. You consent to the jurisdiction of such courts and agree not to assert before such courts any objection to proceeding in such forum.
9. **Acceptance.** Your acceptance of this License Agreement will be indicated by your affirmative acquisition, use, duplication, distribution, or other exploitation of the Documentation. If you do not agree to these terms, please cease all use of the Documentation now.
10. **Questions.** Questions about the Agreement should be directed to www.opentravel.org.

Table of Contents

1 About this Document

This version of the Open Travel Alliance specification constitutes a major revision to the underlying technical architecture and a significant expansion of the 'best practices' recommendations pertaining to OTA message sets.

This revision supercedes 2001A part 1 which was related to architecture and infrastructure. Part 2 of 2001A which contains profile message specifications still stands.

1.1 Intended Audience

This document serves two distinct audiences:

- Working group members who are actively working on designing or revising XML messages for use within OTA specifications. For these working group members the sections 'XML Best Practices' and 'Generic Messages and the Service Action Model' are of interest and provides useful guidelines and recommendations to aid in their work
- Software implementation teams working on an implementation of OTA specifications. For this audience the section 'OTA Infrastructure' will be of particular interest

1.2 Definitions and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in IETF document RFC 2119.

This text makes every attempt to accurately reflect the XML Schemas listed in the Appendices. In the case of a conflict between the document text and the XML Schema, the XML Schema takes precedence.

1.3 Relationship with previous OTA standards

This specification supercedes the infrastructure defined in the OTA 2001A specification, part 1. (Part 2 of 2001A is still applicable as it relates to message definitions). In 2001A best practices, infrastructure and generic messages were interspersed within a single large section. To make this specification easier to read and to allow its audience to focus on the sections which are applicable to their particular work, the overall structure has been revised with separate major sections defining:

- XML best practices
- Generic messages and the Service/Action model
- OTA Infrastructure

1.4 Relationship with ebXML standards

EbXML is sponsored by UN/CEFACT and OASIS as a modular suite of specifications that enable enterprises of any size and in any geographical location to conduct business over the Internet. EbXML Transport Routing and Protocol runs on top of W3C SOAP 1.1, http://www.w3.org/TR/SOAP/ and SOAP with Attachments, providing deeper infrastructure when required such as reliable message delivery and an enhanced security model.

OTA RECOMMENDS ebXML as a viable infrastructure for the exchange of OTA messages across private and public networks and the Infrastructure section in this document covers a detailed mapping of OTA payloads onto an ebXML framework.

OTA implementations are not REQUIRED to use ebXML infrastructure, and parties agreeing bilaterally may use any method for message exchange they mutually agree upon. OTA's goals in making this recommendation are:

- to provide a viable and robust infrastructure which is both open and available
- to enable off-the-shelf implementations on a variety of platforms
- to allow on-the-wire interoperability between implementations

1.5 Use of Namespaces in this specification

Unless otherwise qualified with a prefix, all elements and attributes within this specification are assumed to be within the OTA namespace which is defined as follows:

```
xmlns="http://www.opentravel.org/OTA"
```

The following table defines all namespace prefixes used within this document and their applicable namespaces:

Prefix	*Namespace definition*
{nil}	`xmlns="http://www.opentravel.org/OTA"`
OTA:	`xmlns:OTA="http://www.opentravel.org/OTA"`
eb:	`xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"`
tp:	`xmlns:tp="http://www.ebxml.org/namespaces/tradePartner"`
xlink:	`xmlns:xlink="http://www.w3c.org/1999/xlink"`
xs:	`xmlns:xs="http://www.w3c.org/2001/XMLSchema"`
xsi:	`xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"`
SOAP-ENV:	`xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"`

2 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.

The current W3C XML specifications were created to satisfy a very wide range of diverse applications and this 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 and related standards in that environment.

This document defines the Open Travel 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.

2.1 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/

2.2 Best Practices

2.2.1 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 mechinism XML Schemas have to support this approach is via encapsulating lower level components (element and attribute objects) with-in named type definitions while using (and reusing) this base components to construct messages.

2.2.2 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:

I. Tag Naming conventions
II. Elements vs. Attributes
III. DTD vs. XML Schema
IV. Global vs Local Element Types and Elements/Attributes
V. Namespaces
VI. Versioning XML Schemas
VII. XML Markup - General
VIII. OTA General

Each of the eight items above play 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.

2.3 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).

2.3.1 Tag Naming Conventions (I)

2.3.1.1 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>

2.3.1.2 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>

2.3.1.3 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>

2.3.1.4 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 reduce to:
<ShareSyncInd>

2.3.1.5 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>

2.3.1.6 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>

2.3.1.7 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>
```

2.3.2 Elements vs. Attributes (II)

2.3.2.1 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” />
```

2.3.2.2 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.

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).

2.3.2.3 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.

[1] URLs are considered less than 64 characters

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"/>
```

2.3.3 DTD vs. XML Schema (III)

2.3.3.1 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.
- Provides for a consistent Data Repository syntax.

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

2.3.4.1 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.

2.3.4.2 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=" 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>
```

2.3.4.3 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>
```

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

2.3.5 Namespaces (V)

2.3.5.1 Namespaces (V-1)

Guideline: All schemas specified as compliant with OTA's XML message specifications SHALL put the global names they declare in one namespace; this SHALL be the 'OTA' namespace, which is http://www.opentravel.org/OTA .

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 prefixes on both schema and instance docs.

2.3.5.2 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.

2.3.5.3 Namespaces (V-3)

Guideline: Each XML schema document produced as an 'OTA' namespaced 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:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://www.opentravel.org/OTA"
       xmlns="http://www.opentravel.org/OTA"
       version="http://www.opentravel.org/OTA/2002A-REC/VEH-availability/VehAvailRateRQ-
23.xsd">
```

2.3.6 Versioning XML Schemas (VI)

2.3.6.1 Versioning XML Schemas (VI-1)

Guideline: The root tag for all XML payload instances should contain a 'version' attribute (obtained from the 2001C attributeGroup '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.

[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).

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.
```

2.3.6.2 Versioning XML Schemas (VI-2)

Guideline: Each version of a schema produced under the 'OTA' namespace must have a unique URI value for the 'version' attribute of the <xs:schema> opening tag. The URI must value must correspond to both the OTA payload message root tag name and to the root tag's version attribute value as shown in the format following example.[7] Additonally, if ebXML is used as the transport medium, the URI 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. Also, the 2001C Service and Action transaction model can leverage the 'domain_path' as the Service token.

Example:

```
       <xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema"
       targetNamespace = "http://www.opentravel.org/OTA"
       xmlns = "http://www.opentravel.org/OTA"
       version = "http://www.opentravel.org/OTA/2002A-REC/roottag-nn.xsd">

where:
         '2002A' – bi-annual base version release (ie. 2002B, 2003A).

         'REC'   – can be either 'TEST', 'CAN' or 'REC' which is
                   Test, Candidate or Recommendation message status respectively.

         'roottag' – OTA Payload root tag name.
```

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

[7] An exception to this attribute value exist for OTA schema fragment files – where the value is defined as a simple positive integer.


```
        '-nn'     – is an OTA Schema extension meta-data string which is
                    defined in detail in Guideline 'VI-1'.
```

2.3.7 XML Markup – General (VII)

2.3.7.1 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.

Example:

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

2.3.7.2 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>
```

2.3.7.3 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.

2.3.8 OTA General (VIII)

2.3.8.1 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.

Example:[9]

```
<OTA_VehAvailRateRQ
           xmlns=”http://www.opentravel.org/OTA”
           Version=”2002A”
            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>
```

2.3.8.2 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. This element may also contain the boolean attribute ‘mustProcess’ which notifies that the message receiver must process the ‘TPA_Extension’ data.

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_Extension" type=”xs:anyType”>
```

Sample XML:

```
<OTA_VehResRQ  xmlns=”http://www.opentravel.org/OTA”
           Version=”2002A.Tze123”
            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>
```

2.3.8.3 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.

2.3.8.4 OTA General (VIII-4)

[9] attribute details are shown in guidelines ‘IV-3’ and ‘VI-2’

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.

2.3.8.5 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.

2.3.8.6 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>
```

2.4 Response Message Design

The OTA specification provides for returning application errors when those errors result from interactions with the trading partner's server. This section outlines specific requirements for response messages and any associated errors.

Typically, if a business message, such as updating a customer profile, fails for a business level reason, the business message itself should use the expected response message <xxxRS> to declare a failure when it is returned. This response has meaning only in the context of the business message, based on the notion that a business content level error constitutes the response.

2.4.1 Standard Payload Attributes[10]

The response message <xxxRS> contains the AttributeGroup PayloadStdAttributes on the root element. The meaning and usage of each of these standard attributes is as follows:

- *EchoToken*: a sequence number for additional message identification assigned by the requesting host system. When a request message includes an *EchoToken*, the corresponding response message MUST include an *EchoToken* with an identical value.
- *TimeStamp*: indicates the creation date and time of the message in UTC using the following format specified by ISO 8601: YYYY-MM-DDThh:mm:ssZ with time values using the 24-hour (military) clock. e.g. 20 November 2000, 1:59:38pm UTC becomes 2000-11-20T13:59:38Z
- *Target*: indicates if the message is a test or production message, with a default value of Production. Valid values: (Test | Production)
- *Version*: For all OTA versioned messages, the version of the message is indicated by an integer value.
- *SequenceNmbr* - This optional attribute is used to identify the sequence number of the transaction as assigned by the sending system. Allows for an application to process messages in a certain order, or to request a resynchronization of messages in the event that a system has been offline and needs to retrieve messages that were missed.

Error messages (and warnings), for any valid OTA response message provide a facility to help trading partners identify the outcome of a message.

Note: All OTA versioned message requests MAY result in a response message that consists of a non-versioned StandardError construct alone. When a <StandardError> is not returned, trading partners should be able to quickly determine whether the request succeeded, or had other errors identified by the application that processed the request.

Therefore, every <xxxRS> element MUST have an optional <Success/> element as its first child. The presence of the empty <Success/> element explicitly indicates that the OTA message succeeded. In addition to <Success/>, an implementation may return <Warnings> in the event of one or more non-fatal business context errors, OR <Errors> in the event of a failure to process the message altogether.

2.4.1.1 Mapping OTA Payload Attributes onto ebXML MS Headers

Each of the standard payload attributes has an equivalent in a standard ebXML MS envelope, and as such it is NOT RECOMMENDED that any standard payload attribute except “version” be included when OTA messages are transported over an ebXML MS. The following table details this mapping:

Std Payload Attribute	***EbXML MS element attribute***	***Comments***
EchoToken	eb:MessageId, eb:RefToMessageId	Response messages are matched with requests by placing the eb:MessageId

[10] With the exception of the Version attribute other standard payload attributes are redundant when using the OTA RECOMMENDED ebXML transport. These attributes are provided for backward compatibility and to provide some semblance of infrastructure when operating in a non-ebXML environment.

			element value from the request in the eb:RefToMessageId element of the response
TimeStamp	eb:Timestamp		Automatically generated by an ebXML MS
Target	OTA:SessionControlRequest	Mode	Negotiated during session setup. Possible values: “Test” or “Production”
Version	-	-	No equivalent. OTA RECOMMENDS the Version attribute be sent with each message
SequenceNumber	eb:SequenceNumber		Only applicable for type-B messages

2.4.2 Design Patterns for Response Messages

Error and Warning elements share a common definition (with the exception of the tag name). These common elements and parameter entities provided for convenience when defining message response schemas are defined in the standard include schema fragment "*OTA_v2ent.xsd*" shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created and edited with 'vi' -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.opentravel.org/OTA"
           targetNamespace="http://www.opentravel.org/OTA"
            elementFormDefault="qualified">

    <xs:annotation>
        <xs:documentation xml:lang="en">
OTA v2001C Specification - General elements used in response messages
Copyright (C) 2001 Open Travel Alliance. All rights reserved.
        </xs:documentation>
    </xs:annotation>

    <xs:include schemaLocation="UniqueId.xsd"/>

    <xs:attributeGroup name="OTA_PayloadStdAttributes">
        <xs:annotation>
            <xs:documentation>The OTA_PayloadStdAttributes defines the standard attributes
that appear on the root element for all OTA payloads where ebXML is not being used (with the
exception of version).</xs:documentation>
        </xs:annotation>
        <xs:attribute name="EchoToken" type="xs:string"/>
        <xs:attribute name="TimeStamp" type="xs:string"/>
        <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="xs:string"/>
        <xs:attribute name="SequenceNmbr" type="xs:integer"/>
    </xs:attributeGroup>

    <xs:attributeGroup name="ReqRespVersion">
        <xs:annotation>
```

```
            <xs:documentation>The ReqRespVersion attribute is used to request the version of
the payload message desired for the response.</xs:documentation>
        </xs:annotation>
        <xs:attribute name="ReqRespVersion" type="xs:string"/>
    </xs:attributeGroup>

    <xs:element name="Success">
        <xs:complexType>
            <xs:annotation>
                <xs:documentation>Standard way to indicate success processing an OTA
message</xs:documentation>
            </xs:annotation>
        </xs:complexType>
    </xs:element>

    <xs:element name="Warnings">
        <xs:complexType>
            <xs:annotation>
                <xs:documentation>Indicates successful processing, but warnings
occured.</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element ref="Warning" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="Errors">
        <xs:complexType>
            <xs:annotation>
                <xs:documentation>A Warning and an Error element have the same
structure.</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                <xs:element ref="Error" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name=”ErrorType”>
        <xs:simpleContent>
            <xs:restriction base="xs:string">
                <xs:attribute name="Type" use="required">
                    <xs:simpleType>
                        <xs:restriction base="xs:NMTOKEN">
                            <xs:enumeration value="Unknown"/>
                            <xs:enumeration value="NoImplementation"/>
                            <xs:enumeration value="BizRule"/>
                            <xs:enumeration value="Authentication"/>
                            <xs:enumeration value="AuthenticationTimeout"/>
                            <xs:enumeration value="Authorization"/>
                            <xs:enumeration value="ProtocolViolation"/>
                            <xs:enumeration value="TransactionModel"/>
                            <xs:enumeration value="AuthenticationModel"/>
                            <xs:enumeration value="ReqFieldMissing"/>
                        <xs:enumeration value=”TransportFailure”/>
                            <xs:enumeration value=”EnvelopeFailure”/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="Code" type="xs:string"/>
                <xs:attribute name="DocURL" type="xs:string"/>
                <xs:attribute name="Status" type="xs:string"/>
                <xs:attribute name="Tag" type="xs:string"/>
                <xs:attribute name="RecordId" type="xs:string"/>
            </xs:restriction>
        </xs:simpleContent>
    </xs:complexType>

    <xs:element name="Warning" type=”ErrorType” />
```



```
    <xs:element name="Error" type=”ErrorType” />
</xs:schema>
```

This schema fragment may then be included in other message definition schemas, as the following example illustrates. In this hypothetical example, we define an <ExampleRS> message:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created and edited with 'vi' -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.opentravel.org/OTA"
           targetNamespace="http://www.opentravel.org/OTA"
            elementFormDefault="qualified">
   <xs:include schemaLocation="OTA_v2ent.xsd"/>
   <xs:include schemaLocation=”ExampleElement.xsd”/>

   <xs:element name="ExampleRS">
       <xs:complexType>
           <xs:choice>
               <xs:sequence>
                   <xs:element ref="Success" />
                   <xs:element ref="Warnings" minOccurs="0" />
                   <xs:element ref="Example" />
               </xs:sequence>
               <xs:element ref="Errors" />
           </xs:choice>
           <xs:attributeGroup ref="OTA_PayloadStdAttributes" />
       </xs:complexType>
   </xs:element>
</xs:schema>
```

Note how the definition of the <Example> element, the central element in this <ExampleRS> response message, is imported from another schema fragment ("*ExampleElement.xsd*" in this case).

The attributes of a <Warning> or <Error> element are identical and defined as follows:

Type - The Error element MUST contain the *Type* attribute that uses a recommended set of values to indicate the error type. The initial enumeration list MUST contain:

- *Unknown* - Indicates an unknown error. It is recommended that additional information be provided within the PCDATA, whenever possible.
- *NoImplementation* - Indicates that the target business system has no implementation for the intended request. Additional information may be provided within the PCDATA.
- *BizRule* - Indicates that the XML message has passed a low-level validation check, but that the business rules for the request, such as creating a record with a non-unique identifier, were not met. It is up to each implementation to determine when or if to use this error type or a more specific upper level content error. Additional information may be provided within the PCDATA.
- *AuthenticationModel* - Indicates the type of authentication requested is not recognized. Additional information may be provided within the PCDATA.
- *Authentication* - Indicates the message lacks adequate security credentials. Additional information may be provided within the PCDATA.
- *AuthenticationTimeout* - Indicates that the security credentials in the message have expired. Additional information may be provided within the PCDATA.

- *Authorization* - Indicates the sender lacks adequate security authorization to perform the request. Additional information may be provided within the PCDATA.
- *ProtocolViolation* - Indicates that a request was sent within a message exchange that does not align to the message protocols. Additional information may be provided within the PCDATA.
- *TransactionModel* - Indicates that the target business system does not support the intended transaction-oriented operation. Additional information may be found within the PCDATA.
- *ReqFieldMissing* - Indicates that an element or attribute that is required by the Schema (or required by agreement between trading partners) is missing from the message.
- *VersionViolation* – an invalid/unsupported version of a payload was sent

Code **-** If present, this refers to a table of coded values exchanged between applications to identify errors or warnings.

DocURL - If present, this URL refers to an online description of the error that occurred.

Status - If present, recommended values are `(NotProcessed | Incomplete | Complete | Unknown)` however, the data type is designated as CDATA for versioned message responses, recognizing that trading partners may identify additional status conditions not included in the enumeration.

Tag - If present, this attribute may identify an unknown or misspelled tag that caused an error in processing. It is recommended that the *Tag* attribute use XPath notation to identify the location of a tag in the event that more than one tag of the same name is present in the document. Alternatively, the tag name alone can be used to identify missing data [*Type=ReqFieldMissing*].

RecordId - If present, this attribute allows for batch processing and the identification of the record that failed amongst a group of records.

The following is an example of an error message in which a profile (identified by its UniqueId) was not found:

```
<OTA_ReadProfileRS Version="2">
    <Errors>
        <Error Type="BizRule"> UniqueId 09782345768 not found</Error>
    </Errors>
</OTA_ReadProfileRS>
```


3 Generic Messages and the Service/Action Model

OTA defines a simple service/action model which envelops all defined messages. Although many messages are specific to a particular travel sub-domain (e.g. Air) other messages are generally applicable and may be used more broadly than on one domain-specific service. Many OTA messages are defined in terms of Request/Response pairs (though the infrastructure also provides for reliably-delivered send-only notification type messages) In this section we explore the service/action model and generic messages which are often applicable to multiple high-level services.

3.1 The Service/Action Concept

The Service/Action model is conceptually similar to a Web Services model where an implementation provides one or more high-level services, each of which has one or more applicable actions (these equate to methods available on services). When used in conjunction with Request/Response message pairs, the model is conceptually similar to an RPC[11], though in fact the underlying messaging substrate need not rely on RPCs.

EbXML provides for the definition of S*ervice* and *Action* as placeholders for information pertaining to the intended processing of an ebXML message. From the perspective of the ebXML message service these values are merely passed through and they are provided to allow parties to target their messages for processing or action on particular services within their application systems. *Service* and *Action* take the form of REQUIRED elements within the ebXML message header.

When operating over an ebXML substrate, OTA REQUIRES the use of defined values for *Service* and *Action* elements and the values defined are analogous in concept to Web Services and methods available on a particular web service (think of the ebXML Service being conceptually a Web Service and the ebXML Action being analogous to a method within that web service).

3.1.1 Service/Action Message Mappings

Within the ebXML Service element there is an optional *type* attribute which parties may use in the interpretation of the meaning of service. Within OTA messages flowing over ebXML, the value of the type attribute SHALL be “OTA”.

The following fragment illustrates the context in which *Service* and *Action* will appear within a *MessageHeader*:

Example 1 - MessageHeader showing a read operation[12] on the AirBooking service

```
<eb:MessageHeader id="…" eb:version="1.0" SOAP-ENV:mustUnderstand="1">
    <eb:From>
        <eb:PartyId eb:type=”urn:duns”>008925431</eb:PartyId>
    </eb:From>
```

[11] Remote Procedure Call – a synchronous, blocking request/response message exchange across an underlying network. RPCs are popular as a model among application programmers as they offer complete encapsulation of underlying network and messaging infrastructure.

OTA’s infrastructure specifies high-volume messaging with different classes of delivery. This substrate allows for multiple outstanding requests on a single session and offers considerable performance advantages over RPC based infrastructure.

