



OpenTravel™ Alliance
2001C Infrastructure Specification
(Draft Copy - Public Review)

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 describes 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.

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

Copyright, OpenTravel Alliance (OTA), 2001.

All rights reserved, subject to the User License set out below.

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:*

- 77 **3.1.** ANY DOCUMENTATION PROVIDED PURSUANT TO THIS NON-
78 EXCLUSIVE LICENSE AGREEMENT IS PROVIDED “AS IS” AND
79 NEITHER OTA NOR ANY PERSON WHO HAS CONTRIBUTED TO THE
80 CREATION, REVISION, OR MAINTENANCE OF THE DOCUMENTATION
81 MAKES ANY REPRESENTATIONS OR WARRANTIES, EXPRESS OR
82 IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF
83 MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE.
- 84 **3.1.** Neither OTA nor any person who has contributed to the creation, revision or
85 maintenance of the documentation makes any representations or warranties,
86 express or implied, that the use of the documentation or software will not infringe
87 any third party copyright, patent, patent application, trademark, trademark
88 application, or other right.
- 89 **3.2.** Neither OTA nor any person who has contributed to the creation, revision, or
90 maintenance of the documentation shall be liable for any direct, indirect, special
91 or consequential damages or other liability arising from any use of the
92 documentation or software. You agree not to file a lawsuit, make a claim, or take
93 any other formal or informal action against OTA based upon Your acquisition,
94 use, duplication, distribution, or exploitation of the Documentation.
- 95 **3.3.** The foregoing liability limitations shall apply to and be for the benefit of OTA,
96 any person who has contributed to the creation, revision or maintenance of the
97 documentation, and any member of the board of directors, officer, employee,
98 independent contractor, agent, partner, or joint venturer of OTA or such person.
- 99 **4. No Update Obligation.** Nothing in this Agreement shall be interpreted as requiring OTA to
100 provide You with updates, revisions or information about any development or action
101 affecting the Documentation.
- 102 **5. No Third Party Beneficiary Rights.** This Agreement shall not create any third party
103 beneficiary rights.
- 104 **6. Application to Successors and Assignees.** This Agreement shall apply to the use of the
105 Documentation by any successor or assignee of the Licensee.
- 106 **7. Term.** The term of this license is perpetual subject to Your compliance with the terms of this
107 Agreement, but OTA may terminate this License Agreement immediately upon your breach
108 of this Agreement and, upon such termination you will cease all use duplication, distribution,
109 and/or exploitation of the Documentation in any manner.
- 110 **8. Interpretation and Choice of Forum.** The law of the Commonwealth of Virginia and any
111 applicable Federal law shall govern this Agreement. Any disputes arising from or relating to
112 this Agreement shall be resolved in the courts of the Commonwealth of Virginia, including
113 Federal courts. You consent to the jurisdiction of such courts and agree not to assert before
114 such courts any objection to proceeding in such forum.
- 115 **9. Acceptance.** Your acceptance of this License Agreement will be indicated by your
116 affirmative acquisition, use, duplication, distribution, or other exploitation of the
117 Documentation. If you do not agree to these terms, please cease all use of the
118 Documentation now.
- 119 **10. Questions.** Questions about the Agreement should be directed to www.opentravel.org.
- 120

121 **Table of Contents**

122	1	About this Document	7
123	1.1	Intended Audience	7
124	1.2	Definitions and Conventions	7
125	1.3	Relationship with previous OTA standards.....	7
126	1.4	Relationship with ebXML standards.....	7
127	1.5	Use of Namespaces in this specification	8
128	2	OTA XML Best Practices.....	9
129	2.1	XML Standard Specifications.....	9
130	2.2	Best Practices.....	10
131	2.2.1	Scope	10
132	2.2.2	XML Component Parts and Roles.....	10
133	2.3	OTA XML Guidelines	10
134	2.3.1	Tag Naming Conventions (I).....	10
135	2.3.1.1	XML Tag Names (I-1).....	10
136	2.3.1.2	XML Tag Names (I-2).....	11
137	2.3.1.3	XML Tag Names (I-3).....	11
138	2.3.1.4	XML Tag Names (I-4).....	11
139	2.3.1.5	XML Tag Names (I-5).....	11
140	2.3.1.6	XML Tag Names (I-6).....	11
141	2.3.1.7	XML Tag Names (I-7).....	12
142	2.3.2	Elements vs. Attributes (II).....	13
143	2.3.2.1	Elements vs. Attributes (II-1).....	13
144	2.3.2.2	Elements vs. Attributes (II-2).....	13
145	2.3.2.3	Elements vs. Attributes (II-3).....	13
146	2.3.3	DTD vs. XML Schema (III).....	14
147	2.3.3.1	DTD vs. XML Schema (III-1)	14
148	2.3.4	Global vs Local Element Types and Elements/Attributes (IV).....	14
149	2.3.4.1	Global vs Local Element Types and Elements/Attributes (IV-1).....	14
150	2.3.4.2	Global vs Local Element Types and Elements/Attributes (IV-2).....	14
151	2.3.4.3	Global vs Local Element Types and Elements/Attributes (IV-3).....	15
152	2.3.5	Namespaces (V)	15
153	2.3.5.1	Namespaces (V-1)	15
154	2.3.5.2	Namespaces (V-2)	16
155	2.3.5.3	Namespaces (V-3)	16
156	2.3.6	Versioning XML Schemas (VI)	16
157	2.3.6.1	Versioning XML Schemas (VI-1).....	16
158	2.3.6.2	Versioning XML Schemas (VI-2).....	17
159	2.3.7	XML Markup – General (VII).....	17
160	2.3.7.1	XML Markup - General (VII-1).....	17
161	2.3.7.2	XML Markup - General (VII-2).....	18
162	2.3.7.3	XML Markup - General (VII-3).....	18
163	2.3.8	OTA General (VIII).....	18
164	2.3.8.1	OTA General (VIII-1)	18
165	2.3.8.2	OTA General (VIII-2)	19
166	2.3.8.3	OTA General (VIII-3)	19
167	2.3.8.4	OTA General (VIII-4)	19
168	2.3.8.5	OTA General (VIII-5)	19
169	2.4	Response Message Design	20
170	2.4.1	Standard Payload Attributes.....	20

171	2.4.1.1 Mapping OTA Payload Attributes onto ebXML MS Headers.....	21
172	2.4.2 Design Patterns for Response Messages.....	21
173	3 Generic Messages and the Service/Action Model.....	25
174	3.1 The Service/Action Concept.....	25
175	3.1.1 Service/Action Message Mappings.....	25
176	3.2 Unique Identifiers within OTA Messages.....	26
177	3.2.1 Examples of unique identifiers.....	27
178	3.3 Generic Infrastructure Messages.....	28
179	3.3.1 Create messages.....	28
180	3.3.2 Generic Read message.....	29
181	3.3.3 Generic Update message.....	30
182	3.3.4 Generic Delete message.....	31
183	3.3.5 Generic Cancel Request.....	34
184	3.3.5.1 Consequences of Canceling.....	34
185	3.3.5.2 Read Request Prior to Cancel.....	35
186	3.3.5.3 Security considerations.....	35
187	3.3.5.4 OTA Cancel Messages.....	36
188	3.3.5.5 Confirmation of Cancellation.....	38
189	4 OTA Infrastructure.....	41
190	4.1 Architecture Overview.....	41
191	4.1.1 Reference Model.....	42
192	4.1.2 Transport Protocols.....	43
193	4.1.3 Logging.....	44
194	4.1.4 Auditing.....	44
195	4.2 Message Structure and Packaging.....	45
196	4.2.1 The 'Unit-of-work' Concept.....	46
197	4.2.2 Packaging a single unit-of-work.....	46
198	4.3 Classes of Message Delivery.....	47
199	4.3.1 Historical use within the travel industry.....	47
200	4.3.1.1 Type A.....	47
201	4.3.1.2 Type B.....	48
202	4.3.2 EbXML Classes of Delivery.....	48
203	4.4 EbXML Header Document.....	48
204	4.4.1 OTA Subset of an ebXML Header Document.....	48
205	4.4.1.1 ErrorList Element.....	49
206	4.4.1.2 Acknowledgement Element.....	50
207	4.4.1.3 Via Element.....	50
208	4.4.1.4 MessageHeader Element.....	51
209	4.5 SOAP Body Elements.....	54
210	4.6 EbXML Collaboration Protocol Profile.....	55
211	4.7 EbXML Header Examples.....	56
212	4.7.1 Type A Request Message.....	56
213	4.7.2 Type B Request Message.....	56
214	4.7.3 Type A Response Example.....	57
215	4.7.4 Type B Response Example.....	58
216	4.7.4.1 Reliable Messaging.....	58
217	4.7.4.2 Once and Only Once Messaging.....	58
218	4.7.5 Mapping Class of Delivery to Service/Action Pairs.....	59
219	4.8 Sessions in OTA.....	59
220	4.8.1 What we mean by 'sessions'.....	59
221	4.8.2 What we do not mean by 'sessions'.....	59