[12] Note the use of one of the generic message – OTA_ReadRQ as an action within the AirBooking service


```
    <eb:To>
        <eb:PartyId eb:type="urn:duns">008296571</eb:PartyId>
    </eb:To>
    <eb:CPAId>http://opentravel.org/cpa/2001C/defaultcpp.xml</eb:CPAId>
    <eb:ConversationId>987654321-123456789</eb:ConversationId>
    <eb:Service eb:type="OTA">AirBooking</eb:Service>
    <eb:Action>OTA_ReadRQ</eb:Action>
    <eb:MessageData>
        <eb:MessageId>mid:UUID-2</eb:MessageId>
        <eb:Timestamp>2001-10-25T12:19:05Z</eb:Timestamp>
    </eb:MessageData>
    <eb:QualityOfServiceInfo eb:deliverySemantics=”BestEffort”/>
</eb:MessageHeader>
```

As such, the following Services are defined within OTA[13]:

OTA Service	*Description*
Profile	This service provides operations on customer profiles
VehicleBooking	Service to check availability, book, modify and/or cancel vehicle rentals
AirBooking	Service to check air/flight availability and for air/flight reservation/booking
TravelInsurance	Travel Insurance related service
HotelBooking	Service to search for and identify hotels, check availability, book, modify and/or cancel hotel accommodations
HotelResNotification	Service for delivery of hotel bookings between systems (e.g. between central systems and property-based systems)
HotelPropertyInformation	Services for actions pertaining to detailed per-stay accommodation statistics, agency commission reports and property related statistics of interest to 3rd party systems such as revenue-management systems
MeetingProfile	Service to create/modify meeting profiles for group/convention related business
PackageBooking	Service to check availability, book/modify/cancel holiday/tour packages
Session	OTA Infrastructure service used to establish, and/or terminate sessions
GolfTeeTimes	Service for locating a golf course, checking availability and booking tee times

3.2 Unique Identifiers within OTA Messages

Each record that identifies a unique business document containing travel-related information, such as a profile or reservation record, MUST have a unique identifier assigned by the system

[13] Working groups should define additional services as new message sets are defined. See section 6 for detail mappings of messages onto specific actions on these services.

that creates it with the tag name <UniqueId>[14]. The unique identifier on the record MUST contain both a *Type* and an *Id* attribute. It MAY optionally include a *URL* and an *Instance* attribute. The syntax for a <UniqueId> is formally defined in the following Schema fragment:

Schema 1 - <UniqueID>:

```
<xs:element name="UniqueId">
    <xs:complexType>
        <xs:attribute name="URL" type="xs:string"/>
        <xs:attribute name="Type" type="xs:string" use="required"/>
        <xs:attribute name="Id" type="xs:ID" use="required"/>
        <xs:attribute name="Instance" type="xs:string"/>
    </xs:complexType>
</xs:element>
```

- *URL* - This optional attribute is what makes a <UniqueId> instance globally unique outside the context of a single bilateral conversation between known trading partners. OTA RECOMMENDS that the URL be a reference to the public OTA implementation for each trading partner. ***Note***: In the absence of having a public URL, the reference for this attribute could be determined by bilateral agreement.
- *Type* - This enumerated attribute references the type of object this <UniqueId> refers to, and gives this element its generality. By convention, the *Type* attribute value is the same as the OTA element tag name for the referenced object, for example, "Profile" or “Reservation”. As additional message types are defined in future versions of OTA specifications, the *Type* attribute enumeration will expand to include additional tag names values where a <UniqueId> applies.
- *Id* - This represents a unique identifying value assigned by the creating system, using the XML data type ID. The *Id* attribute might, for example, reference a primary-key value within a database behind the creating system’s implementation.
- *Instance* - This optional attribute represents the record as it exists at a point in time. An *Instance* is used in update messages where the sender must assure the server that the update sent refers to the most recent modification level of the object being updated. Every time the record changes *Instance* assumes a different value.

Possible implementation strategies for *Instance* values are:

- a timestamp
- a monotonically increasing sequence (incremented on each update)
- an md5 sum of the binary representation of the object in its persistent store

3.2.1 Examples of unique identifiers

A valid unique identifier MAY contain only the *Type* and *Id* (a unique string assigned by the system that created it) attributes:

```
<UniqueId Type="Profile" Id="1234567"/>
```

To ensure that a unique identifier is globally unique (in the universal namespace) add a *URL* attribute which includes a fully-qualified domain-name:

[14] OTA <UniqueId> elements are not related to the unique identifiers supplied within an ebXML MessageHeader i.e. the eb:MessageId is generated by an ebMS and refers to the ebXML message as a whole whereas an OTA UniqueId is generated within application space and uniquely identifies a business object, having the same lifecycle as that business object.


```
<UniqueId URL="http://vmguys.com/OTAengine/" Type="Profile" Id="1234567"/>
```

This *Id* is assured of being globally unique in any namespace as the URL points to a vendor's OTA implementation which, in turn, relies on the unique domain name for the vendor assigned by a government approved Internet Domain Name registrar. OTA has considered the potential effect a name change would have on globally unique identifiers, and noted that a change in URL or primary domain name should not present an issue unless another business entity assumes the previous URL unaltered.

This approach to unique naming takes into consideration the following benefits:

- provides a simple and succinct representation
- guaranteed to be a globally unique identifier within the universal namespace (with the use of the URL attribute)
- becomes applicable and reusable in other OTA specifications

3.3 Generic Infrastructure Messages

Defining certain actions at the infrastructure level allows for reuse on multiple services and avoids duplication of work between domain-specific working groups. Generic infrastructure messages also offer opportunities for software reuse within implementations. The basic operations include Create, Read, Update, and Delete – memorably abbreviated CRUD. These four verbs provide consistent conventions for basic actions affecting both infrastructure and business elements in OTA specifications.

The Create, Read, and Delete actions MUST apply only to entire records. Updates allow for addressing one or more individual elements, and making changes to part(s) of a record.

3.3.1 Create messages

Create messages define an operation that generates a new record with a unique identifier. The sequence follows these steps:

- Requestor sends a Create request along with the initial data, and optionally a unique identifier.
- Receiver creates a new record and assigns a unique identifier (e.g. a Profile Id or Reservation Id).
- Receiver responds with a message providing a unique identifier for the new record created and optionally, any data entered by the requestor.

Example 2 – Create request message:[15]

```
<OTA_CreateProfileRQ Version=”2”>
    <UniqueId Type="Profile"/>
    <Profile>
        <Customer Gender="Male">
            <PersonName>
                <GivenName>John</GivenName>
                <Surname>Smith</Surname>
            </PersonName>
                .....
        </Customer>
```

[15] We thank Adam Athimuthu for creating the sample “Create" messages.


```
    </Profile>
</OTA_CreateProfileRQ>
```

Example 3 - Create response message:

```
<OTA_CreateProfileRS Version="2">
    <Success />
    <UniqueId
        URL="http://www.sixcontinents.com/OTAProcessor"
        Type="Profile"
        Id="1234567"/>
</OTA_CreateProfileRS>
```

There is no generic OTA message for create. Each working group should define new *Create* messages for their business objects using the patterns outlined above.

3.3.2 Generic Read message

The Read infrastructure action defines an operation that opens an existing record and transmits information contained in that record. The Read operation enables the user to identify a particular record and retrieve its entire contents. The basic operation has the following steps:

- Requestor queries the database where the record resides by sending a Read request message with the object's unique identifier
- Receiver returns the record to the requestor

The use of the OTA <UniqueId> element allows for a generalized read transaction message. With the object type specified via the *Type* attribute, the action type is identified within a general read request.

The use of the OPTIONAL OTA <POS> element allows an implementation to determine whether the remote user has permission to view the object being read.

Example 4 - OTA ReadRQ message:

```
    <OTA_ReadRQ ReqRespVersion="2">
        <UniqueId
            URL="http://vmguys.com/OTAEngine/"
            Type="Profile"
            Id="0507-12345"/>
    </OTA_ReadRQ>
```

ReqRespVersion - The optional "Request Response Version" attribute allows the sender to indicate the version desired for the response message. For example, the OTA_ReadRQ message sample above indicates a request to return an OTA Customer Profile.

This request applies to all types of objects, not just profiles. The type of the generated response depends, of course, on the *Type* specified in the request; for example, the <OTA_ReadRQ> shown in the example would generate a <OTA_ReadProfileRS> message that contains an OTA Profile as a response, as in the example below.[16]

This generalization significantly reduces the maintenance burden for individual infrastructure verbs, with effectively zero loss in semantics.

Example 5 - Read response message:

```
    <OTA_ReadProfileRS Version="2">
        <UniqueId
```

[16] Expected responses for each request type are formally defined in section 6: Service/Action mappings.


```
            URL="http://vmguys.com/OTAEngine/"
            Type="Profile"
            Id="0507-12345"
            Instance="20000531172200"/>
        <Profile>
            ...
            ...
        </Profile>
    </OTA_ReadProfileRS>
```

An *Instance* value returned in a Read response, if not implemented as a timestamp, may specify the same instance value to all requestors until the record is changed by a subsequent action to the record, such as an update.

The generic *OTA_ReadRQ* message is formally defined by the following schema fragment:

Schema 2 - <OTA_ReadRQ>:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created and edited with 'vi' -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns="http://www.opentravel.org/OTA"
            targetNamespace="http://www.opentravel.org/OTA"
             elementFormDefault="qualified">

    <xs:annotation>
        <xs:documentation xml:lang="en">
OTA v2001C Specification - Generic OTA_ReadRQ message definition
Copyright (C) 2001 Open Travel Alliance. All rights reserved.
        </xs:documentation>
    </xs:annotation>
    <xs:include schemaLocation="OTA_v2ent.xsd"/>
    <xs:include schemaLocation=”OTA_POS.xsd”/>

    <xs:element name="OTA_ReadRQ">
        <xs:annotation>
            <xs:documentation xml:lang="en">
A generic message, available as an action on several OTA services
which requests a server to read and return the document type
identified by the UniqueId element.
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="UniqueId"/>
                <xs:element ref=”POS” minOccurs=”0”/>
            </xs:sequence>
            <xs:attributeGroup ref="OTA_PayloadStdAttributes"/>
            <xs:attributeGroup ref="ReqRespVersion"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

3.3.3 Generic Update message

The Update infrastructure action defines an operation that opens an existing record, identifies the information that needs changing, then transmits data corresponding to the appropriate elements in the tree, and adds or replaces those data in the record.

Because Update operations are more complex and can affect parts of the record rather than the entire record, handling update messages generally can be more difficult. As a result, two approaches to updating records are defined in this specification.

The goals considered in the design of the Update operation include:

- Minimizing the size of a payload on the wire to represent an update transaction

- Defining an explicit representation about what has changed
- Defining a representation with a clear and simple conceptual model
- Creating a representation that is content-independent and general-purpose in nature so as to be reusable throughout future OTA specifications
- Providing a simple-to-implement "replace" option to allow developers to get simpler implementations running quickly - at the expense of the first 2 goals (representation of change and size of message) above

Because data to be modified may be stored in a database and not in an XML document format, it may not be possible to reconstruct the original document that transmitted the data. Therefore, it is RECOMMENDED that implementations utilizing the partial update process perform a Read request to obtain the structure of the XML tree prior to constructing an Update request.

The definition of the Update message is lengthy. An example update request for a customer profile modification is shown below. A detailed treatment of update requests, how to generate and approaches to processing them is included in an appendix (see section 5 – OTA Update Messages).

Example 6 - OTA_UpdateRQ[17]:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.2 (http://www.vmguys.com/vmtools/) -->
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
               ReqRespVersion="2">
    <UniqueId Type="Profile"
              Id="987654321"
               URL="http://www.vmguys.com/OTAEngine/"
               Instance="20011012160659" />
    <Position XPath="/Profile/Customer/AddressInfo/Address/StreetNmbr">
        <Attribute Name="PO_Box" Operation="delete" />
    </Position>
    <Position XPath="/Profile/Customer/TelephoneInfo">
        <Attribute Name="PhoneUse" Operation="modify" Value="Home" />
    </Position>
    <Position XPath="/Profile/Customer/PersonName/MiddleName">
        <Subtree Operation="delete" />
    </Position>
    <Position XPath="/Profile/Customer">
        <Subtree Operation="insert" Child="5">
            <RelatedTraveler Relation="Child">
                <PersonName>
                    <NamePrefixNamePrefix>Ms.</NamePrefixNamePrefix>
                    <GivenName>Amy</GivenName>
                    <MiddleName>E.</MiddleName>
                    <Surname>Smith</Surname>
                </PersonName>
            </RelatedTraveler>
        </Subtree>
        <Attribute Name="Gender" Operation="modify" Value="Male" />
    </Position>
</OTA_UpdateRQ>
```

3.3.4 Generic Delete message

[17] Additional examples, including before and after images, may be found in section 5 – OTA Update messages

The Delete infrastructure action defines an operation that identifies an existing record, and removes the entire record from the database. The use of the Delete action depends upon the business rules of an organization. Alternative strategies, such as mapping a duplicate record to another by use of the UniqueId, may be considered.

The requestor MAY also verify the record before deleting it to ensure the correct record has been identified prior to deleting it. In this case, the use of the *Instance* attribute may be useful in determining whether the record has been updated more recently than the information that is intended to be deleted. That choice, again, would be dictated by good business practices.

Steps in the Delete operation include:

- Requestor submits a Read request to view the record
- Receiver returns the record for the requestor to view
- Requestor submits a Delete request.
- Receiver removes the record and returns an acknowledgement

The use of the OPTIONAL OTA <POS> element allows an implementation to determine whether the remote user has permission to delete the object being read.

Example 7 - illustrating the Delete process, Read request:

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_ReadRQ ReqRespVersion="2">
    <UniqueId>
        URL=http://vmguys.com/OTAEngine/
        Type="Profile"
        Id="0507-12345"
    </UniqueId>
</OTA_ReadRQ>
```

Example 8 - illustrating the Delete process Read response:

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_ReadProfileRS Version=”2”>
    <UniqueId>
        URL="http://vmguys.com/OTAEngine/"
        Type="Profile"
        Id="0507-12345"
        Instance="2"
    </UniqueId>
    <Profile>
    ...
    </Profile>
</OTA_ReadProfileRS>
```

Example 9 - illustrating the Delete request:

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_DeleteRQ ReqRespVersion="2">
    <UniqueId>
        URL="http://vmguys.com/OTAEngine/"
        Type="Profile"
        Id="0507-12345"
        Instance="2"
    </UniqueId>
</OTA_DeleteRQ>
```

Example 10 - illustrating the Delete response:

```
<?xml version="1.0" encoding="UTF-8"?>
<OTA_DeleteRS Version=”2”>
    <UniqueId>
        URL="http://vmguys.org/OTAEngine/"
        Type="Profile"
        Id="0507-12345"
```



```
        Instance="0"/>
    </UniqueId>
    <Success/>
</OTA_DeleteRS>
```

A generic OTA delete message is formally defined by the following schema fragment:

Schema 3 - <OTA_DeleteRQ>:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created and edited with 'vi' -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.opentravel.org/OTA"
           targetNamespace="http://www.opentravel.org/OTA"
            elementFormDefault="qualified">

    <xs:annotation>
        <xs:documentation xml:lang="en">
OTA v2001C Specification - Generic OTA_DeleteRQ message definition
Copyright (C) 2001 Open Travel Alliance. All rights reserved.
        </xs:documentation>
    </xs:annotation>
    <xs:include schemaLocation="OTA_v2ent.xsd"/>
    <xs:include schemaLocation”OTA_POS.xsd”/>

    <xs:element name="OTA_DeleteRQ">
        <xs:annotation>
            <xs:documentation xml:lang="en">
A generic message, available as an action on several OTA services
which requests a server to delete the business object
identified by the UniqueId element.
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="UniqueId" />
                <xs:element ref=”POS” minOccurs=”0” />
            </xs:sequence>
            <xs:attributeGroup ref="OTA_PayloadStdAttributes"/>
            <xs:attributeGroup ref="ReqRespVersion"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

A response to a delete request follows the normal pattern for OTA responses and is formally defined by the following schema fragment:

Schema 4 - <OTA_DeleteRS>:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created and edited with 'vi' -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns="http://www.opentravel.org/OTA"
            targetNamespace="http://www.opentravel.org/OTA"
           elementFormDefault="qualified">

    <xs:annotation>
        <xs:documentation xml:lang="en">
OTA v2001C Specification - Generic OTA_DeleteRS message definition
Copyright (C) 2001 Open Travel Alliance. All rights reserved.
        </xs:documentation>
    </xs:annotation>
    <xs:include schemaLocation="OTA_v2ent.xsd"/>

    <xs:element name="OTA_DeleteRS">
        <xs:annotation>
            <xs:documentation xml:lang="en">
Response to a generic OTA_DeleteRQ message, available as an action
```

```
on several OTA services which requests a server to delete the business
object identified by the UniqueId element.
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:choice>
                <xs:sequence>
                    <xs:element ref="Success" />
                    <xs:element ref="Warnings" minOccurs="0" />
                    <xs:element ref="UniqueId" />
                </xs:sequence>
                <xs:element ref="Errors" />
            </xs:choice>
            <xs:attributeGroup ref="OTA_PayloadStdAttributes"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

3.3.5 Generic Cancel Request

OTA has developed a generic cancel message for use within the travel industry because there are common patterns between industry verticals for canceling a reservation, and it is anticipated that the messages defined here would be used to cancel a reservation made using the specifications in this publication.

The basic pattern for all industry verticals is to identify the reservation by some form of unique identification number, whether that number is called a Record Locator Number, PNR, Reservation ID, Confirmation ID, or called by some another name. While the nomenclature may differ from system to system, the convention is universal: 1) identify the ID unique to the system assigned to perform the cancellation, and 2) receive a confirmation of the cancellation as a response.

If a travel service is not going to be fulfilled, the availability of inventory is affected, and the system holding the inventory will want to release that inventory. Typically, the cancel action is executed by the receiving system, as it is usually the system that holds the reserved inventory. The cancellation allows the inventory to be returned back to the marketplace and enables the supplier to resell it as quickly as possible.

A cancellation could conceptually be considered as a special case of a "modify" transaction because the record of the reservation is generally not removed immediately from a system, but placed into a different status. However, a cancel transaction differs from a modification because the entirety of the services booked will not be consumed. The information exchange differs because the system doing the cancellation does not have to return the bulk of the reservation information as is the case when confirming the modification of reservation information. The process becomes much simpler: send the unique identifier to the system with the indication that the action to be taken is to perform a cancel; receive a response that the action has taken place.

A cancellation also differs from the generic infrastructure verb, OTA_DeleteRQ. The Delete action removes the record out of the database, while in a cancellation, the record of the reservation is not necessarily deleted. A cancellation is a business process driven by the business rules applied to the reservation, and effectively provides a change of status of that reservation.

3.3.5.1 Consequences of Canceling

When a cancel message is sent, two possibilities exist: the reservation may be canceled without penalty, or the cancellation incurs a penalty for doing so. For many travel services, perhaps the majority in this day and age of bargains, promotional fares and special rates, some sort of restrictions may apply. A cancellation of travel services may result in forfeiture of an amount paid for a guarantee, or deposit, etc.

3.3.5.2 Read Request Prior to Cancel

The logical steps that occur for a cancellation recognize that prior to sending a cancellation message, the party holding the reservation may wish to retrieve and review that reservation. Reasons for doing so include reviewing associated information that may communicate the cancellation policy, or simply to confirm the identification of the reservation to be cancelled, among a series of reservations held by that party.

The function of retrieving the reservation uses the generic OTA_ReadRQ, specifying the UniqueId of the reservation, with `Type="Reservation"`, and optionally supplying a URL to identify the party holding the reservation or the machine location where the reservation is being stored.

Example 11 - <OTA_ReadRQ> message:

```
    <OTA_ReadRQ ReqRespVersion="1">
        <UniqueId
            URL="http://provider1.com/OTAEngine/"
            Type="Reservation"
            Id="05071G4325"/>
    </OTA_ReadRQ>
```

The response is a specific OTA Read message, containing the reservation information.

Example 12 - <OTA_ReadHotelResRS> message:

```
<OTA_ReadHotelResRS Version="1">
    <UniqueId
        URL="http://vmguys.com/OTAEngine/"
        Type="Reservation"
        Id="05071G4325"
        Instance="20000531172200"/>
    <HotelRes>
        ...
        ...
    </HotelRes>
</OTA_ReadHotelResRS>
```

An optional *Instance* value returned in a Read response, if implemented as a timestamp, may indicate the most recent record of the reservation and allow for synchronization if the requesting and receiving party do not hold the same reservation.

3.3.5.3 Security considerations

Inherent in the business practices of companies exchanging information is the requirement for some kind of qualification that entitles the requestor to receive the reservation information. Sufficient information should be sent for the receiving system to be able to recognize the requesting system, and to qualify it to receive the information.