222	4.8.3	The OTA Session service.....	60
223	4.8.3.1	Session/CreateRQ and Session/CreateRS.....	60
224	4.8.3.2	Session/CloseRQ and Session/CloseRS.....	63
225	4.8.3.3	SessionControl Schema definition.....	64
226	4.8.4	Securing OTA Sessions.....	65
227	4.8.4.1	Basic-authorization and SSL.....	65
228	4.8.4.2	Towards deeper security.....	66
229	4.9	Web Services Description for OTA ebXML.....	66
230	5	OTA Update Messages.....	69
231	5.1	Representing change in XML.....	69
232	5.2	Position Representation with XPath.....	69
233	5.3	Operands.....	70
234	5.4	Operations.....	70
235	5.5	Order of Representation and Application.....	73
236	5.6	Update Examples.....	74
237	5.7	Validation of Update Messages.....	80
238	5.8	The Simple "Replace" verb.....	80
239	5.9	OTA_UpdateRS – Responding to a generic OTA_UpdateRQ message.....	81
240	6	Service and Action Mappings.....	82
241	6.1	The Session Service.....	82
242	6.2	The Profile Service.....	82
243	6.3	The VehicleBooking Service.....	83
244	6.4	The AirBooking Service.....	83
245	6.5	The TravelInsurance Service.....	83
246	6.6	The HotelBooking Service.....	84
247	6.7	The HotelResNotification Service.....	84
248	6.8	The HotelPropertyInformation Service.....	84
249	6.9	The MeetingProfile Service.....	85
250	6.10	The PackageBooking Service.....	85
251	6.11	The GolfTeeTimes Service.....	85
252	6.12	Determining Services Supported.....	86
253	6.12.1	The <ServicesSupported> element.....	86
254	6.12.2	The <Service> element.....	87
255	6.12.3	The <Action> element.....	87
256	6.12.4	The <Extension> element.....	87
257	6.12.5	The ServicesSupported Schema.....	87
258	6.13	Sample Session Message Flow.....	89
259	7	Summary of Infrastructure Changes.....	91
260			
261			

262 **1 About this Document**

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

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

268 **1.1 Intended Audience**

269 This document serves two distinct audiences:

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

276 **1.2 Definitions and Conventions**

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

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

283 **1.3 Relationship with previous OTA standards**

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

- 290 • XML best practices
- 291 • Generic messages and the Service/Action model
- 292 • OTA Infrastructure

293 **1.4 Relationship with ebXML standards**

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

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

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

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

308 **1.5 Use of Namespaces in this specification**

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

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

312

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

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

315

316 **2 OTA XML Best Practices**

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

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

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

333 **2.1 XML Standard Specifications**

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

- 337
- 338 • Extensible Markup Language (XML) 1.0 (Second Edition) :
339 • <http://www.w3.org/TR/2000/REC-xml-20001006>
 - 340 • XML Schema Parts 0 - 2:
341 • <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/>
342 • <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/>
343 • <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>

344 **2.2 Best Practices**

345 **2.2.1 Scope**

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

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

355 **2.2.2 XML Component Parts and Roles**

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

358 I. Tag Naming conventions

359 II. Elements vs. Attributes

360 III. DTD vs. XML Schema

361 IV. Global vs Local Element Types and Elements/Attributes

362 V. Namespaces

363 VI. Versioning XML Schemas

364 VII. XML Markup - General

365 VIII. OTA General

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

370 **2.3 OTA XML Guidelines**

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

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

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

376 **Example:** An example (if applicable).