Systems may assume a point-to-point connection upon receiving a request to retrieve a booking, but may still need to know who is on the terminal, particularly with transactions that may come from travel web sites where the individual has the opportunity to log on and control their own reservation. The booking engine, or application processing the request, is tasked to qualify the requestor so that it can be certain that the party is who they represent themselves to be. This usually involves furnishing some kind of information such as a last name, membership number, confirmation number, or credit card number (at least the last 4 digits of the credit card number) that was used for the reservation. That level of verification may be required even with the identification of the source and booking channel supplied by the Point-of-Sale information.

This generic cancel message assumes that the software asking for the read or cancel action has pre-determined the other party's rights for viewing at either end of the message conversation. The same considerations for the security of the connection, as well as identification of the requesting party, is true for transactions involving reading and modifying a profile.

It is common practice for many systems to keep a summary level view of a reservation, retaining information about the requestor. Therefore, within the remit of information in the request, some level of security can be applied to view or cancel that information. In many cases, sending the transaction to retrieve a reservation must be made through the channel or source where the original booking took place.

Complications arise when, within a booking or channel, all of the other qualifiers might be right, but the request is not made through the same source. For example, a travel agency branch office could be handling a request for a customer who made the original reservation through another location. In that case, the retrieving application is then tasked to take the qualification to the next level and match the request up to the channel and source to display only those reservations that were booked through that agency.

Travel suppliers may wish to support returning all reservations that can be identified with their company, e.g. using identifiable confirmation numbers, regardless of where the reservation was booked (e.g. through their Central Reservation System, or through a web site, etc.). Conversely, the customer who had booked a reservation through a specific web site, could not go on another travel supplier web site, and retrieve a booking made on the competitor's site. The reservation could only be retrieved through the original source.

Note: It is likely that within one company, systems will be able to use the generic Read or Cancel request for conversations between trusted sources, such as an interface from their web site to a legacy system. Additional information may be needed to establish a level of confidence between trading partners. As the nature of XML is to be extensible, partners wishing to expand upon requirements in their environment may use the `<TPA_Extension>` defined elsewhere in this document.[18]

The use of the OPTIONAL OTA `<POS>` element allows an implementation to determine whether the remote user has permission to cancel a particular booking.

3.3.5.4 OTA Cancel Messages

This specification provides a request/response pair of messages to support the functionality of canceling a reservation.

The following message pair is used on all applicable services to cancel a reservation:

- **OTA_CancelRQ** - Identifies the reservation and requests a cancellation.
- **OTA_CancelRS** – MAY return a list of rules that govern the cancellation, a cancellation number upon execution of the cancel action, or Warnings or Errors if the processing of the request did not succeed.

3.3.5.4.1 OTA Cancel Request

The root element of the OTA_CancelRQ contains the standard payload attributes found in all OTA payload documents as well as the attribute *ReqRespVersion=* that requests a specific version

[18] See section 2.3.8.2 Best Practices Guideline VIII-2 for details and examples of <TPA_Extension>

of the response message. As this is the first publication of the OTA cancel message set, currently the only valid value is "1".

The cancel request also has an attribute, `CancelType = "  "`, that defines the action requested in the cancel message.

Attributes of OTA_CancelRQ are as follows:

- ***OTA_PayloadStdAttributes*** - includes the 5 standard attributes on all OTA messages.
- ***ReqRespVersion*** - Requests a version of the response message.
- ***CancelType*** - An enumerated type indicating the type of request made for the cancellation. Valid Values are: (Initiate | Ignore | Confirm).

  Initiate - Indicates the initial request to cancel a reservation.

  Ignore - Indicates a roll-back of the request to cancel, leaving the reservation intact.

  Confirm - Indicates a request to complete the cancellation.

3.3.5.4.2 Cancel Request

The cancel request is formally defined by the following schema fragment:

Schema 5 - <OTA_CancelRQ> message:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created and edited with 'vi' -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns="http://www.opentravel.org/OTA"
            targetNamespace="http://www.opentravel.org/OTA"
           elementFormDefault="qualified">

    <xs:annotation>
        <xs:documentation xml:lang="en">
OTA v2001C Specification - Generic OTA_CancelRQ message definition
Copyright (C) 2001 Open Travel Alliance. All rights reserved.
        </xs:documentation>
    </xs:annotation>
    <xs:include schemaLocation="OTA_v2ent.xsd"/>
    <xs:include schemaLocation=”OTA_POS.xsd”/>

    <xs:element name="OTA_CancelRQ">
        <xs:annotation>
            <xs:documentation xml:lang="en">
A generic message, available as an action on several OTA services which
requests a server to cancel the booking identified by the UniqueId element.
            </xs:documentation>
        </xs:annotation>

        <xs:complexType>
            <xs:sequence>
                <xs:element ref="UniqueId"/>
                <xs:element ref=”POS” minOccurs=”0”/>
            </xs:sequence>
            <xs:attributeGroup ref="OTA_PayloadStdAttributes"/>
            <xs:attributeGroup ref="ReqRespVersion"/>
            <xs:attribute name="CancelType" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="Initiate"/>
                        <xs:enumeration value="Ignore"/>
                        <xs:enumeration value="Confirm"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
```

```
        </xs:complexType>
    </xs:element>
</xs:schema>
```

3.3.5.4.3 Cancel Request - Sample XML message

The following sample message is an example of an initial cancel message:

```
<OTA_CancelRQ xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:noNamespaceSchemaLocation="OTA_CancelRQ.xsd"
              Version="1" ReqRespVersion="1"
              CancelType="Initiate">
    <UniqueId  URL=" http://provider1.org/OTAEngine/"
              Type="Reservation"
              Id="0507G4325"
              Instance="2001-06-03T13:09:21"/>
</OTA_CancelRQ>
```

3.3.5.5 Confirmation of Cancellation

When a cancellation has been executed, the majority of systems will return a cancellation number. The identification number supplied serves to confirm that the action has taken place, and makes it possible to track monetary ramifications that may be associated with the cancellation.

A single cancel request can result in a transaction that straightforwardly cancels a reservation and returns a confirmation that the cancellation has taken place. Alternatively, the request to cancel a reservation may violate a business rule, subjecting the party to a cancellation penalty. In that case, it is desirable to return information about the cancellation penalty in the response.

The OTA_CancelRQ message allows for the possibility to perform a two-phase transaction to inform the party wishing to cancel of the penalties that would be incurred, and to allow them to verify their intention to cancel. The CancelType attribute is a flag that can be set to the choices of (Initiate | Ignore | Confirm). This enables the message conversation to send in an initial request to cancel a reservation, anticipate a response that returns the consequences of doing so, and allows for the option of rolling back the transaction and choosing NOT to cancel the reservation.

3.3.5.5.1 OTA Cancel Response

The `<OTA_CancelRS>` message follows the standard design pattern for response messages.[19] Additionally, it MAY return a collection of cancellation rules, with penalty amounts, if incurred, and an indication of the status of the cancellation request, either "Cancelled", "Pending", or "Ignored" if the transaction has been rolled back and the reservation remains intact.

Attributes of OTA_CancelRS are as follows:

- ***OTA_PayloadStdAttributes*** - includes the 5 standard attributes on all OTA messages.
- ***CancelID*** - The identification number of the cancellation.
- ***Status*** - An enumerated type indicating the status of the cancellation request. Valid Values are: (Pending | Ignored | Canceled).

  Pending - Indicates the initial request to cancel a reservation is pending confirmation to complete the cancel action. Cancel rules may have been returned along with the response.

  Ignored - Indicates the request to cancel was rolled back, leaving the reservation intact.

[19] See sectin 2.4.2 “Response Message Design Patterns” for details

Canceled - Indicates the cancellation is complete. A cancellation ID may have been returned along with the response.

By providing the option of a two-step process, individual business rules may determine how their system processes the initial request. If there are no penalties involved in the cancellation, the cancel transaction can take place and the response return the cancellation number along with the status that the reservation has been cancelled.

If the processing system determines that a cancellation policy has been invoked, it may choose to send back the OTA_CancelRS with the Status="*Pending*", accompanied by a collection of cancellation rules, allowing the originating party to determine if the cancellation should proceed. The originating system would then resend the OTA_CancelRQ. A CancelType="*Ignore*" would anticipate a response with the Status "*Ignored*", thus ending the message conversation with no action being taken to cancel the reservation.

A CancelType = "*Commit*" indicates a definitive "Yes" to process the cancellation. This message would anticipate the response of Status="*Cancelled*", along with the return of a Cancellation Id, and that transaction would complete the cancellation process. The cancel RQ is the same message in each case, with the CancelType attribute indicating the action to be taken on the request.

3.3.5.5.2 Cancel Rules

The <OTA_CancelRS> message has two child elements: the reservation record is identified by the use if the UniqueId element, and the cancellation rules and/ or penalties are communicated by the use of the CancelRules collection that may return one or many CancelRule elements.

The attributes of the CancelRule element are as follows:

- ***CancelByDate*** - Identifies the date by which a cancellation should be made in order to avoid incurring a penalty.
- ***Amount*** - The monetary amount incurred as a result of the cancellation.
- ***CurrencyCode*** - The code that identifies the currency of the penalty amount, using ISO 4217.

The data type of the CancelRule element is *xs:string*, which can be used for text describing the cancellation penalties. Cancel Rule is a repeating element and can be used as many times as needed to communicate the cancel penalties and rules

3.3.5.5.3 Cancel Response

The Cancel response is formally defined by the following schema fragment:

Schema 6 - <OTA_CancelRS> message:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created and edited with 'vi' -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.opentravel.org/OTA"
           targetNamespace="http://www.opentravel.org/OTA"
            elementFormDefault="qualified">

    <xs:annotation>
        <xs:documentation xml:lang="en">
OTA v2001C Specification - Generic OTA_CancelRS message definition
Copyright (C) 2001 Open Travel Alliance. All rights reserved.
        </xs:documentation>
    </xs:annotation>
    <xs:include schemaLocation="OTA_v2ent.xsd"/>

    <xs:element name="OTA_CancelRS">
```

```
        <xs:complexType>
            <xs:choice>
                <xs:sequence>
                    <xs:element ref="Success" />
                    <xs:element ref="Warnings" minOccurs="0" />
                    <xs:element ref="UniqueId" />
                    <xs:element ref="CancelRules"/>
                </xs:sequence>
                <xs:element ref="Errors" />
            </xs:choice>
            <xs:attributeGroup ref="OTA_PayloadStdAttributes" />
            <xs:attribute name="Status" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="Pending"/>
                        <xs:enumeration value="Ignored"/>
                        <xs:enumeration value="Canceled"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="CancelId" type="xs:string"/>
        </xs:complexType>
    </xs:element>

    <xs:element name="CancelRules">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="CancelRule" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="CancelRule">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:string">
                    <xs:attribute name="CancelByDate" type="xs:string"/>
                    <xs:attribute name="Amount" type="xs:string"/>
                    <xs:attribute name="CurrencyCode" type="xs:string"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

3.3.5.5.4 Cancel Response - Sample XML message

The following sample message is an example of a cancel response message:

```
<OTA_CancelRS xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
              xmlns=”http://www.opentravel.org/OTA”
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA_CancelRS.xsd"
              Version="2"
              Status="Canceled"
              CancelId="1236597GQ345">
    <Success/>
    <UniqueId URL="http://provider1.org/OTAEngine/"
             Type=”Reservation"
             Id="0507G4325"
             Instance="2001-06-03T13:09:21"/>
    <CancelRules>
        <CancelRule CancelByDate="2001-05-31T13:19:42-05:00"
                    Amount="75.00"
                    CurrencyCode="USD">
Cancellations within 30 days incur a penalty of $75.00</CancelRule>
    </CancelRules>
</OTA_CancelRS>
```


4 OTA Infrastructure

In this section we explore the revised OTA infrastructure. See section 7 for a list of changes since the last publication of the infastructure specification in 2001A.

4.1 Architecture Overview

OTA's Infrastructure Architecture borrows heavily from ebXML's Technical Architecture[20] [ebTA] and Message Service[21] [ebMS]. EbXML's Message Service, which is based on SOAP version 1.1 and the Soap with Attachments informational document, provides the functionality needed for two or more parties to engage in an "electronic business transaction". Products that implement the ebMS specification should be capable of the reliable and secure exchange of business data associated with a "business transaction". It is expected that any company implementing an OTA compliant system will be capable of supporting synchronous and asynchronous processing models as required by the various OTA message types described elsewhere in this document

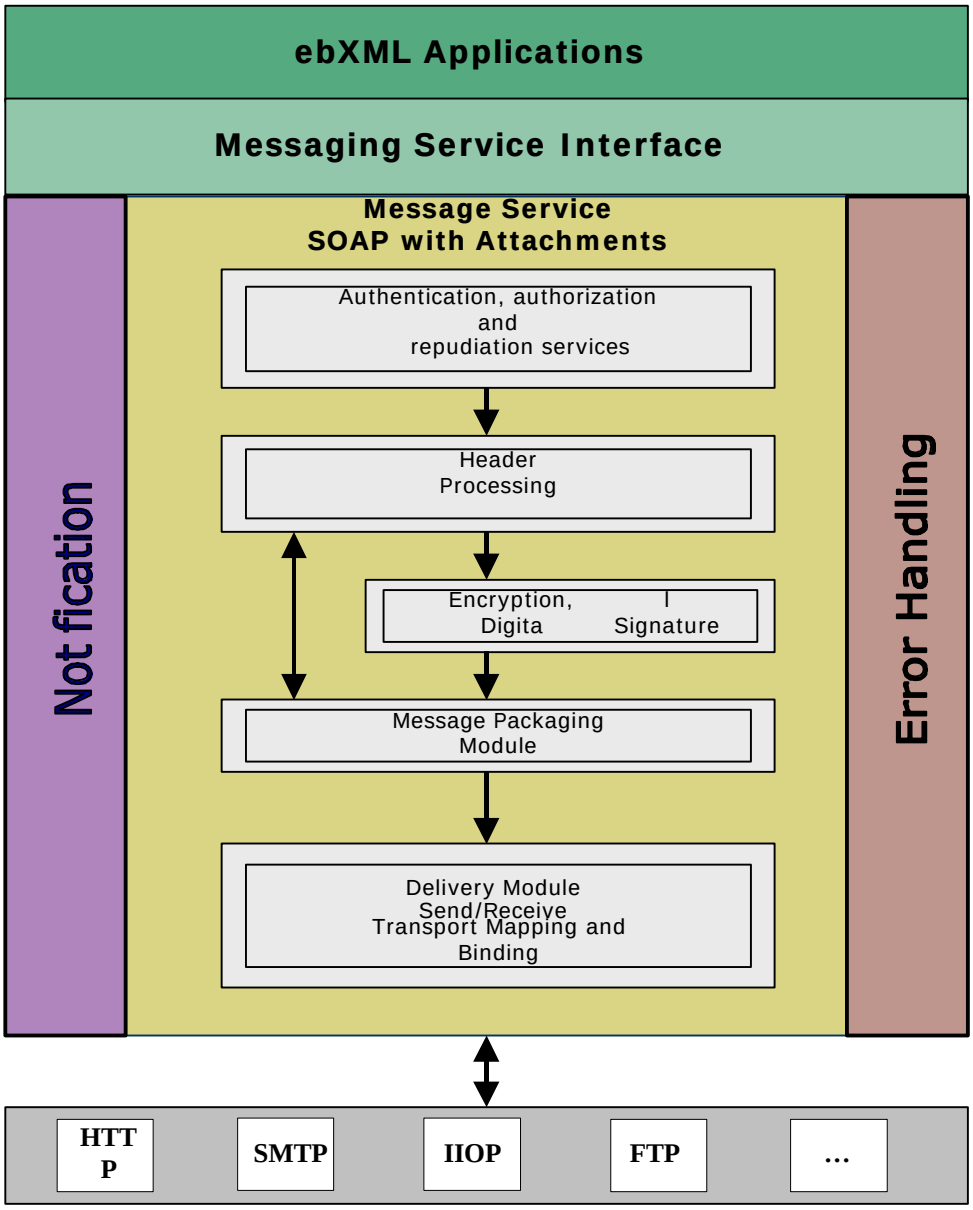
Business Applications interface with an ebMS through a "service interface", which is unique to each ebMS product. The "service interface" allows applications to request services and receive notifications from an ebMS product. When an application needs to engage in an electronic business transaction with a trading partner it must call upon its local ebMS service, through the service interface, to establish a secure and reliable session with a trading partners ebMS/OTA system. Once a session is established the ebMS/OTA systems exchange information germane to the type of "service/action" being requested. A session may include multiple message exchanges between the ebMS/OTA systems of two or more parties.

. V2001 offers a typical developer working on an OTA-compliant application the option of using a standard ebMS rather than (as with v2001A) being compelled to design and implement the OTA infrastructure. It is the ebMS product vendor who is responsible for implementing the functionality described in the ebMS specification [ebMS]. A high quality ebMS implementation would also provide facilities for encrypting and digitally signing data, access control, authentication and authorization, real-time error notifications, logging, auditing and administrative tasks.

Following is an architectural diagram of an ebMS:

[20] See: http://www.ebxml.org/specs/ebTA.pdf

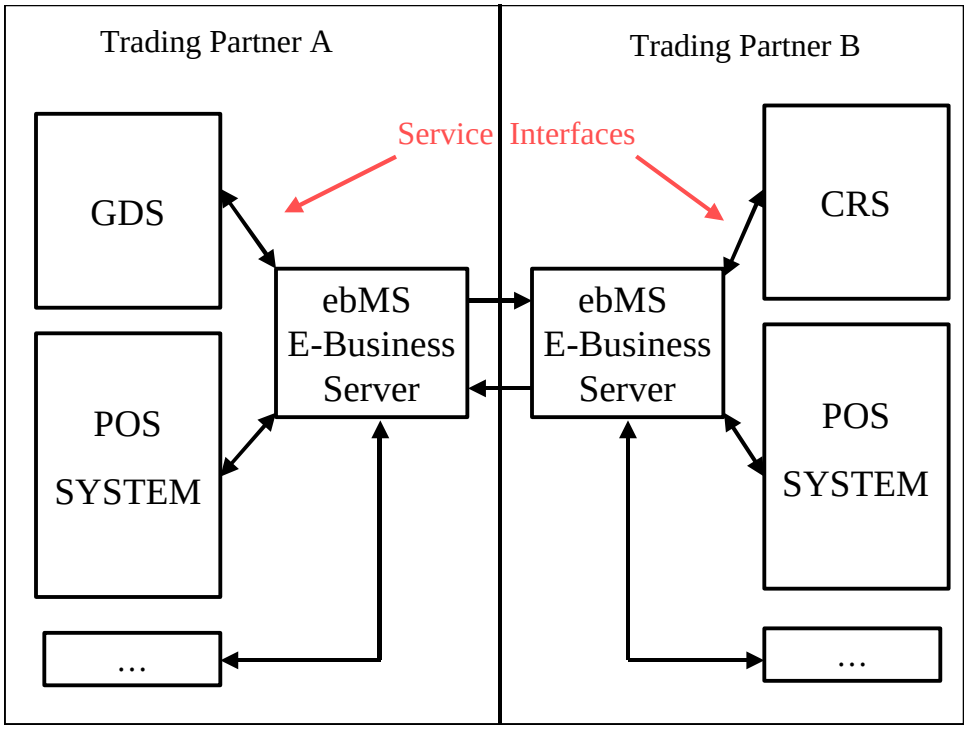
[21] See: http://www.ebxml.org/specs/ebMS.pdf



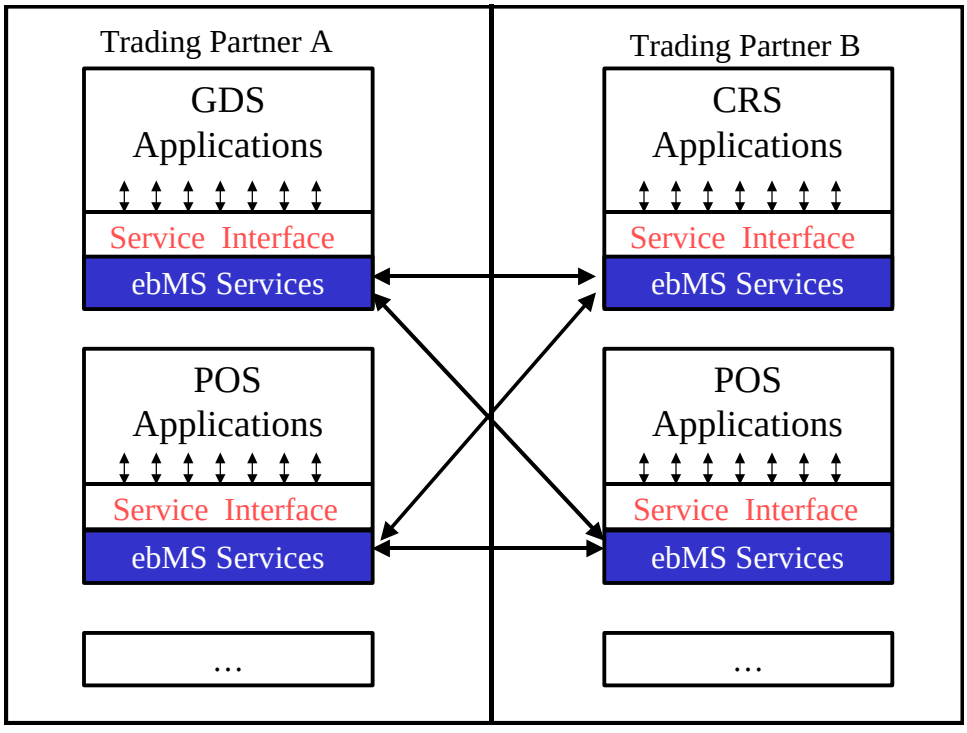
4.1.1 Reference Model

An ebMS may serve as a centralized "E-Business Server" to manage all of a company's electronic business transaction exchanges with trading partners. Alternatively ebMS functionality may be "integrated" within each business application that engages in electronic business transactions with trading partners. Regardless of which model is used ebMS implementers SHOULD provide the same service levels, access control, reliability, availability and security required by the electronic business transactions defined by OTA.

Below is a reference model depicting a centralized E-Business Server:



Below is a reference model depicting the integration of ebMS functionality within Business Applications:



4.1.2 Transport Protocols

The ebMS was designed to be transport neutral, however there are certain transport protocols that are more likely to be used for electronic transactions between trading partners over the Internet, one such protocol is HTTP (ref: RFC 2616).

EbMS/OTA implementers MUST support the HTTP protocol binding specifications defined in appendix B.2 of the ebMS specification. Additionally, ebMS/OTA implementers MUST support Basic Authentication (ref: RFC 2617) for access control, operating over a secure channel, using SSL Version 3.0 or TLS (ref: RFC 2246) with 128-bit key sizes for symmetric encryption algorithms. EbMS implementers MUST accept self signed digital certificates, as well as those of well-known Certificate Authorities (e.g. Verisign, Entrust, Thawte, et al), during the establishment of an SSL session.

OTA implementers SHOULD maintain ebMS systems that are available 24 hours per day, 7 days per week, to receive and process electronic business transactions from trading partners.

4.1.3 Logging

Organizations offering OTA transactions SHOULD provide logging capability, regardless of the type of transaction in the business message (e.g., travel verbs, infrastructure verbs), and trading partners MAY exchange event logs to provide audit trails.

Because of the lack of widely used standards or conventions for defining event logs, OTA does not require use of a specific log format, nor does the message architecture preclude any logging capability. Logging capabilities are expected to vary based upon the capabilities of an underlying ebXML message service implementation.

4.1.4 Auditing

EbMS product implementations SHOULD maintain audit logs that contain information used to track and correlate message exchanges between trading partners. The following information SHOULD be stored in an audit log:

- Date and time of entry
- Sender and Receiver IP addresses
- To/PartyId
- From/PartyId
- Status of the exchange (success/failure)
- MessageId
- RefToMessageId
- CPAId
- ConversationId
- Service
- Action
- Via
- QualityOfServiceInfo/syncreply value, if present
- Contents of Each Reference Element within a manifest
- Contents of Acknowledgement element, if present

- Contents of Error elements, if present

EbMS products SHOULD provide administrative functions to search and view logs.

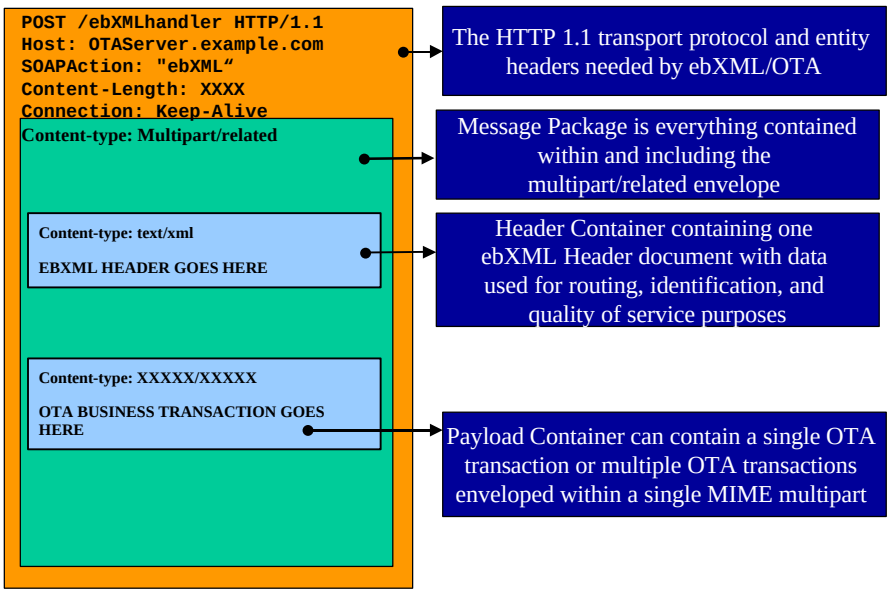
4.2 Message Structure and Packaging

An ebXML Message following OTA's conventions MUST contain one header container and zero or one payload container. Both containers are enveloped by a single MIME/Multipart envelope and the entire package is referred to as a *Message Package*.

The two containers within a *Message Package* are described below:

- The first MIME part, referred to as the *Header Container*, contains one ebXML header document. The header document contains only those elements and attributes required by OTA, the details of which are specified elsewhere in this document.
- The optional second MIME part, referred to as the *Payload Container*, contains application level payloads containing business data germane to the service/action identified in the ebXML header. A single payload container may contain one or more OTA business transactions.

The general structure and composition of an OTA compliant ebXML Message is described in the following figure.



4.2.1 The 'Unit-of-work' Concept

EbXML is extremely flexible in how messages can be packaged within payloads and in the absence of any guidelines implementers may choose substantially different approaches to packaging. As such OTA strongly recommends that all OTA message payloads within a given ebXML package pertain to one and only one 'unit of work'.

The concept of a unit of work is very similar to the concept of transactions in a transaction processing environment. An easy way to decide whether messages belong to the same unit of work is by applying the following tests:

- Do the messages fall within the boundary of a single transaction?
- Do the messages constitute items within the same batch?
- A single OTA message by definition matches the unit-of-work criteria

4.2.2 Packaging a single unit-of-work

The "scope" of a unit-of-work is highly dependent on the capabilities of the systems that engage in an electronic transaction. For example, a reservation system may be capable of performing a reservation for a single type of transaction (airline reservation), whereas another system may be capable of performing a reservation for an airline, automobile and hotel in the context of a single transaction. In the case where a single transaction (e.g. one airline reservation) is the unit-of-work this data may be packaged as a single MIME body part with a content-type (aka media type) identifying the data (e.g. application/xml, application/EDI-X12, image/jpeg, etc.) each will have a corresponding entry in the manifest that references it. Following is an example of a single transaction packaged within a single payload container:

```
Content-ID: airlinereservation111@imacompany.com
Content-Type: application/xml: charset="utf-8"

<OTA_AirBookRQ>
<!-- content omitted for brevity… -->
</OTA_AirBookRQ>
```

When a unit-of-work involves multiple transaction documents (e.g. a reservation delivery notification with a corresponding customer profile) the data may be packaged within a MIME multipart content-type containing multiple body parts, with each body part containing a single transaction document. For example the following multipart MIME structure represents a single unit-of-work that would exist in a single payload container:

```
Content-type: multipart/related; boundary="BoundarY"
--BoundarY
Content-ID: hotelreservation111@imacompany.com
Content-Type: application/xml: charset="utf-8"

<OTA_HotelResNotifRQ>
<!-- content omitted for brevity… -->
</OTA_HotelResNotifRQ>

--BoundarY
Content-ID: hotelreservation111profile@imacompany.com
Content-Type: application/xml: charset="utf-8"

<OTA_CreateProfileRQ>
<!-- content omitted for brevity… -->
</OTA_CreateProfileRQ>

--BoundarY--
```


In some cases an application system may operate in a “batch mode” where multiple, related transactions may be packaged as a single unit-of-work. For example, many POS systems process a batch of transactions during the settlement process. A single “batch” may contain individual transactions (e.g. authorizations, credits, voids, etc.) that are processed as a single unit-of-work. OTA compliant systems are allowed to package multiple transactions into a “batch” for processing as a single unit-of-work. For example a hotel reservation system capable of processing a “batch” of reservations may expect to receive the following payload container:

```
Content-type: multipart/related; boundary="BoundarY"
--BoundarY
Content-ID: hotelreservation111@imacompany.com
Content-Type: application/xml: charset=”utf-8”

<OTA_HotelResNotifRQ>
</OTA_HotelResNotifRQ>

--BoundarY
Content-ID: hotelreservation112@imacompany.com
Content-Type: application/xml: charset=”utf-8”

<OTA_HotelResNotifRQ>
</OTA_HotelResNotifRQ>

--BoundarY
Content-ID: hotelreservation113@imacompany.com
Content-Type: application/xml: charset=”utf-8”

<OTA_HotelResNotifRQ>
</OTA_HotelResNotifRQ>

--BoundarY--
```

4.2.3 Content-type for OTA XML Payloads

OTA recommends that implementations specify a content-type of application/xml with the optional character set attribute set to “utf-8”.

4.3 Classes of Message Delivery

The EbXML Messaging Service [ebMS] on top of SOAP with attachments which does not inherently provide any underlying mechanism for reliable messaging. An ebXML ebMS implementation will however provide the ability to use reliable messaging for the transport of messages.

Within OTA it is anticipated that there is a need for both reliable and non-reliable messaging, depending upon the type of messages being exchanged.

4.3.1 Historical use within the travel industry

The travel industry has a long history of automated message exchange dating back to the 1960’s. Broadly speaking two classes of message delivery have been used and continue to be used in legacy systems:

- Type A – interactive, best effort delivery
- Type B – store-and-forward, ‘guaranteed’, ordered delivery

4.3.1.1 Type A[22]

Type A message delivery is typically used for the expedient delivery of interactive messages (usually request/response pairs synchronous within a sub-conversation). Upon non-response, failure or timeout applications may fall back to alternate messages via type-B.

NOTE: Type A messages utilize ebXML's Best Effort delivery semantics, which makes no guarantee as to the delivery of a message, nor does it prevent duplicate messages from being delivered. It is possible to have multiple deliveries of a single Type A message, which could result in the receipt and processing of duplicate messages by a receiving Message Service Handler. It is assumed that a recipient application program, above the Message Service Handler, will prevent the processing of duplicate Type A messages. This will be relatively easy to do given timeToLive constraints.

4.3.1.2 Type B[23]

Type B message delivery is typically used for 'guaranteed', ordered delivery of messages. Delivery of type-B messages is often at a lower priority than type-A messages. Type-B messages are used in scenarios where responses may or may not be expected. They are not used for synchronous request/response type exchanges.

4.3.2 EbXML Classes of Delivery

The ebMS has been designed to meet a spectrum of requirements from one-way, unreliable message delivery up-to and including guaranteed, ordered message delivery using synchronous or asynchronous exchanges.

4.4 EbXML Header Document

The ebMS defines an ebXML header as an XML document, structured according to SOAP version 1.1. A SOAP XML document consists of one Envelope Element, which contains one mandatory body element and one optional header element. The ebMS mandates that all ebXML header documents MUST contain both header and body elements. The following example depicts a skeleton SOAP structure:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Header>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.4.1 OTA Subset of an ebXML Header Document

An ebXML header document contains the information necessary for two communicating Message Service Handlers to engage in a conversation involving the exchange and processing of an OTA unit-of-work. The ebMS defines several complex header elements that are used for

[22] The term 'type-A' comes from the SLC P1024A wire protocol commonly used throughout the 1970's and 1980's (the acronym SLC stands for Synchronous Link Control – synchronous, 6-bit, little or no error recovery)

[23] the term 'type-B' comes from the SLC P1024B wire protocol commonly used throughout the 1970's and 1980's

identification, routing, Quality of Service, Tracking and Status reporting. OTA does not require an implementer to support all of the functionality defined by ebMS. OTA implementers are only required to support the functionality needed to support the exchange of type A and B messages.

ebXML's Message Service identifies 6 child elements of the SOAP header element and 4 child elements of the SOAP body element. For purposes specific to OTA, only the following ebXML header elements are allowed to appear as child elements of the SOAP Header:

Element Name	*Usage*	*Message Types*
Acknowledgement	REQUIRED	B, RESPONSE
	OPTIONAL	A, RESPONSE
Via	REQUIRED	A, REQUEST
MessageHeader	REQUIRED	A and B, REQUEST and RESPONSE

NOTE: the ebXML Message Service 1.0 specification supports two additional elements within a SOAP-ENV:Header, eb:ErrorList, eb:TraceHeaderList and ds:Signature. These elements are not required by this version of the OTA specification, but they may be used between consenting parties as needed.

4.4.1.1 Acknowledgement Element

The `Acknowledgement` element will only appear as part of an ebXML response message. It is used to indicate that a request message has been successfully received by a receiving Server. Further details of the `Acknowledgement` element can be found in section 8.6 of the ebMS. The `Acknowledgement` element used by OTA contains the following attributes and child elements:

The `Acknowledgement` element contains three REQUIRED attributes:

- `SOAP-ENV:mustUnderstand` with a value of **"1"**
- `SOAP-ENV:actor` with the value **"http://schemas.xmlsoap.org/soap/actor/next"**
- `eb:version` with a value of **"1.0"**

The `Acknowledgement` element contains one required element, `Timestamp` that contains the creation date and time of the `Acknowledgement` formatted in accordance with XML Schema timeInstant. Following is an `Acknowledgement` example:

```
<eb:Acknowledgment SOAP-ENV:mustUnderstand="1" eb:version="1.0"
              SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
          <eb:Timestamp>2001-03-09T12:22:30Z</eb:Timestamp>
</eb:Acknowledgement>
```

The existence of an `Acknowledgement` element in a Response indicates that a Request message was successfully received. The existence of an `ErrorList` in a Response indicates that a Request has failed.

4.4.1.2 Via Element

The `Via` element is used to indicate the type of response (synchronous/asynchronous) expected of a server receiving a request message. Further details of the `Via` element can be found in

section 8.7 of the ebMS. It is only used on type A messages, those requiring a synchronous, business level response.

The `Via` element contains four REQUIRED attributes and one optional attribute:

REQUIRED attributes:

- `SOAP-ENV:mustUnderstand` with a value of **"1"**
- `SOAP-ENV:actor` with a value of "**http://schemas.xmlsoap.org/soap/actor/next**"
- `eb:version` with a value of **"1.0"**
- `eb:syncReply` with a value of **"true"**

OPTIONAL attribute:

- `eb:ackRequested` with a value of "**Unsigned"**

Type A messages MAY contain an `eb:ackRequested="Unsigned"` to indicate that the sender requests that the receiving Message Service Handler return an `Acknowledgement` element indicating that a message was successfully received and processed.

Following is an example usage of the `Via` element:

```
<eb:Via SOAP-ENV:mustUnderstand="1" SOAP-
ENV:actor="http://schemas.xmlsoap.org/soap/actor/next" eb:version="1.0" eb:syncReply="true"
eb:ackRequested="Unsigned"> </eb:Via>
```

4.4.1.3 MessageHeader Element

Each Type A and type B, request and response message MUST contain one MessageHeader Element. Each MessageHeader element MUST contain one SOAP-ENV:mustUnderstand attribute with a value of **"1"** (including double quotes) and one eb:version attribute with a value of **"1.0"**. See example below:

```
<eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
…
</eb:MessageHeader>
```

OTA utilizes the following child elements of the Message Header element:

Element Name	*Usage*	*Message Types*	*Purpose*
To	REQUIRED	ALL	Identify Recipient
From	REQUIRED	ALL	Identify Sender
CPAId	REQUIRED	ALL	Identify a TPA
ConversationId	REQUIRED	ALL	Session context
Service	REQUIRED	ALL	Service to invoke
Action	REQUIRED	ALL	Action to perform
MessageData	REQUIRED	ALL	Message Identification
QualityOfServiceInfo	REQUIRED	Type A and B Request	Indicates type of delivery semantics
SequenceNumber	REQUIRED	Type B Request	Needed for ordered delivery
Description	OPTIONAL	ALL	Human readable description

4.4.1.3.1 To element

Contains one child element, PartyId, which contains the DUNS Number of the intended recipient. The PartyId element contains one attribute, “type”, containing the value “urn:x12.org:I05:01”. This value indicates the data contained in the PartyId element is semantically defined by the X12 organization (x12.org), I05 indicates the particular set of identifiers defined by X12 to identify “parties” and 01 indicates the content of PartyId is a DUNS Number. Following is an example of the To element:

```
<eb:To>
    <eb:PartyId eb:type=”urn:x12.org:I05:01”>123456789</eb:PartyId>
</eb:To>
```

4.4.1.3.2 From element

Contains one child element, PartyId, which contains the DUNS Number of the sender. The PartyId element contains one attribute, “type”, containing the value “urn:x12.org:I05:01”. This value indicates the data contained in the PartyId element is semantically defined by the X12 organization (x12.org), I05 indicates the particular set of identifiers defined by X12 to identify “parties” and 01 indicates the content of PartyId is a DUNS Number. Following is an example of the From element:

```
<eb:From>
<eb:PartyId eb:type=”urn:x12.org:I05:01”>987654321</eb:PartyId>
</eb:From>
```

4.4.1.3.3 CPAId element

This element is required by ebXML, however its content is determined by mutual consent between parties. CPAId may be used to identify a particular agreement (e.g. trading partner agreement) that defines the “rules of engagement” two communicating parties agree to abide by. If no such agreement exists CPAId MUST contain a constant value of “NULL”, for example:

```
<eb:CPAId>NULL</eb:CPAId>
```

4.4.1.3.4 ConversationId element

This element is used to provide context for a particular exchange of messages. It is primarily used for session identification. The content of ConversationId is a compound string consisting of sid@FQDN. A “sid” is a session identifier. Each party is required to construct unique sid values for each new session established. A “Fully Qualified Domain Name” (FQDN) MUST be appended to a sid, in order to ensure the uniqueness of ConversationId’s across company boundaries.

A ConversationId is obtained by a sender during the establishment of a new session. Refer to the section titled “Sessions in OTA” for detailed usage of this element. Following is an example ConversationId:

```
<eb:ConversationId>20011017161501-777@B2Bserver.imacompany.com
</eb:ConversationId>
```

4.4.1.3.5 Service and Action elements

The Service and Action elements identify the particular operation being performed, which correspond directly to specific protocol behavior. The Service element may contain one of the possible values defined in the section titled “Service and Action Mappings”. The service element MUST contain a “type” attribute with the fixed value “OTA” when using Services defined by this document. Other Services MAY be utilized which are not defined within this document (e.g.

ebXML Ping message). When using non-OTA defined Services implementers are expected to follow the usage guidelines defined for the Service.

The Action element is used to provide additional context to the Service element. The Action identifies the protocol "verb". Taken together the Service/Action should provide sufficient identification to invoke proper protocol behavior between parties. The list of possible values for the Service element, the type attribute and the Action element are listed in section titled "Service and Action Mappings".

Following is an example:

```
<eb:Service type="OTA">Profile</eb:Service>
<eb:Action>OTA_CreateProfileRQ</eb:Action>
```

4.4.1.3.6 MessageData element

- This element is used to provide identification information for each ebXML message exchanged between parties. There are three possible child elements under MessageData:

Element Name	*Usage*	*Message Types*	*Purpose*
MessageId	REQUIRED	ALL	Unique Message Id
Timestamp	REQUIRED	ALL	Creation date/time
RefToMessageId	REQUIRED	Responses Only	MessageId of request
TimeToLive	REQUIRED	Type A Request	Specify delivery and processing expiration date/time

4.4.1.3.6.1 MessageId Element

Contains a unique message identifier formed in conformance with RFC 2392, which defines a string containing uniqueidentifier@FQDN, for example:

```
<eb:MessageId>mid:abc123@b2bserver.imacompany.com</eb:MessageId>
```

4.4.1.3.6.2 Timestamp Element

Contains the creation date and time of the message formatted in accordance with XML Schema timeInstant, for example:

```
<eb:TimeStamp>2001-02-15T11:12:12Z</eb:Timestamp>
```

4.4.1.3.6.3 RefToMessageId Element

Contains the message identifier from the original MessageId that "this" message is in response to, for example:

```
<eb:RefToMessageId>mid:abc123@b2bserver.imacompany.com</eb:RefToMessageId>
```

4.4.1.3.6.4 TimeToLive Element

Contains the expiration date and time of a message formatted in accordance with XML Schema timeInstant, for example:

```
<eb:TimeToLive>2001-02-15T11:12:12Z</eb:TimeToLive>
```

Both the sending and receiving ebMS are expected to generate and report an error condition whenever message delivery/processing has aborted due to expiration. A Sending ebMS MUST

notify a higher layer application, through its service interface or by some other means, whenever a message request aborts due to expiration. If a Sending ebMS has been configured to retry message delivery for messages with BestEffort deliverySemantics (ref: retries in section 10.2.6 of [ebMS], NOTE: this is not the recommended behavior for BestEffort deliverySemantics) then the sending ebMS SHOULD attempt to resend a message that aborts due to expiration. The recommended minimum time for TimeToLive SHOULD be no less than 5 seconds in the future from the date time value contained in the MessageData/Timestamp.

A request message containing a TimeToLive value is considered satisfied within its allotted time period if a corresponding response message is received by the sending ebMS before the point in time identified by the TimeToLive.

Message Service Handlers are expected to maintain time synchronization by synchronizing systems clocks, at least monthly, with a recognized time source, such as the National Institute of Standards and Technology NIST-F1 Cesium Fountain Atomic Clock, http://nist.time.gov/timezone.cgi?UTC/s/0

4.4.1.3.7 QualityOfServiceInfo element

This element is used to indicate the need for reliable message exchange between parties. The ebMS defines three possible attributes for this element, however OTA will only utilize two, **deliverySemantics** and **messageOrderSemantics.**

This element is used on type A and B Request Messages. Type A messages MUST specify a deliverySemantics attribute with the value **"BestEffort"**, (including double quotes). Type B messages MUST specify a deliverySemantics attribute with the value **"OnceAndOnlyOnce"** (including double quotes) and messageOrderSemantics containing the value "Guaranteed". Examples below:

TYPE A:

```
<eb:QualityOfServiceInfo eb:deliverySemantics="BestEffort"/>
```

TYPE B:

```
<eb:QualityOfServiceInfo eb:deliverySemantics="OnceAndOnlyOnce"
messageOrderSemantics="Guaranteed"/>
```

4.4.1.3.8 SequenceNumber

This element indicates the sequence in which messages MUST be processed by a receiving Message Service. Each Type B request message MUST contain a SequenceNumber element in order to guarantee ordered delivery. This element contains an integer value in the range 0-99999999. There is one REQUIRED attribute, "status", that MUST contain one of the following values, "Reset" or "Continue". Detailed usage of this element may be found in section 8.4.8 of [ebMS].

4.4.1.3.9 Description

This element contains a human readable description that SHOULD be germane to the Service/Action of the message in which it appears. The element MAY be present on any request or response message.

4.5 SOAP Body Elements

The following ebXML header elements MUST be located within the SOAP Body element, when required:

- **Manifest**

The Manifest element is used to identify all of the payload documents within in a payload container. Each document within a payload container MUST be identified by a corresponding Reference element within the Manifest. The Manifest element and at least one Reference element MUST be present in a Message Package containing a payload container.

The Reference element is used to identify information needed by a particular service/action pair in order to perform proper processing. Conceptually, the Reference element can be thought of as identifying the "arguments" needed by a particular Service/Action. A Reference element contains two Xlink attributes that provide information about a payload. The two attributes are:

Attribute Name	*Purpose*	*Possible Values*
xlink:href	Contains a URI identifying a payload document	Examples: cid:123@imacompany.com http://www.imacompany.com/file1
xlink:type		"simple"

Following is an example of a Manifest element containing one Reference element, indicating the presence of one payload document:

```
    <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
        <eb:Reference xlink:href="cid:hotelreservation111@example.com"
           xlink:type="simple">
        </eb:Reference>
    </eb:Manifest>
```

4.6 EbXML Collaboration Protocol Profile

EbXML has defined the Collaboration-Protocol-Profile and Agreement Specification [ebCPP[24]] for the definition of key parameters used by trading partners within the context of their on-the-wire eCommerce conversations.

Whereas portions of a valid CPP document are pertinent only to the parties participating within a particular conversation, other portions of a CPP may be more generally applicable. In the interests of fostering wire compatibility between implementations the OTA RECOMMENDS the use of the following CPP fragment within any CPP documents you may define. This OTA fragment defines transport and delivery channels consistent with the infrastructure defined in this document:

```
<tp:Transport tp:transportId="HTTPS01">
  <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
  <tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
  <tp:Endpoint tp:uri="http://example.com/servlet/ebxmlhandler"
               tp:type="allPurpose"/>
  <tp:TransportSecurity>
    <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
  </tp:TransportSecurity>
</tp:Transport>

<tp:DocExchange tp:docExchangeId="TYPEA">
  <tp:ebXMLBinding tp:version="1.0">
```

[24] see: http://www.ebxml.org/specs/ebCPP.pdf


```
     <tp:ReliableMessaging tp:deliverySemantics="BestEffort" tp:idempotency="true">
     </tp:ReliableMessaging>
  </tp:ebXMLBinding>
</tp:DocExchange>

<tp:DocExchange tp:docExchangeId="TYPEB">
 <tp:ebXMLBinding tp:version="1.0">
   <tp:ReliableMessaging tp:deliverySemantics="OnceAndOnlyOnce"
                         tp:idempotency="true"
                         tp:messageOrderSemantics="NotGuaranteed">
     <tp:Retries>5</tp:Retries>
     <tp:RetryInterval>1800</tp:RetryInterval> <!--time in seconds-->
     <tp:PersistDuration>24H</tp:PersistDuration>
   </tp:ReliableMessaging>
 </tp:ebXMLBinding>
</tp:DocExchange>

<tp:DeliveryChannel tp:channelId="TYPEA" tp:transportId="HTTPS01"
                    tp:docExchangeId="TYPEA">
    <tp:Characteristics tp:syncReplyMode="responseOnly"
                        tp:nonrepudiationOfOrigin="false"
                        tp:nonrepudiationOfReceipt="false"
                        tp:secureTransport="true"
                        tp:confidentiality="false"
                        tp:authenticated="true"
                        tp:authorized="true"/>
</tp:DeliveryChannel>

<tp:DeliveryChannel tp:channelId="TYPEB" tp:transportId="HTTPS01"
                    tp:docExchangeId="TYPEB">
   <tp:Characteristics tp:syncReplyMode="none"
                       tp:nonrepudiationOfOrigin="false"
                       tp:nonrepudiationOfReceipt="false"
                       tp:secureTransport="true"
                       tp:confidentiality="false"
                       tp:authenticated="true"
                       tp:authorized="true"/>
</tp:DeliveryChannel>
```

4.7 EbXML Header Examples

4.7.1 Type A Request Message

```
Content-ID: <ebxhmheader111@B2B.company.com>
Content-Type: text/xml; charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope  xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
                    xmlns:eb=”http://www.ebxml.org/namespaces/messageHeader”
                    xmlns:xlink=”http://www.w3.org/1999/xlink”>
  <SOAP-ENV:Header>
    <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
        <eb:From>
            <eb:PartyId type="urn:x12.org:I05:01">123456789</eb:PartyId>
        </eb:From>
        <eb:To>
            <eb:PartyId type=" urn:x12.org:I05:01">912345678</eb:PartyId>
        </eb:To>
        <eb:CPAId>NULL</eb:CPAId>
    <eb:ConversationId>20001209133003285720b2b.company.com
          </eb:ConversationId>
        <eb:Service eb:type="OTA">Profile</eb:Service>
        <eb:Action>OTA_CreateProfileRQ</eb:Action>
        <eb:MessageData>
             <eb:MessageId>mid:20001209-133003-28572@b2b.company.com</eb:MessageId>
             <eb:Timestamp>2001-02-15T11:12:12Z </eb:Timestamp>
             <eb:TimeToLive>2001-02-15T11:12:17Z </eb:TimeToLive>
        </eb:MessageData>
        <eb:QualityOfServiceInfo eb:deliverySemantics="BestEffort"/>
```



```
    </eb:MessageHeader>
  <eb:Via SOAP-ENV:mustUnderstand=”1”
          SOAP-ENV:actor=”http://schemas.xmlsoap.org/soap/actor/next”
          eb:version=”1.0” eb:syncReply=”true” />
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
        <eb:Reference xlink:href="cid:profile111@b2b.company.com"
           xlink:type="simple">
        </eb:Reference>
    </eb:Manifest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.7.2 Type B Request Message

```
Content-ID: <ebxhmheader111@B2B.company.com>
Content-Type: text/xml; charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope  xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
                    xmlns:eb=”http://www.ebxml.org/namespaces/messageHeader”
                    xmlns:xlink=”http://www.w3.org/1999/xlink”>
<SOAP-ENV:Header>
    <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
        <eb:From>
            <eb:PartyId type="urn:x12.org:I05:01">123456789</eb:PartyId>
        </eb:From>
        <eb:To>
            <eb:PartyId type=" urn:x12.org:I05:01">912345678</eb:PartyId>
        </eb:To>
        <eb:CPAId>NULL</eb:CPAId>
    <eb:ConversationId>20001209133003328572@b2b.company.com
             </eb:ConversationId>
        <eb:Service eb:type="OTA">Profile</eb:Service>
        <eb:Action>OTA_CreateProfileRQ</eb:Action>
        <eb:MessageData>
             <eb:MessageId>mid:20001209-133003-28572@b2b.company.com</eb:MessageId>
             <eb:Timestamp>2001-02-15T11:12:12Z </eb:Timestamp>
        </eb:MessageData>
        <eb:QualityOfServiceInfo eb:deliverySemantics="OnceAndOnlyOnce"
                                 eb:messageOrderSemantics=”Guaranteed”/>
       <eb:SequenceNumber eb:status=”Reset”>0</eb:SequenceNumber>   </eb:MessageHeader>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
    <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
        <eb:Reference xlink:href="cid:profile111@b2b.company.com"
           xlink:type="simple">
        </eb:Reference>
        <eb:Reference xlink:href="cid:profile112@b2b.company.com"
           xlink:type="simple">
        </eb:Reference>
        <eb:Reference xlink:href="cid:profile113@b2b.company.com"
           xlink:type="simple">
        </eb:Reference>
        <eb:Reference xlink:href="cid:profile114@b2b.company.com"
           xlink:type="simple">
        </eb:Reference>
    </eb:Manifest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.7.3 Type A Response Example

```
Content-ID: <ebxhmheader111@B2Bserver.imacompany.com>
Content-Type: text/xml; charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope  xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
                    xmlns:eb=”http://www.ebxml.org/namespaces/messageHeader”
```



```
                    xmlns:xlink=”http://www.w3.org/1999/xlink”>
<SOAP-ENV:Header>
    <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
        <eb:From>
            <eb:PartyId type="urn:x12.org:I05:01">912345678</eb:PartyId>
        </eb:From>
        <eb:To>
            <eb:PartyId type=" urn:x12.org:I05:01">123456789</eb:PartyId>
        </eb:To>
        <eb:CPAId>NULL</eb:CPAId>
    <eb:ConversationId>20001209133003283572@b2b.company.com
             </eb:ConversationId>
        <eb:Service eb:type="OTA">Profile</eb:Service>
        <eb:Action>OTA_CreateProfileRS</eb:Action>
        <eb:MessageData>
             <eb:MessageId>mid:20001209-133503-1111@B2Bserver.imacompany.com</eb:MessageId>
             <eb:Timestamp>2001-02-15T11:15:12Z </eb:Timestamp>
                          <eb:RefToMessageId> mid:20001209-133003-
28572@b2b.company.com</eb:RefToMessageId>
        </eb:MessageData>
    </eb:MessageHeader>

</SOAP-ENV:Header>
<SOAP-ENV:Body>
    <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
        <eb:Reference xlink:href="cid:profile111@b2b.imacompany.com"
           xlink:type="simple">
        </eb:Reference>
    </eb:Manifest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.7.4 Type B Response Example

```
Content-ID: <ebxhmheader111@B2Bserver.imacompany.com>
Content-Type: text/xml; charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope  xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
                    xmlns:eb=”http://www.ebxml.org/namespaces/messageHeader”
                    xmlns:xlink=”http://www.w3.org/1999/xlink”>
<SOAP-ENV:Header>
    <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
        <eb:From>
            <eb:PartyId type="urn:x12.org:I05:01">912345678</eb:PartyId>
        </eb:From>
        <eb:To>
            <eb:PartyId type=" urn:x12.org:I05:01">123456789</eb:PartyId>
        </eb:To>
        <eb:CPAId>NULL</eb:CPAId>
    <eb:ConversationId>20001209133003283572@b2b.company.com
             </eb:ConversationId>
        <eb:Service eb:type="OTA">Profile</eb:Service>
        <eb:Action>OTA_CreateProfileRS</eb:Action>
        <eb:MessageData>
             <eb:MessageId>mid:20001209-133503-1111@B2Bserver.imacompany.com</eb:MessageId>
             <eb:Timestamp>2001-02-15T11:15:12Z </eb:Timestamp>
                          <eb:RefToMessageId> mid:20001209-133003-
28572@b2b.company.com</eb:RefToMessageId>
        </eb:MessageData>
    </eb:MessageHeader>
<eb:Acknowledgment SOAP-ENV:mustUnderstand="1" eb:version="1.0"
               SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
            <eb:Timestamp>2001-03-09T12:22:30Z</eb:Timestamp>
</eb:Acknowledgement>

</SOAP-ENV:Header>
<SOAP-ENV:Body>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


4.7.4.1 Reliable Messaging

In the context of OTA, reliable messaging refers to the requirements that an ebXML Message Service Product be capable of reliably delivering and persisting incoming and outgoing payload data until the successful delivery to a trading partner or an internal application. It is expected that all ebXML compliant Message Service products are capable of performing reliable messaging in accordance with the ebMS specification following the requirements for `reliableMessagingMethod="ebXML"` (the only type supported by OTA at this time and the default method for ebXML) in section 10.2.4.

A message requires reliable delivery if the `deliverySemantics="OnceAndOnlyOnce"`. Only type B request messages require reliable delivery.

4.7.4.2 Once and Only Once Messaging

ebXML supports two type of delivery Semantics:

1. OnceAndOnlyOnce
2. BestEffort (this is the default when unspecified)

All type A request messages MUST utilize BestEffort deliverySemantics. All type B request messages MUST utilize OnceAndOnlyOnce deliverySemantics with guaranteed ordered delivery. The OnceAndOnlyOnce deliverySemantics are used when guaranteed, single delivery of a message is required. A Message Service Handler is expected to identify and ignore duplicate type B message requests.

CAUTION: Messages sent using the BestEffort deliverySemantics may result in failed delivery or multiple deliveries of the same message. Implementers should take caution when using type A messages to ensure that failed message delivery or multiple message delivery does not compromise the integrity of a system. This can be easily achieved by using sequence numbers. Also, guaranteed ordered delivery does NOT insure that data contained in the payload portion of a message is "processed" in the order they were received.

4.7.5 Mapping Class of Delivery to Service/Action Pairs

In section 7 for each Service/Action pair there is a RECOMMENDED class of delivery. Implementations following these recommendations are expected to have wire interoperability.

4.8 Sessions in OTA

Previous versions of OTA infrastructure defined both session-oriented and single-shot conversations with differing conversational semantics and message control structures for each of these.

As there are currently no known use-cases among defined messages for the single-shot messaging OTA has defined simpler session-oriented conversations and RECOMMENDS that implementations always use a valid session during all conversations.

4.8.1 What we mean by 'sessions'

The term 'session' has many different meanings depending on context. Within the context of OTA infrastructure session is defined as an authenticated, authorized on-going conversation between two implementations. Sessions have the following characteristics:

- They are ongoing

- Session initiation and session termination are relatively low overhead
- Sessions may be long-lived (they may last for days, even weeks on end)
- Sessions are leased[25]
- Messages belonging to a session are identified with that session via a conversation-id and sequence numbers
- Session conversation-ids provide a convenient 'hook' for implementation end-points to maintain state (related, of course, to a particular session) e.g. security
- Sessions follow a client/server model

4.8.2 What we do not mean by 'sessions'

It is important to also explicitly note what we do not mean by the term 'session'. In particular, the concept of OTA sessions is completely orthogonal to the notion of 'session control' within booking conversations (those familiar with session control will recall that a 'session' is initiated implicitly and terminated explicitly via an 'ET' (End Transaction) message – analogous to a commit or an 'IG' (Ignore) message – analogous to a rollback).

Currently there is no notion of traditional travel industry session control within defined OTA message sets, however as and when such a notion is defined it will be via message sets and new actions on corresponding services. This notion of 'session-control' belongs at a higher layer much closer to the application, and is conceptually entirely different from the infrastructure layer sessions being defined here.

4.8.3 The OTA Session service

The OTA session service is used to establish context for either a single request/response message exchange or a series of request/response message exchanges between parties. A Sender MUST establish a Session with a Receiver before any message exchanges containing OTA business transactions (units-of-work) will be allowed.

Messages exchanged during a session may be related to a single OTA unit-of-work or multiple, unrelated units-of-work. OTA's session service is also used to negotiate and establish operational parameters to govern communications that occur within a given session.

The Session service defines five possible Actions::

1. CreateRQ – sent by session initiator to establish a new session
2. CreateRS – sent by recipient of a CreateRQ to grant or deny a session
3. CloseRQ – sent by session initiator to close a session
4. CloseRS – sent by the recipient of a CloseRQ to indicate session closure
5. ErrorRS - sent by a server in response to any message to indicate a session error

4.8.3.1 Session/CreateRQ and Session/CreateRS

4.8.3.1.1 Session/CreateRQ

[25] Similar to a DHCP lease, or a lease on an apartment rights within a session will expire automatically unless renewed prior to the agreed expiration date and time

This service/action pair is used to request the start of a new session. The initiator of Session/CreateRQ MUST construct an ebXML message header, following the requirements for Type A request messages. The ebXML Header MessageHeader/ConversationId MUST contain a value of **NULL.**

A single payload container is used to pass a SessionControlRequest document that defines the operational parameters requested by the initiator. The SessionControlRequest document is an XML document containing the following information:

- A REQUIRED, recurring OTA_Version element that is used to identify the various OTA versions supported by the sender. A single attribute, preference, containing a numerical value that is used to indicate the senders "preferred" version to be used during this session. The lowest preference value indicates the senders highest preference. A preference value of "0" or the absence of a preference attribute indicates no preference.
- An OPTIONAL mode attribute that is used to identify "Test" or "Production" mode. If not specified `mode="Production"` is assumed.
- A REQUIRED version attribute (currently set to "1")
- An OPTIONAL LeaseRequestTime element that specifies the amount of time a session is expected to remain "alive". When present, the element contains a positive integer value or "0" (zero). The element also contains one OPTIONAL attribute, cadence, that contains one of the following values:
    - seconds
    - minutes
    - hours
    - days

  When the LeaseRequestTime element contains a value of "0" (zero) or the element is not present, this is semantically equivalent to "infinite". If the "cadence" attribute is not present, the value in LeaseRequestTime element SHALL be interpreted as "seconds".
- An OPTIONAL MaxIdleTime element that specifies the amount of time a session may remain idle before the server system "expires" the session. When present, the element contains a positive integer value or "0" (zero). The element also contains one OPTIONAL attribute, cadence, that contains one of the following values:
    - seconds
    - minutes
    - hours
    - days

  When the MaxIdleTime element contains a value of "0" (zero) or the element is not present, this is semantically equivalent to "infinite". If the "cadence" attribute is not present, the value in MaxIdleTime element SHALL be interpreted as "seconds".

An example SessionControlRequest document follows:

```
<SessionControlRequest mode="Production" version="1">
<OTA_Version preference="1">2001C</OTA_Version>
<OTA_Version preference="2">2001A</OTA_Version>
<LeaseRequestTime cadence="seconds">60</LeaseRequestTime>
<MaxIdleTime cadence="minutes">15</MaxIdleTime>
</SessionControlRequest>
```


Additional elements and attributes may be added in a later version of this specification that could be used to identify OTA capabilities (e.g. supported Transactions) and other operational parameters.

4.8.3.1.2 Session/CreateRS

This service/action pair is sent in response to a Session/CreateRQ to inform the requesting party the status of the request. The sender of Session/CreateRS MUST construct an ebXML message header, with a RefToMessageId containing the value of MessageId in the Session/CreateRQ message, along with the other information required within response messages. The ebXML Header MessageHeader/ConversationId MUST contain a value of **NULL.**

A single payload container is used to pass a SessionControlResponse document that defines the status of the request and other operational parameters that will be used during the session. The SessionControlResponse document is an XML document containing the following attributes and elements:

A "status" attribute containing one of the following values:

- Approved
- Rejected
- A single OTA_Version element identifies the OTA version that both parties agree to follow during the session.
- A single ConversationId element containing the value of ConversationId that MUST be used in the MessageHeader/ConversationId element of all subsequent messages exchanged during this session.
- An OPTIONAL LeaseRequestTime element that specifies the amount of time a server has will allow a session to remain "alive". This element MUST be present, if the corresponding Session/CreateRQ message contained a LeaseRequestTime. The element contains a positive integer value or "0" (zero). The element also contains one OPTIONAL attribute, cadence, that contains one of the following values:
    - seconds
    - minutes
    - hours
    - days

  When the LeaseRequestTime element contains a value of "0" (zero) or the element is not present, this is semantically equivalent to "infinite". If the "cadence" attribute is not present, the value in LeaseRequestTime element SHALL be interpreted as "seconds".
- An OPTIONAL MaxIdleTime element that specifies the amount of time a server will allow a session to remain idle before it "expires" the session. This element MUST be present, if the corresponding Session/CreateRQ message contained a MaxIdleTime. When present, the element contains a positive integer value or "0" (zero). The element also contains one OPTIONAL attribute, cadence, that contains one of the following values:
    - seconds
    - minutes
    - hours

- days

When the MaxIdleTime element contains a value of “0” (zero) or the element is not present, this is semantically equivalent to “infinite”. If the “cadence” attribute is not present, the value in MaxIdleTime element SHALL be interpreted as “seconds”.

- Zero or more Reason elements indicating the reason why a Session was rejected. The Reason element MAY only be present when the status attribute of the SessionControlResponse element contains the value “Rejected”. When a session has been Rejected the SessionControlResponse document MUST NOT contain a ConversationId element.
- Zero or one ServiceSupported element, see section 6 for details of this element. If the value of the status attribute is `status=”Approved”` a <`ServicesSupported`> element MUST be present.

Following are two example SessionControlResponse documents:

```
<SessionControlResponse version=”1” status=”Approved”>
    <OTA_Version>2001C</OTA_Version>
    <ConversationId>20011027081907-862@host.imacompany.com</ConversationId>
    <LeaseRequestTime cadence=”seconds”>60</LeaseRequestTime>
    <MaxIdleTime cadence=”minutes”>15</MaxIdleTime>
    <ServicesSupported>
        <!—- content omitted for brevity -->
    </ServicesSupported>
</SessionControlResponse>
```

```
<SessionControlResponse verson=”1” status=”Rejected”>
<OTA_Version>2001C</OTA_Version>
<Reason>No System Available to Process Request – Scheduled Maintenance</Reason>
</SessionControlResponse>
```

The negotiated parameters returned on the response define the values that will be used for the remainder of the session.

4.8.3.2 Session/CloseRQ and Session/CloseRS

4.8.3.2.1 Session/CloseRQ

This service/action pair is used to request the closure of a session. The initiator of Session/CloseRQ MUST construct an ebXML message header, following the requirements for Type A request messages. The ebXML Header MessageHeader/ConversationId MUST contain the value of the session to be closed. This is the same ConversationId value that was contained in the Session/CreateRS SessionControlResponse document when the session was Accepted**.**

The recipient of a Session/CloseRQ must take steps to ensure that only authorized parties are allowed to close a session. Any Session/CloseRQ messages containing a ConversationId that was not issued to the party issuing the Session/CloseRQ MUST be responded to with Session/CloseRS message containing a <`ErrorList`><`Error`> indicating that the CloseRQ has failed. and the session MUST remain active, if it is already active.

A SESSION MAY ONLY BE CLOSED BY THE PARTY THAT WAS GRANTED THE SESSION AND WITHIN THE CONTEXT OF THE SESSION BEING CLOSED (by using the MessageHeader/ConversationId that was issued by the Acceptor of the session). Implementers SHOULD maintain some identifying characteristics (e.g. IP address of original Requesting Party) as verification before completing a Session/CloseRQ

Once a session has been closed, either due to expiration or during a CloseRQ, no further message exchanges are allowed over that session. Any attempt to send a message containing a ConversationId for a session that has been closed MUST be responded to with a <ErrorList><Error> indicating that the session is no longer active within the appropriate response Service/Action message.

There is no payload associated with this message.

4.8.3.2.2 Session/CloseRS

This service/action pair is sent in response to a Session/CloseRQ indicating that a session has been closed. The sender of Session/CloseRS MUST construct an ebXML message header, with a RefToMessageId containing the value of MessageId in the Session/CloseRQ message, along with the other information required within response messages. The ebXML Header MessageHeader/ConversationId MUST contain the value of the closed session.

A SESSION MAY ONLY BE CLOSED BY THE PARTY THAT WAS GRANTED THE SESSION AND WITHIN THE CONTEXT OF THE SESSION BEING CLOSED (identified by the ConversationId that was issued by the Acceptor of the session). Implementers SHOULD maintain some identifying characteristics (e.g. IP address of original Requesting Party) as verification before completing a Session/CloseRQ and issuing a Session/CloseRS.

Once a session has been closed, either due to expiration or during a CloseRQ, no further message exchanges are allowed over that session. Any attempt to send a message containing a ConversationId for a session that has been closed MUST be responded to with a <ErrorList><Error> indicating that the session is no longer active within the appropriate response Service/Action message.

There is no payload associated with this message.

4.8.3.3 SessionControl Schema definition

The following schema fragment formally defines SessionControl messages:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created and edited with 'vi' -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.opentravel.org/OTA"
    targetNamespace="http://www.opentravel.org/OTA"
    elementFormDefault="qualified">

    <xs:annotation>
        <xs:documentation xml:lang="en">
OTA v2001C Specification - SessionControl message definitions
Copyright (C) 2001 Open Travel Alliance. All rights reserved.
        </xs:documentation>
    </xs:annotation>

    <xs:element name="SessionControlRequest">
        <xs:annotation>
            <xs:documentation xml:lang="en">
A SessionControlRequest is used to negotiate parameters to apply
to an OTA session.
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="OTA_Version" maxOccurs="unbounded" />
                <xs:element ref="LeaseRequestTime" minOccurs="0" />
                <xs:element ref="MaxIdleTime" minOccurs="0" />
            </xs:sequence>
            <xs:attribute name="version" type="versionNumber" use="required" />
```

```
            <xs:attribute name="mode" default="Production">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="Test"/>
                        <xs:enumeration value="Production"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
        </xs:complexType>
    </xs:element>

    <xs:element name="OTA_Version" type="xs:string">
        <xs:attribute name="preferred" type="xs:integer" />
    </xs:element>

    <xs:element name="LeaseRequestTime" type="xs:integer">
        <xs:attribute name="cadence" type="cadenceType" default="seconds"/>
    </xs:element>

    <xs:simpleType name="cadenceType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="seconds"/>
            <xs:enumeration value="minutes"/>
            <xs:enumeration value="hours"/>
            <xs:enumeration value="days"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:element name="MaxIdleTime" type="xs:integer">
        <xs:attribute name="cadence" type="cadenceType" default="seconds"/>
    </xs:element>

    <xs:element name="SessionControlResponse">
        <xs:annotation>
            <xs:documentation xml:lang="en">
A SessionControlResponse is used to negotiate parameters to apply
to an OTA session.
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="OTA_Version" />
                <xs:element ref="ConversationId" />
                <xs:element ref="LeaseRequestTime" minOccurs="0" />
                <xs:element ref="MaxIdleTime" minOccurs="0" />
                <xs:element ref="Reason" minOccurs="0" maxOccurs="unbounded" />
                <xs:element ref="ServicesSupported" minOccurs="0" />
            </xs:sequence>
            <xs:attribute name="version" type="versionNumber" use="required" />
            <xs:attribute name="status" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="Accepted"/>
                        <xs:enumeration value="Rejected"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
        </xs:complexType>
    </xs:element>

    <xs:element name="ConversationId" type="xs:string" />
    <xs:element name="Reason" type="xs:string" />

    <xsd:simpleType name="versionNumber">
        <xsd:annotation>
            <xsd:documentation xml:lang="en">
All OTA version numbers are unsigned integers in the range 1..9999.
Version numbers should begin with the value '1' and be incremented
each time a message is revised.
            </xsd:documentation>
        </xsd:annotation>
```



```
        <xsd:restriction base="xsd:unsignedShort">
            <xsd:minInclusive value="1"/>
            <xsd:maxInclusive value="9999"/>
        </xsd:restriction>
    </xsd:simpleType>
</xs:schema>
```

4.8.3.4 Session/ErrorRS

OTA maintains an error response message <ErrorRS>, which an implementation may send in response to any request in the event of a session error. Note that this is an infrastructure and/or session related error, not an application level error. For application errors use an <OTA:Errors> element in the standard design pattern used for all response messages (see section 2.4.2 for details).

The following table contains all possible OTA session errors.

errorCode	*ErrorMessages*
SessionFailure-100	Version not supported
SessionFailure-101	Session has expired
SessionFailure-102	Session already closed
SessionFailure-103	Parameter not supported

The following schema fragment formally defines an ErrorRS message:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.opentravel.org/OTA"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.opentravel.org/OTA"
elementFormDefault="qualified">
    <xs:annotation>
        <xs:documentation xml:lang="en">
OTA v2001C Specification - Session ErrorRS message definition
Copyright (C) 2001 Open Travel Alliance. All rights reserved.
        </xs:documentation>
    </xs:annotation>
    <xs:include schemaLocation="OTA_v2ent.xsd"/>
    <xs:element name="ErrorRS">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:string">
                    <xs:attributeGroup ref="OTA_PayloadStdAttributes"/>
                    <xs:attribute name="ErrorCode" use="required">
                        <xs:simpleType>
                            <xs:restriction base="xs:NMTOKEN">
                                <xs:enumeration value="SessionFailure-100"/>
                                <xs:enumeration value="SessionFailure-101"/>
                                <xs:enumeration value="SessionFailure-102"/>
                                <xs:enumeration value="SessionFailure-103"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                    <xs:attribute name="Severity" use="optional"/>
                    <xs:attribute name="ErrorMessage" use="optional">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="OTA version not supported"/>
                                <xs:enumeration value="Session has expired"/>
                                <xs:enumeration value="Session already closed"/>
                                <xs:enumeration value="Parameter not supported"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
```



```
            </xs:extension>
          </xs:simpleContent>
       </xs:complexType>
    </xs:element>
</xs:schema>
```

4.8.4 Securing OTA Sessions

All data passed over an OTA session MUST be protected from unauthorized parties. Additionally, all servers used by OTA implementers MUST maintain access control in order to prevent unauthorized use of an OTA system. The security and integrity of an OTA system should be a top priority for all OTA implementers.

4.8.4.1 Basic-authorization and SSL

All OTA implementers MUST support, at a minimum, SSL version 3.0 using a minimum key size of 128 bits for symmetric cryptographic algorithms and a key size of 2048 bits for asymmetric cryptographic algorithms (e.g.public key). Presently, only servers are required to implement Digital Certificates. Some future version may require clients to implement Digital Certificates for authentication purposes during the establishment of a SSL connection. During the establishment of an SSL connection OTA servers are required to present a Digital Certificate, as part of the SSL handshake. All OTA clients MUST accept self signed Digital Certificates as well as those that have been signed by a recognized Certificate Authority (e.g. Verisign, Entrust, Thawte, et al).

All OTA implementers MUST use HTTP Basic Authentication (usernames and passwords) (ref: RFC 2617) to control access to an OTA system. Any party that fails to provide an authorized username/password pair in the Authorization header of an HTTP request when sending an OTA Request or Response message using HTTP POST method will likely generate an HTTP 401 error[26]. Implementers are expected to secure the transport of sensitive username/password data by only transporting this information over an established SSL connection with a server that is known to belong to your trading partner.

4.8.4.2 Towards deeper security

EbXML offers many levels of security beyond our base recommendation of basic-authorization over SSL, including the ability to:

- Digitally sign payloads
- Encrypt payloads
- Digitally sign and encrypt payloads
- Require client-side X.509 certificates on HTTP/S sessions

Different commercial ebXML implementations are expected to have differing level of support for these additional security features. Beyond our recommendation for a basic authenticated and secure channel it is up to trading partners to agree on any deeper level of security to be applied to their eCommerce exchanges.

4.9 Web Services Description for OTA ebXML

[26] The HTTP 401 error code indicates an unauthorized attempt to access a restricted resource

The following web services description language [wsdl][27] fragment formally defines OTA type A and type B requests and responses from a web services perspective, should OTA partners want to make use of a UDDI registry which supports WSDL:

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:s=”http://www.w3.org/2001/XMLSchema”
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:mime=”http://schemas.xmlsoap.org/wsdl/mime/”
 xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
 xmlns:soap=”http://schemas.xmlsoap.org/wsdl/soap/”
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 targetNamespace="http://www.ebxml.org/namespaces/messageHeader"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

 <types>
      <schema targetNamespace=http://www.ebxml.org/namespaces/messageHeader"
                    xmlns="http://www.w3.org/2000/10/XMLSchema">
 </types>

  <message name="TypeARequest">
    <part name="MessageHeader" type="eb:MessageHeader" />
    <part name="Via" type="eb:Via" />
    <part name="Manifest" type="eb:Manifest" />
  </message>

  <message name="TypeAResponse">
    <part name="MessageHeader" type="eb:MessageHeader" />
    <part name="ErrorList" type="eb:ErrorList" />
    <part name="Manifest" type="eb:Manifest" />
  </message>

  <message name="TypeBRequest">
    <part name="MessageHeader" type="eb:MessageHeader" />
    <part name="Manifest" type="eb:Manifest" />
  </message>
  <message name="TypeBResponse">
    <part name="MessageHeader" type="eb:MessageHeader" />
    <part name="Acknowledgement" type="eb:Acknowledgement" />
    <part name="ErrorList" type="eb:ErrorList" />
  </message>

<portType name="ebXML">
    <operation name="TypeARequest">
      <input message="eb:TypeARequest" />
      <output message="eb:TypeAResponse" />
    </operation>
    <operation name="TypeBRequest">
      <input message="eb:TypeBRequest" />
      <output message="eb:TypeBResponse" />
    </operation>
  </portType>

  <binding name="ebxmlhandler" type="eb:ebXML">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
                   style="document" />

    <operation name="TypeARequest">
      <soap:operation soapAction="ebXML" style="document" />
      <input>
      <soap:header
             namespace="http://www.ebxml.org/namespaces/messageHeader" />
        <soap:body
             namespace="http://www.ebxml.org/namespaces/messageHeader" />
        <mime:mimeXml part="Body" />
      </input>
```

[27] [wsdl] – Web Services Description Language – a W3C Technical Recommendation available at: http://www.w3c.org/TR/wsdl


```
      <output>
      <soap:header
           namespace="http://www.ebxml.org/namespaces/messageHeader" />
        <soap:body
           namespace="http://www.ebxml.org/ namespaces/messageHeader" />
        <mime:mimeXml part="Body" />
      </output>
    </operation>

    <operation name="TypeBRequest">
      <soap:operation soapAction="ebXML" style="document" />
      <input>
      <soap:header
            namespace="http://www.ebxml.org/namespaces/messageHeader" />
        <soap:body
 namespace="http://www.ebxml.org/namespaces/messageHeader"/>
        <mime:mimeXml part="Body" />
      </input>
      <output>
        <soap:header
namespace=" http://www.ebxml.org/namespaces/messageHeader"/>
      </output>
    </operation>
  </binding>

  <service name="ebxmlhandler">
    <port name="ebxml" binding="eb:ebxmlhandler">
      <http:address location="http://b2b.company.com/servlet/ebxml" />
    </port>
  </service>
</definitions>
```


5 OTA Update Messages

As described before, OTA update messages were designed with the following goals:

- Minimizing the size of a payload on the wire to represent an update transaction
- Defining an explicit representation for what has changed
- Defining a representation with a clear and simple conceptual model
- Creating a representation that is content-independent and general-purpose in nature so as to be reusable throughout future OTA specifications
- Providing a simple-to-implement "replace" option to allow developers to get simpler implementations running quickly - at the expense of the first 2 goals (representation of change and size of message) above.

5.1 Representing change in XML

Representing change is not a problem unique to the OTA. The approach presented in this specification assumes use of the following:

- library utilities[28] that can compare before and after images of a particular XML document and then generate a succinct representation of the differences (conceptually similar to a GNU *diff* utility)
- a representation for differences that is both human readable and understandable by automated tools
- library utilities that can take an XML document, apply a differences document and generate the same after image (conceptually similar to a GNU *patch* utility)

Applying an Update action involves a tree-to-tree comparison, similar to well-known operations outside the XML arena. This specification assumes that prior to sending an `<OTA_UpdateRQ>` message, the sender has used standard libraries to generate the differences between the 'before' and desired 'after' images of the XML document. Implementors may wish to consult documentation of algorithms for tree-to-tree comparison and correction in computer science publications[29], to provide the background for understanding this XML-specific approach.

5.2 Position Representation with XPath

The general concept of an OTA generic update request is to send a `<Position>` element followed by one or more operations to be applied at that position. XPath is a well-known W3C recommendation for representing a node in an XML document. The two best known applications of XPath are within XSL and XLink (themselves XML recommendations).

Within the OTA update representation, XPath is used to reference a specific element within a source document. This XPath is encoded within the *XPath* attribute in the `<Position>`

[28] an open source implementation of such a library is available from OTA member VM Systems, Inc. (see http://www.vmguys.com/vmtools/)

[29] For an excellent paper outlining the commonalties and differences between the best known algorithms see: "Tree-to-tree Correction for Document Trees", Technical Report 95-372 by David T. Barnard, Gwen Clarke and Nicholas Duncan (available from Queen's University, Kingston Ontario at: ftp://ftp.qucis.queensu.ca/pub/reports/1995-372.ps)

element. In OTA update messages, the *XPath* attribute within any given <Position> will always refer to an element, and this explanation therefore is restricted to that proper subset of XPath notation which refers to elements. The representation of <Position> followed by one or more operations is used, as updates will often consist of more than one operation to be performed at the same position. This representation will often be shorter than the alternative of embedding an XPath position explicitly within each operation.

It is important to note that when processing a differences document, insertion or deletion of nodes may invalidate subsequent XPath references. This is addressed in the section "Order of Representation and Application" (See Section 5.5).

5.3 Operands

In keeping with the design goals of minimal representation and explicit representation of change, changes are represented as occurring at four different levels, using the following operands:

- *Attribute* - performs an operation on the attribute *name* of the currently selected Element
- *Element* - performs an operation on the structure (i.e. content or children) of the currently selected element (but not its attributes)
- *Subtree* - allows for the grafting or pruning of an element which has its own children
- *Root* - this special attribute is ONLY used for a simple 'replace' alternative where an updated representation of the entire object is sent as opposed to sending incremental differences

In the case of <Element> and <Subtree> operands, insert operations are performed on children of the selected Element (selection is made via the <Position> *Xpath* attribute). Children are always specified via the *Child* attribute, which is a one-relative integer where children positions are specified left-to-right, with position one indicating the leftmost child.

5.4 Operations

Operations are specified on an operand via the Operation attribute. Operations are "*insert*", "*modify*" and "*delete*", though the "*modify*" operation is not applicable to the operand <Subtree>. The operand <Root> is the only operand to have the special "*replace*" operation.

The following table describes the result of each operation on its associated operand:

Operand	***Operation***	***Description***
Attribute	insert	The attribute *name* with value *value* will be added to selected element
Attribute	modify	The value of the attribute *name* will be changed to *value* on the selected element
Attribute	delete	The attribute *name* will be deleted from the selected element
Element	insert	A child element will be inserted beneath the currently selected element. This child will be inserted at one-relative position *Child* (see also Subtree insert)
Element	modify	The value of the currently selected element will be modified to the value specified. The text content of an element is removed by using a modify operation with a null replacement value.

Element	delete	The currently selected element is deleted. If this element has children of its own, these children will become children of their grandparent in the tree, with their position being equivalent to an insertion at the point previously occupied by the current element. Use Subtree delete to remove an Element and all its children from its currently selected parent
Subtree	insert	The subtree will be inserted beneath the currently selected element. This subtree will be inserted as a child at position *Child* (see also Element insert)
Subtree	delete	The subtree beginning at the currently selected element will be deleted (use Element delete if an element is to be removed, but its children preserved)
Root	replace	A replacement representation of the entire document object follows. This operation allows implementors to avoid the complexity of the difference representation and send a full 'after' image of the updated document object

Although it is possible to replace the entire tree by performing a <Subtree> *delete* operation on the root, it is not possible to then insert a new subtree as a replacement. The <Root> *replace* operation is provided for this purpose.

The syntax for an <OTA_UpdateRQ> is formally defined in the following schema fragment:

Schema 7 - <OTA_UpdateRQ>:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created and edited with 'vi' -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns="http://www.opentravel.org/OTA"
            targetNamespace="http://www.opentravel.org/OTA"
           elementFormDefault="qualified">

    <xs:annotation>
        <xs:documentation xml:lang="en">
OTA v2001C Specification - Generic OTA_UpdateRQ message definition
Copyright (C) 2001 Open Travel Alliance. All rights reserved.
        </xs:documentation>
    </xs:annotation>
    <xs:include schemaLocation="OTA_v2ent.xsd"/>
    <xs:include schemaLocation="OTA_POS.xsd"/>

    <xs:element name="OTA_UpdateRQ">
        <xs:annotation>
            <xs:documentation xml:lang="en">
This element represents the incremental changes to the business document
referred to by UniqueId.
            </xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="UniqueId"/>
                <xs:element ref="POS" minOccurs="0"/>
                <xs:element ref="Position" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attributeGroup ref="OTA_PayloadStdAttributes"/>
            <xs:attributeGroup ref="ReqRespVersion"/>
        </xs:complexType>
    </xs:element>

    <xs:element name="Position">
```



```
        <xs:complexType>
            <xs:choice>
                <xs:sequence>
                    <xs:element ref="Attribute" minOccurs="0" maxOccurs="unbounded"/>
                    <xs:element ref="Element" minOccurs="0" maxOccurs="unbounded"/>
                    <xs:element ref="Subtree" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:element ref="Root"/>
            </xs:choice>
            <xs:attribute name="XPath" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="Attribute">
        <xs:complexType>
            <xs:attribute name="Operation" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="insert"/>
                        <xs:enumeration value="modify"/>
                        <xs:enumeration value="delete"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="Name" type="xs:string" use="required"/>
            <xs:attribute name="Value" type="xs:string"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="Element">
        <xs:complexType>
            <xs:sequence>
                <xs:any/>
            </xs:sequence>
            <xs:attribute name="Operation" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="insert"/>
                        <xs:enumeration value="modify"/>
                        <xs:enumeration value="delete"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="Child" type="xs:string"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="Subtree">
        <xs:complexType>
            <xs:sequence>
                <xs:any/>
            </xs:sequence>
            <xs:attribute name="Operation" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="insert"/>
                        <xs:enumeration value="delete"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="Child" type="xs:string"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="Root">
        <xs:complexType>
            <xs:sequence>
                <xs:any/>
            </xs:sequence>
            <xs:attribute name="Operation" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="replace"/>
                    </xs:restriction>
                </xs:simpleType>
```



```
            </xs:attribute>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

5.5 Order of Representation and Application

The XPath notation is used to determine the position in an XML document where operations are to be applied. As that position is specified via an XPath, it is entirely possible that application of an operation will invalidate subsequent XPath expressions if care is not taken explicitly in the order of presentation.

An XML document is inherently a tree, and when considering an XML document, the most natural order for presentation is a depth-first pre-order traversal (this is the way in which an XML document is represented in XML notation). Unfortunately, that order is the one that is most likely to invalidate XPaths when applying differences in order.

Therefore, for purposes of representing differences in an update message, positions and operations are presented in an order which favors a post-order traversal[30]. This ensures that differences may be applied sequentially to a source document to transform it into a target without invalidating the XPath of any subsequent unprocessed difference.

A brief illustration will help make this point clearer. Assuming the following abstract source document:

```
<A>
    <B>
        <C/>
        <D>
            <E/>
        </D>
    </B>
    <F>
        <G>
            <H/>
        </G>
        <I>
            <J/>
        </I>
    </F>
</A>
```

The depth-first pre-order traversal presents elements in the following sequence:

```
<A><B><C><D><E><F><G><H><I><J>
```

However, a post-order traversal presents elements in the following alternate sequence:

```
<C><E><D><B><H><G><J><I><F><A>
```

When considering tree-traversals in the context of XML, perhaps the easiest way to think of it is as follows:

- depth-first pre-order traversal presents elements in the order that opening tags occur
- post-order traversal presents elements in the order that element closure occurs (i.e. follow the closing tags)

[30] It should be noted that the fastest known algorithms for tree-to-tree correction operate on a post-order traversal

An additional requirement for order is placed on repeating sequences of elements, as follows:

When operation(s) are to be performed on more than one element in a repeating sequence the `<Position>` for those elements in the sequence must be presented from right-to- left, i.e. from the largest index to the smallest.

By presenting and applying differences using a post-order traversal representation (and reverse index order for repeating sequences) operations applied to the current position cannot break the XPath to a subsequent position as later operations are deeper within the tree (closer to the root) down any given branch.

5.6 Update Examples

Given the difference representation above, the following examples illustrate this technique. These examples illustrate a chain of updates, showing before and after images at each step. All these changes are designed to be incremental, that is, the "after" image from the previous update becomes the "before" image for the next update.

Example 13 - initial "before" document image

```
<Profile>
    <Customer>
        <PersonName NameType="Default">
            <NamePrefix>Mr.</NamePrefix>
            <GivenName>George</GivenName>
            <MiddleName>A.</MiddleName>
            <Surname>Smith</Surname>
        </PersonName>
        <TelephoneInfo PhoneTech="Voice" PhoneUse="Work" >
            <Telephone>
                <AreaCityCode>206</AreaCityCode>
                <PhoneNumber>813-8698</PhoneNumber>
            </Telephone>
        </TelephoneInfo>
        <PaymentForm>
            ...
        </PaymentForm>
        <Address>
                <StreetNmbr POBox="4321-01">1200 Yakima St</StreetNmbr>
                <BldgRoom>Suite 800</BldgRoom>
                <CityName>Seattle</CityName>
                <StateProv PostalCode="98108">WA</StateProv>
                <CountryName>USA</CountryName>
        </Address>
    </Customer>
</Profile>
```

Let’s begin by making a simple change – we’ll update the profile to reflect a change in the customer’s area code from ‘206’ to ‘253’. The simplest way to implement this is to send the entire profile as a change, but this is a large message to send for a simple change of area code (see the example below):

Example 14 - update of AreaCode using <Root Operation="replace"/>

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://www.opentravel.org/OTA
OTA_UpdateRQ.xsd"
              ReqRespVersion="2">
  <UniqueId Type="Profile" Id="9876543210"
              URL="http://www.vmguys.com/OTAEngine" Instance="1" />
```



```
  <Position XPath="/Profile">
    <Root Operation="replace">
      <Profile xmlns="">
        <Customer>
          <PersonName NameType="Default">
            <NamePrefix>Mr.</NamePrefix>
            <GivenName>George</GivenName>
            <MiddleName>A.</MiddleName>
            <Surname>Smith</Surname>
          </PersonName>
          <TelephoneInfo PhoneTech="Voice" PhoneUse="Work">
            <Telephone>
              <AreaCityCode>253</AreaCityCode>
              <PhoneNumber>813-8698</PhoneNumber>
            </Telephone>
          </TelephoneInfo>
          <PaymentForm>...</PaymentForm>
          <AddressInfo>
            <Address>
              <StreetNmbr PO_Box="4321-01">1200 Yakima St</StreetNmbr>
              <BldgRoom>Suite 800</BldgRoom>
              <CityName>Seattle</CityName>
              <StateProv PostalCode="98108">WA</StateProv>
              <CountryName>USA</CountryName>
            </Address>
          </AddressInfo>
        </Customer>
      </Profile>
    </Root>
  </Position>
</OTA_UpdateRQ>
```

If we use the more succinct representation supported by OTA generic updates this same change of area code from ‘206’ to ‘253’ can be represented by the much more succinct and expressive message below:

Example 15 - change area code use <Element Operation="modify">:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
              ReqRespVersion="2">
    <UniqueId Type="Profile" Id="9876543210"
              URL="http://www.vmguys.com/OTAEngine" Instance="1" />
    <Position XPath="/Profile/Customer/TelephoneInfo/Telephone/AreaCityCode">
        <Element Operation="modify">253</Element>
    </Position>
</OTA_UpdateRQ>
```

Note that the operation `"modify"` replaces the PCDATA within the element. In order to delete the data, the update request is sent with empty content in the element. (A `"delete"` operation deletes the entire element.)

Example 16 - update of RelatedTraveler using <Subtree Operation="insert">

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
              ReqRespVersion="2">
  <UniqueId Type="Profile" Id="9876543210"
            URL="http://www.vmguys.com/OTAEngine" Instance="2" />
  <Position XPath="/Profile/Customer">
```



```
    <Subtree Operation="insert" Child="5">
      <RelatedTraveler Relation="Child">
        <PersonName>
          <GivenName>Devin</GivenName>
          <MiddleName>R.</MiddleName>
          <Surname>Smith</Surname>
        </PersonName>
      </RelatedTraveler>
    </Subtree>
    <Subtree Operation="insert" Child="6">
      <RelatedTraveler Relation="Child">
        <PersonName>
          <GivenName>Amy</GivenName>
          <MiddleName>E.</MiddleName>
          <Surname>Smith</Surname>
        </PersonName>
      </RelatedTraveler>
    </Subtree>
    <Subtree Operation="insert" Child="7">
      <RelatedTraveler Relation="Child">
        <PersonName>
          <GivenName>Alfred</GivenName>
          <MiddleName>E.</MiddleName>
          <Surname>Newman</Surname>
        </PersonName>
      </RelatedTraveler>
    </Subtree>
  </Position>
</OTA_UpdateRQ>
```

Example 17 - document image after <Subtree Operation="insert">

```
<Profile>
    <Customer>
        <PersonName NameType="Default">
            <NamePrefix>Mr.</NamePrefix>
            <GivenName>George</GivenName>
            <MiddleName>A.</MiddleName>
            <Surname>Smith</Surname>
        </PersonName>
        <TelephoneInfo PhoneTech="Voice" PhoneUse="Work"
            <Telephone>
                <AreaCityCode>253</AreaCityCode>
                <PhoneNumber>813-8698</PhoneNumber>
            </Telephone>
        </TelephoneInfo>
        <PaymentForm>
            ...
        </PaymentForm>
        <Address>
            <StreetNmbr POBox="4321-01">1200 Yakima St</StreetNmbr>
            <BldgRoom>Suite 800</BldgRoom>
            <CityName>Seattle</CityName>
            <StateProv PostalCode="98108">WA</StateProv>
            <CountryName>USA</CountryName>
        </Address>
        <RelatedTraveler Relation="Child"
            <PersonName>
                <GivenName>Devin</GivenName>
                <MiddleName>R.</MiddleName>
                <Surname>Smith</Surname>
            </PersonName>
        </RelatedTraveler>
        <RelatedTraveler Relation="Child"
            <PersonName>
                <GivenName>Amy</GivenName>
                <MiddleName>E.</MiddleName>
                <Surname>Smith</Surname>
            </PersonName>
```



```
        </RelatedTraveler>
        <RelatedTraveler Relation="Child"
            <PersonName>
                <GivenName>Alfred</GivenName>
                <MiddleName>E.</MiddleName>
                <Surname>Newman</Surname>
            </PersonName>
        </RelatedTraveler>
    </Customer>
</Profile>
```

Example 18 - update of document using <Subtree Operation="delete"/>

This operation will delete the first and third Related Travelers. (Note the required order of the operations):

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
              ReqRespVersion="2">
    <UniqueId Type="Profile" Id="9876543210"
              URL="http://www.vmguys.com/OTAEngine" Instance="3" />
    <Position XPath="/Profile/Customer/RelatedTraveler[3]">
        <Subtree Operation="delete" />
    </Position>
    <Position XPath="/Profile/Customer/RelatedTraveler[1]">
        <Subtree Operation="delete" />
    </Position>
</OTA_UpdateRQ>
```

Note: Use of the `<Delete>` element removes a designated element from the tree. Any children of the element removed will move up to the grandparent of the element. If the desired result is the removal of the element and all of its children, the element `<Subtree>` paired with the operation "`delete`" should be used, as in the example above.

Example 19 - Document image after <Subtree Operation="delete"/>

```
<Profile>
    <Customer>
        <PersonName NameType="Default">
            <NamePrefix>Mr.</NamePrefix>
            <GivenName>George</GivenName>
            <MiddleName>A.</MiddleName>
            <Surname>Smith</Surname>
        </PersonName>
        <TelephoneInfo PhoneTech="Voice" PhoneUse="Work" >
            <Telephone>
                <AreaCityCode>253</AreaCityCode>
                <PhoneNumber>813-8698</PhoneNumber>
            </Telephone>
        </TelephoneInfo>
        <PaymentForm>
            ...
        </PaymentForm>
        <AddressInfo>
            <Address>
                <StreetNmbr POBox="4321-01">1200 Yakima St</StreetNmbr>
                <BldgRoom>Suite 800</BldgRoom>
                <CityName>Seattle</CityName>
                <StateProv PostalCode="98108">WA</StateProv>
                <CountryName>USA</CountryName>
            </Address>
        </AddressInfo>
        <RelatedTraveler Relation="Child" >
```



```
                <PersonName>
                    <GivenName>Amy</GivenName>
                    <MiddleName>E.</MiddleName>
                    <Surname>Smith</Surname>
                </PersonName>
        </RelatedTraveler>
    </Customer>
</Profile>
```

Example 20 - Update document using <Attribute Operation="modify"/>

This examples changes the TelephoneInfo PhoneUse attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
              ReqRespVersion="2">
    <UniqueId Type="Profile" Id="9876543210"
              URL="http://www.vmguys.com/OTAEngine" Instance="4" />
    <Position XPath="/Profile/Customer/TelephoneInfo">
        <Attribute Name="PhoneUse" Operation="modify" Value="Home" />
    </Position>
</OTA_UpdateRQ>
```

Example 21 - Document image after <Attribute Operation="modify"/>

```
<Profile>
    <Customer>
        <PersonName NameType="Default">
            <NamePrefix>Mr.</NamePrefix>
            <GivenName>George</GivenName>
            <MiddleName>A.</MiddleName>
            <Surname>Smith</Surname>
        </PersonName>
        <TelephoneInfo PhoneTech="Voice" PhoneUse="Home">
            <Telephone>
                <AreaCityCode>253</AreaCityCode>
                <PhoneNumber>813-8698</PhoneNumber>
            </Telephone>
        </TelephoneInfo>
        <PaymentForm>
            ...
        </PaymentForm>
        <AddressInfo>
            <Address>
                <StreetNmbr PO_Box="4321-01">1200 Yakima St</StreetNmbr>
                <BldgRoom>Suite 800</BldgRoom>
                <CityName>Seattle</CityName>
                <StateProv PostalCode="98108">WA</StateProv>
                <CountryName>USA</CountryName>
            </Address>
        </AddressInfo>
        <RelatedTraveler Relation="Child">
                <PersonName>
                    <GivenName>Amy</GivenName>
                    <MiddleName>E.</MiddleName>
                    <Surname>Smith</Surname>
                </PersonName>
        </RelatedTraveler>
    </Customer>
</Profile>
```

Example 22 - Update document using <Attribute Operation="delete"/>

This operation will delete the PO_Box attribute of StreetNmbr.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
              ReqRespVersion="2">
    <UniqueId Type="Profile" Id="9876543210"
              URL="http://www.vmguys.com/OTAEngine" Instance="5" />
    <Position XPath="/Profile/Customer/AddressInfo/Address/StreetNmbr">
        <Attribute Name="PO_Box" Operation="delete" />
    </Position>
</OTA_UpdateRQ>
```

Example 23 - Document image after <Attribute Operation="delete"/>

```
<Profile>
    <Customer>
        <PersonName NameType="Default">
            <NamePrefix>Mr.</NamePrefix>
            <GivenName>George</GivenName>
            <MiddleName>A.</MiddleName>
            <Surname>Smith</Surname>
        </PersonName>
        <TelephoneInfo PhoneTech="Voice" PhoneUse="Home">
            <Telephone>
                <AreaCityCode>253</AreaCityCode>
                <PhoneNumber>813-8698</PhoneNumber>
            </Telephone>
        </TelephoneInfo>
        <PaymentForm>
            ...
        </PaymentForm>
        <AddressInfo>
            <Address>
                <StreetNmbr>1200 Yakima St</StreetNmbr>
                <BldgRoom>Suite 800</BldgRoom>
                <CityName>Seattle</CityName>
                <StateProv PostalCode="98108">WA</StateProv>
                <CountryName>USA</CountryName>
            </Address>
        </AddressInfo>
        <RelatedTraveler Relation="Child">
                <PersonName>
                    <GivenName>Amy</GivenName>
                    <MiddleName>E.</MiddleName>
                    <Surname>Smith</Surname>
                </PersonName>
        </RelatedTraveler>
    </Customer>
</Profile>
```

Example 24 - Compound update using multiple operations

These compound operations will insert the Gender attribute on Customer, delete the MiddleName, and insert NamePrefix on the Related Traveler:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
              ReqRespVersion="2">
    <UniqueId Type="Profile" Id="9876543210"
              URL="http://www.vmguys.com/OTAEngine" Instance="6" />
    <Position XPath="/Profile/Customer/RelatedTraveler/PersonName">
        <Element Operation="insert" Child="1">
```



```
            <NamePrefix xmlns="">Ms.</NamePrefix>
        </Element>
    </Position>
    <Position XPath="/Profile/Customer/PersonName/MiddleName">
        <Element Operation="delete" />
    </Position>
    <Position XPath="/Profile/Customer">
        <Attribute Name="Gender" Operation="insert" Value="Male" />
    </Position>
</OTA_UpdateRQ>
```

Example 25 - document image after compound update

```
<Profile>
    <Customer Gender="Male">
        <PersonName NameType="Default">
            <NamePrefix>Mr.</NamePrefix>
            <GivenName>George</GivenName>
            <Surname>Smith</Surname>
        </PersonName>
        <TelephoneInfo PhoneUse="Home">
            <Telephone PhoneTech="Voice" >
                <AreaCityCode>253</AreaCityCode>
                <PhoneNumber>813-8698</PhoneNumber>
            </Telephone>
        </TelephoneInfo>
        <PaymentForm>
            ...
        </PaymentForm>
        <Address>
            <StreetNmbr>1200 Yakima St</StreetNmbr>
            <BldgRoom>Suite 800</BldgRoom>
            <CityName>Seattle</CityName>
            <StateProv PostalCode="98108">WA</StateProv>
            <CountryName>USA</CountryName>
        </Address>
        <RelatedTraveler Relation="Child">
            <PersonName>
                <NamePrefix>Ms.</NamePrefix>
                <GivenName>Amy</GivenName>
                <MiddleName>E.</MiddleName>
                <Surname>Smith</Surname>
            </PersonName>
        </RelatedTraveler>
    </Customer>
</Profile>
```

5.7 Validation of Update Messages

The update request message is a valid message within its own structure, but since it identifies only a portion of the content of an XML tree, it cannot be validated in the context of the business schema. Validation at the level of the business schema must occur after the update has been completed. Implementors may wish to validate the image of the document before changes have been applied and again after the changes have been applied in order to ascertain that the desired result has been obtained and is valid within the business context.

5.8 The Simple "Replace" verb

The Replace infrastructure verb defines an operation that updates an existing record by replacing the existing information with a complete overlay image of the document as it would appear after changes are applied. The Replace verb does not require a Read request to obtain an image of the

current record prior to sending the message to replace it with the information being transmitted, although this may be desirable from the standpoint of good business practice.

The "replace" verb allows implementors greater flexibility in choosing how they wish to perform updates of information, since difference representation may be bypassed and a simple replacement image of the document object to be updated is sent. This reduces the complexity of handling updates for some implementations, but at the expense of size of the messages.

Example 26 - Document update using <Root Operation="replace">

```
<OTA_UpdateRQ ReqRespVersion="2">
    <UniqueId  URL="http://vmguys.com/OTAEngine/"
                   Type="Profile"
                   Id="12345678"
                   Instance="7"/>
    <Position XPath="/Profile">
        <Root Operation="replace">
            <Profile>
                ...
                <!-- include entired updated 'after' image here -->
                ...
            </Profile>
        </Root>
    </Position>
</OTA_UpdateRQ>
```

5.9 OTA_UpdateRS – Responding to a generic OTA_UpdateRQ message

An OTA update response follows the design pattern for OTA responses and is formally defined by the following schema fragment:

Schema 8 - <OTA_UpdateRS>:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created and edited with 'vi' -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.opentravel.org/OTA"
           targetNamespace="http://www.opentravel.org/OTA"
            elementFormDefault="qualified">

    <xs:annotation>
        <xs:documentation xml:lang="en">
OTA v2001C Specification - Generic OTA_UpdateRS message definition
Copyright (C) 2001 Open Travel Alliance. All rights reserved.
The message id of the message being responded to is in the ebXML header.
        </xs:documentation>
    </xs:annotation>
    <xs:include schemaLocation="OTA_v2ent.xsd"/>

    <xs:element name="OTA_UpdateRS">
        <xs:complexType>
            <xs:choice>
                <xs:sequence>
                    <xs:element ref="Success"/>
                    <xs:element ref="Warnings" minOccurs="0"/>
                    <xs:element ref="UniqueId"/>
                </xs:sequence>
                <xs:element ref="Errors"/>
            </xs:choice>
            <xs:attributeGroup ref="OTA_PayloadStdAttributes"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```


6 Service and Action Mappings

This section defines all the *Action* values which are permitted within each particular *Service*. In all cases there are RECOMMENDED 'class of delivery' semantics (see section 4.3 for definition of these).

Each defined OTA message is assigned its own *Action* within a specific *Service*. The generic messages are assigned as a permissible *Action* within each *Service* for which they are applicable. Working groups are strongly encouraged to make use of the generic messages wherever possible.

Within OTA *Services* many *Actions* are part of request/response pairs. When this is the case the corresponding expected response or type of request is indicated.

The criteria for grouping related *Actions* into the same *Service* are as follows:

- All actions within a service are logically related and are likely to be implemented by the same application system
- Generic messages are defined as actions on each service to which they apply
- Request/response pairs always belong to the same service, though in some circumstances they may belong to more than one service (e.g. generic messages such as OTA_ReadRQ).

6.1 The Session Service

The following table defines the actions available within this service:

Session Action	*Class of Delivery*	*Response Expected*	*In Response To*
CreateRQ	Type-A	CreateRS	-
CreateRS	Type-A	-	CreateRQ
CloseRQ	Type-A	CloseRS	-
CloseRS	Type-A	-	CloseRQ
ErrorRS	Type-A/Type-B	-	Any OTA RQ

During Session creation parties MUST establish which services and actions are supported via a <ServicesSupported> element on the <SessionControlResponse> message. This process is outlined in the following section.

6.2 The Profile Service

The following table defines the actions available within this service:

Profile Action	*Class of Delivery*	*Response Expected*	*In Response To*
OTA_CreateProfileRQ	Type-A	OTA_CreateProfileRS	-
OTA_CreateProfileRS	Type-A	-	OTA_CreateProfileRQ
OTA_ReadRQ	Type-A	OTA_ReadProfileRS	-
OTA_ReadProfileRS	Type-A	-	OTA_ReadRQ
OTA_UpdateRQ	Type-A	OTA_UpdateRS	-

OTA_UpdateRS	Type-A	-	OTA_UpdateRQ
OTA_DeleteRQ	Type-A	OTA_DeleteRS	-
OTA_DeleteRS	Type-A	-	OTA_DeleteRQ

6.3 The VehicleBooking Service

The following table defines the actions available within this service:

VehicleBooking Action	*Class of Delivery*	*Response Expected*	*In Response To*
OTA_VehAvailRateRQ	Type-A	OTA_VehAvailRateRS	-
OTA_VehAvailRateRS	Type-A	-	OTA_VehAvailRateRQ
OTA_VehResRQ	Type-A	OTA_VehResRS	-
OTA_VehResRS	Type-A	-	OTA_VehResRQ
OTA_CancelRQ	Type-A	OTA_CancelRS	
OTA_CancelRS	Type-A	-	OTA_CancelRQ

6.4 The AirBooking Service

The following table defines the actions available within this service:

AirBooking Action	*Class of Delivery*	*Response Expected*	*In Response To*
OTA_SpecificFlightAvailRQ	Type-A	OTA_SpecificFlightAvailRS	-
OTA_SpecificFlightAvailRS	Type-A	-	OTA_SpecificFlightAvailRQ
OTA_SpecificAirlineAvailRQ	Type-A	OTA_SpecificAirlineAvailRS	-
OTA_SpecificAirlineAvailRS	Type-A	-	OTA_SpecificAirlineAvailRQ
OTA_MultipleAirlineAvailRQ	Type-A	OTA_MultipleAirlineAvailRS	-
OTA_MultipleAirlineAvailRS	Type-A	-	OTA_MultipleAirlineAvailRS
OTA_AirBookRQ	Type-A	OTA_AirBookRS	-
OTA_AirBookRS	Type-A	-	OTA_AirBookRQ
OTA_CancelRQ	Type-A	OTA_CancelRS	
OTA_CancelRS	Type-A	-	OTA_CancelRQ

6.5 The TravelInsurance Service

The following table defines the actions available within this service:

TravelInsurance Action	*Class of Delivery*	*Response Expected*	*In Response To*
OTA_InsuranceQuoteRQ	Type-A	OTA_InsuranceQuoteRS	-
OTA_InsuranceQuoteRS	Type-A	-	OTA_InsuranceQuoteRQ

OTA_InsuranceBookRQ	Type-A	OTA_InsuranceBookRS	-
OTA_InsuranceBookRS	Type-A	-	OTA_InsuranceBookRQ

6.6 The HotelBooking Service

The following table defines the actions available within this service:

HotelBooking Action	*Class of Delivery*	*Response Expected*	*In Response To*
OTA_HotelSearchRQ	Type-A	OTA_HotelSearchRS	-
OTA_HotelSearchRS	Type-A	-	OTA_HotelSearchRQ
OTA_HotelAvailRQ	Type-A	OTA_HotelAvailRS	-
OTA_HotelAvailRS	Type-A	-	OTA_HotelAvailRQ
OTA_HotelResRQ	Type-A	OTA_HotelResRS	-
OTA_HotelResRS	Type-A	-	OTA_HotelAvailRQ
OTA_CancelRQ	Type-A	OTA_CancelRS	
OTA_CancelRS	Type-A	-	OTA_CancelRQ

6.7 The HotelResNotification Service[31]

The following table defines the actions available within this service:

HotelResNotification Action	*Class of Delivery*	*Response Expected*	*In Response To*
OTA_HotelResNotifRQ	Type-A	OTA_HotelResNotifRS	-
OTA_HotelResNotifRS	Type-A	-	OTA_HotelResNotifRQ
OTA_GetMsgRQ	Type-A	OTA_GetMsgRS	-
OTA_GetMsgRS	Type-A	-	OTA_GetMsgRQ
OTA_GetMsgInfoRQ	Type-A	OTA_GetMsgInfoRS	-
OTA_GetMsgInfoRS	Type-A	-	OTA_GetMsgInfoRQ

6.8 The HotelPropertyInformation Service[32]

The following table defines the actions available within this service:

HotelPropertyInformation Action	*Class of Delivery*	*Response Expected*	*In Response To*
OTA_CommNotifRQ	Type-A	OTA_CommNotifRS	-

[31] Some of the messages on this service are ideal candidates for a single type-B notification message rather than request/response pair.

[32] Some of the messages on this service are ideal candidates for a single type-B notification message rather than request/response pairs.

OTA_CommNotifRS	Type-A	-	OTA_CommNotifRQ
OTA_StayInfoNotifRQ	Type-A	OTA_StayInfoNotifRS	-
OTA_StayInfoNotifRS	Type-A	-	OTA_StayInfoNotifRQ
OTA_StatisticsNotifRQ	Type-A	OTA_StatisticsNotifRS	-
OTA_StatisticsNotifRS	Type-A	-	OTA_StatisticsNotifRQ
OTA_StatisticsRQ	Type-A	OTA_StatisticsRS	-
OTA_StatisticsRS	Type-A	-	OTA_StatisticsRQ
OTA_GetMsgRQ	Type-A	OTA_GetMsgRS	-
OTA_GetMsgRS	Type-A	-	OTA_GetMsgRQ
OTA_GetMsgInfoRQ	Type-A	OTA_GetMsgInfoRS	-
OTA_GetMsgInfoRS	Type-A	-	OTA_GetMsgInfoRQ

6.9 The MeetingProfile Service

The following table defines the actions available within this service:

MeetingProfile Action	*Class of Delivery*	*Response Expected*	*In Response To*
OTA_CreateMeetingProfileRQ	Type-A	OTA_CreateMeetingProfileRS	-
OTA_CreateMeetingProfileRS	Type-A	-	OTA_CreateMeetingProfileRQ

6.10 The PackageBooking Service

The following table defines the actions available within this service:

PackageBooking Action	*Class of Delivery*	*Response Expected*	*In Response To*
OTA_PkgAvailRQ	Type-A	OTA_PkgAvailRS	-
OTA_PkgAvailRS	Type-A	-	OTA_PkgAvailRQ
OTA_PkgBookRQ	Type-A	OTA_PkgBookRS	-
OTA_PkgBookRS	Type-A	-	OTA_PkgBookRQ

6.11 The GolfTeeTimes Service

The following table defines the actions available within this service:

Session Action	*Class of Delivery*	*Response Expected*	*In Response To*
OTA_CourseSearchRQ	Type-A	OTA_CourseSearchRS	-
OTA_CourseSearchRS	Type-A	-	OTA_CourseSearchRQ
OTA_CourseAvailRQ	Type-A	OTA_CourseAvailRS	-
OTA_CourseAvailRS	Type-A	-	OTA_CourseAvailRQ

OTA_CourseResRQ	Type-A	OTA_CourseResRS	-
OTA_CourseResRS	Type-A	-	OTA_CourseResRQ

6.12 Determining Services Supported

A *CreateRS* action on the *Session* service MUST be accompanied by a <ServicesSupported> within a <SessionControlResponse> document located in the payload container of the message. Upon receipt of Session *CreateRQ* request an implementation responds with a *CreateRS* action message with a status attribute status=”Accepted”. Here is an example of a <SessionControlResponse> with a <ServicesSupported> element:

Example 27 - sample <SessionControlResponse> payload:

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

<SessionControlResponse xmlns="http://www.opentravel.org/OTA"
    xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opentravel.org/OTA SessionControlResponse.xsd"
    Version="1" status=”Approved”>
  <OTA_Version>2001C<OTA_Version>
  <ConversationId>20011027081907-862@host.imacompany.com</ConversationId>

  <ServicesSupported>
    <Service Type="OTA" Name="HotelBooking">
        <Action Name="OTA_HotelSearchRQ" Version="2"/>
        <Action Name="OTA_HotelAvailRQ" Version="2"/>
        <Action Name="OTA_HotelResRQ" Version="2">
            <Extension Name="VMStayAmenities" Version="1" Required="no" />
        </Action>
    </Service>

    <Service Type="OTA" Name="Profile">
        <Action Name="OTA_CreateProfileRQ" Version="2">
            <Extension Name="DeltaSkymiles" Version="1" Required="no" />
        </Action>
        <Action Name="OTA_ReadRQ" Version="2"/>
        <Action Name="OTA_UpdateRQ" Version="2"/>
    </Service>

    <Service Type="OTA" Name="Session">
        <Action Name="CreateRQ" Version="1" />
        <Action Name="CloseRQ" Version="1" />
    </Service>
  </ServicesSupported>
</SessionControlResponse>
```

In this example, an implementation supports two services (*HotelBooking* and *Profile*) in addition to the *Session* service itself.

6.12.1 The <ServicesSupported> element

The *ServicesSupported* element has the following attributes

- *Version* : The version of this message. Currently a value of 1.

The *ServicesSupported* element MUST contain a sequence of at least one *Service* element. N.B. *Services* and *Actions* included in a *ServicesSupported* message are always from the perspective of the system sending the message i.e. the message formally defines which services the implementation provides and which actions upon each service it implements and will accept

messages for. An implementation specifies the request actions it will accept, but does not need to specify the response actions it will generate (these are implied based upon the Service definition tables earlier in this section).

6.12.2 The <Service> element

The *Service* element has the following required attributes:

- *Type:* A value of "OTA". This is the value that MUST be used in the *type* attribute of the <*eb:Service*> element on the <*eb:MessageHeader*>
- *Name*: The service name. This is the value which MUST be used as the content of the <*eb:Service*> element on the <*eb:MessageHeader*>

The *Service* element MUST contain a sequence of at least one *Action* element.

6.12.3 The <Action> element

The *Action* element has the following required attributes:

- *Name*: The action name. This is the value that MUST be used as the content of the <*eb:Action*> element on the <*eb:MessageHeader*>. This name also usually corresponds directly to one of the defined OTA message types (except in the case of the *Session* service)
- *Version*: The version of the OTA message supported. Implementations that support more than one version of a given message can indicate this by including additional <*Action*> elements with the same *Name* but different values for *Version*

The *Action* element MAY contain a sequence of *Extension* elements.

6.12.4 The <Extension> element

An Extension indicates a bilaterally agreed upon message extension between two trading partners, using the <`TPA_Extension`> mechanism. The *Extension* element has the following attributes:

- *Name*: The name of an extension (required)
- *Version*: the version of that extension (required)
- *Required*: a yes/no value indicating whether the extension is mandatory (defaults to 'no')

6.12.5 The ServicesSupported Schema

The following schema fragment formally defines the <`ServicesSupported`> element:

Schema 9- ServicesSupported.xsd

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

<!-- Created and edited with 'vi' -->

<xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.opentravel.org/OTA"
           targetNamespace="http://www.opentravel.org/OTA"
           elementFormDefault="qualified">

    <xs:annotation>
```



```
        <xs:documentation xml:lang="en">
OTA v2001C Specification - ServicesSupported message definition
Copyright (C) 2001 Open Travel Alliance. All rights reserved.
        </xs:documentation>
    </xs:annotation>

    <xs:simpleType name="versionNumber">
        <xs:annotation>
            <xs:documentation xml:lang="en">
All OTA version numbers are unsigned integers in the range 1..9999.
Version numbers should begin with the value '1' and be incremented
each time a message is revised.
            </xs:documentation>
        </xs:annotation>

        <xs:restriction base="xs:unsignedShort">
            <xs:minInclusive value="1"/>
            <xs:maxInclusive value="9999"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:element name="ServicesSupported">
        <xs:annotation>
            <xs:documentation xml:lang="en">
Generated and sent during OTA session negotiation to indicate the services
a given implementation supports.
            </xs:documentation>
        </xs:annotation>

        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Service" minOccurs="1" maxOccurs="unbounded" />
            </xs:sequence>
            <xs:attribute name="Version" type="versionNumber" />
        </xs:complexType>
    </xs:element>

    <xs:element name="Service">
        <xs:annotation>
            <xs:documentation xml:lang="en">
Indicates a defined OTA Service which this implementation supports,
providing one or more of the standard actions defined on that service.
            </xs:documentation>
        </xs:annotation>

        <xs:complexType>
            <xs:sequence>
                <xs:element ref="Action" minOccurs="1" maxOccurs="unbounded" />
            </xs:sequence>
            <xs:attribute name="Type" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="OTA"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="Name" type="xs:string" use="required" />
        </xs:complexType>
    </xs:element>

    <xs:element name="Action">
        <xs:annotation>
            <xs:documentation xml:lang="en">
Indicates an Action upon a defined OTA Service which this implementation
supports. For most OTA Services the  Action name is equivalent to the action
verb and root tag of the primary payload document.
            </xs:documentation>
        </xs:annotation>

        <xs:complexType>
            <xs:sequence>
```

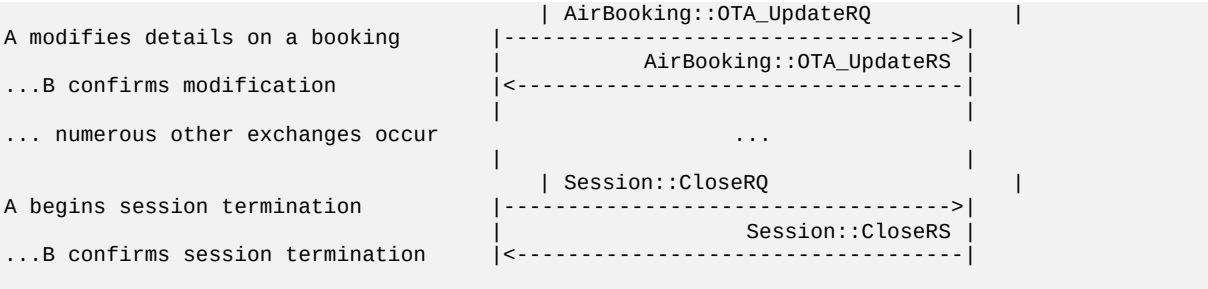
```
            <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
        <xs:attribute name="Name" type="xs:string" use="required" />
        <xs:attribute name="Version" type="versionNumber" use="required" />
    </xs:complexType>
</xs:element>

<xs:element name="Extension">
    <xs:annotation>
        <xs:documentation xml:lang="en">
Indicates an extension point within a standard OTA action that this
implementation understands.
        </xs:documentation>
    </xs:annotation>

    <xs:complexType>
        <xs:attribute name="Name" type="xs:string" use="required" />
        <xs:attribute name="Version" type="versionNumber" use="required" />
        <xs:attribute name="Required" default="no">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="yes"/>
                    <xs:enumeration value="no"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
</xs:element>

</xs:schema>
```

6.13 Sample Session Message Flow

The example below shows the messages exchanged during a sample session, including session initiation and termination.

Example 28 - sample session

```
Comments:                          System A                          System B
=========                          ========                         ========
A initiates SSL connection with B     |HTTP POST with Authorization        |
Containing Username/password data     |                                    |
While creating a new session          | Session::CreateRQ                  |
A initiates a new session with B      |----------------------------------->|
                                      |  Session::CreateRS status=Accepted |
...B accepts the session              |<-----------------------------------|
                                      |                                    |
                                      | AirBooking::OTA_MultiAirAvailRQ    |
A checks air availability             |----------------------------------->|
                                      |    AirBooking::OTA_MultiAirAvailRS |
...B responds with availability       |<-----------------------------------|
                                      |                                    |
                                      | AirBooking::OTA_AirBookRQ          |
A makes an air booking request        |----------------------------------->|
                                      |          AirBooking::OTA_AirBookRS |
...B responds                         |<-----------------------------------|
                                      |                                    |
                                      | AirBooking::OTA_MultiAirAvailRQ    |
A makes a different avail. request    |----------------------------------->|
                                      |    AirBooking::OTA_MultiAirAvailRS |
...B responds                         |<-----------------------------------|
                                      |                                    |
                                      | AirBooking::OTA_CancelRQ           |
A initiates cancellation of a booking |----------------------------------->
                                      |           AirBooking::OTA_CancelRS |
...B confirms cancellation            |<-----------------------------------|
                                      |                                    |
```

7 Summary of Infrastructure Changes

This section is informative and provides a description of the changes in infrastructure from previously published OTA standards.

- Movement from published DTDs to published XML schemas for specification of message syntax and the underlying reference model
- A mapping to ebXML 1.0 Transport, Routing and Packaging as a RECOMMENDED infrastructure substrate for OTA implementations
- Elimination of the previously mandatory <Control> payload in favor of equivalent capabilities provided by underlying infrastructure
- Definition of the Service/Action concept and mapping of each defined OTA message as an <Action> on at least one <Service>
- Concrete definition of the notion of OTA sessions and the definition of a Session <Service> which controls session setup and termination
- Elimination of the previous non-versioned VersionDiscovery mechanism in favor of <ServicesSupported> negotiation during session establishment
- Elimination of the <Sendby> semantics allowed for in the previous <Control> section. OTA STRONGLY RECOMMENDS all message exchanges occur within the context of a valid session
- Support for <ReplyTo> <CCTo> and the OrigBodyReq attribute has been dropped from this specification. These elements, or a mechanism providing similar functionality will be considered during a future specification when a valid use-case dictates (this kind of functionality may be provided by a publish-subscribe messaging model)

8 Appendix - Utilizing ebXML v2.0 in OTA Solutions[33]

The recent release of ebXML TR&P v2.0 offers several additional advantages when used to support the exchange of OTA-compliant documents. The following sections will describe how this updated specification may best be mapped to OTA.

What did / did not change from ebXML V1.0 to ebXML V2.0:

A. The usage of MIME is unchanged since ebXML1.0 relied on that specification to define the packaging and version 2.0 added nothing but clarification to the existing documentation. No impact for OTA.

B. The Manifest payload remains unchanged except for the obvious update to the eb:version attribute value, which is now "2.0" for all top-level ebXML SOAP extension element, whereas this value had been "1.0". No impact for OTA.

C. The QualityOfServiceInfo element has been eliminated and its constituent aspects have been separated out. This has the most significant (positive) impact on the existing OTA specification.

D. The Via element: has been eliminated and replaced with a SyncReply element . No impact for OTA.

The Impact to the OTA 2001C Specification:

The impact is minimal, but quite favorable. With the replacement of the QualityOfServiceInfo element by a set of "reliable Message Parameters", ebXML V2.0 better addresses the needs of the OTA infrastructure than its precdecessor. Specifically, until this point, OTA TypeA messages were forced to use deliverySemantics="BestEffort" and there was no futher QualityOfService support available. Now by specifying:

DuplicateElimination

be set for all such messages, the receiving OTA process no longer has to cache all received MessageIDs to perform such checking on its own, as it will be automatically done by the Mesasge Service Handler.

When sending OTA TypeB messages (guaranteed ordered once-only delivery), an OTA-compliant application would need to ensure that both:

DuplicateElimination

ACKRequested

parameters were specificed. Additional Message Parameters controlling retry timeout values and message persistence may also be made available through the ebXML TR&P provider. These may now optionally be used to tailor an OTA session to the idiosyncrasies of the physical connection over which it is established.

[33] The OpenTravel Alliance will do a more comprehensive revision to map to the most current ebXML MS version in a future specification.