377 **2.3.1 Tag Naming Conventions (I)**

378 **2.3.1.1 XML Tag Names (I-1)**

379 **Guideline:** Use mixed case tag names, with the leading character of each word in upper case and
380 the remainder in lower case without the use of hyphens between words (a.k.a. “UCC camel case”
381 or “PascalCasing”).

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

383 **Example:** <WorkAddress> <PostalCode>

384 **2.3.1.2 XML Tag Names (I-2)**

385 **Guideline:** Acronym abbreviations are discouraged, but where needed, use all upper case.

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

389 **Example:** <BusinessURL> <HomeUSA>

390 **2.3.1.3 XML Tag Names (I-3)**

391 **Guideline:** Word abbreviations are discouraged, however where needed, word abbreviations
392 should use UCC camel case.

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

396 **Example:** <ProductInfo> <BldgPermit>

397 **2.3.1.4 XML Tag Names (I-4)**

398 **Guideline:** Element and attribute names should not exceed 25 characters. Tag names should be
399 spelled out except where they exceed 25 characters, then standardized abbreviations should be
400 applied.

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

403 **Example:** The tag: <ShareSynchronizationIndicator> can be reduce to:
404 <ShareSyncInd>

405 **2.3.1.5 XML Tag Names (I-5)**

406 **Guideline:** Where the merger of tag name words and acronyms cause two upper case characters
407 to be adjacent, separate them with an underscore (‘_’).

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

409 **Example:** <PO_Box>, <UDDI_Keys>

410 **2.3.1.6 XML Tag Names (I-6)**

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

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

415 **Example:** <ContactAddress> <HomeAddress> <WorkAddress>

416 2.3.1.7 XML Tag Names (I-7)

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

<i>Simple datatype</i>	<i>Description</i>
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 [Thh: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.

421

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

424 **Example:**

```

425 <xs:complexType name="TravelEventDateTime">
426   <xs:simpleContent>
427     <xs:extension base="xs:dateTime">
428       <xs:attribute name="type">
429         <xs:simpleType>
430           <xs:restriction base="xs:string">
431             <xs:enumeration value="depart"/>
432             <xs:enumeration value="arrive"/>
433             <xs:enumeration value="pick-up"/>
434             <xs:enumeration value="drop-off"/>
435             <xs:enumeration value="checkin"/>
436             <xs:enumeration value="checkout"/>

```

```

437     </xs:restriction>
438     </xs:simpleType>
439     </xs:attribute>
440     </xs:extension>
441     </xs:simpleContent>
442 </xs:complexType>

```

443 2.3.2 Elements vs. Attributes (II)

444 2.3.2.1 Elements vs. Attributes (II-1)

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

- 447 • it is not atomic (i.e. It has attributes or child elements of its own) OR
- 448 • the anticipated length of the attribute value is greater than 64 characters¹ OR
- 449 • presence or absence of the attribute represents a semantic 'choice' or branch within the
 450 schema OR
- 451 • an element should also be used where it is likely that the data element in question will be
 452 extended in the future.

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

455 **Example:**

456 Element:

```

457 <LocationDescription>Five miles South of highway 85 and Main St. intersection next to Town
458 Square Mall</LocationDescription>
459

```

460 Attribute:

```

461 <ArrivalAirport LocationCode="MIA" />

```

462 2.3.2.2 Elements vs. Attributes (II-2)

463 **Guideline:** Do not overload element tags with too many attributes (no more than 10 as a rule of
 464 thumb) by encapsulating attributes within child elements that are more closely related (or more
 465 granular). This should be done for those attributes that are likely to be extended by OTA or by
 466 specific trading partners.

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

470 2.3.2.3 Elements vs. Attributes (II-3)

471 **Guideline:** Multiple XML element containers should be used for repeating elements. A single
 472 XML <element> container should **NOT** be used for "simpleType" repeating content (via the
 473 XML Schema "list" construct).

474 **Rationale:** Provides consistency for OTA approved repeating data fields.

¹ URLs are considered less than 64 characters

475 **Example:**

```
476 <States>NY FL CA</States>
477 or
478 <Location RegionStates = "NY" "FL" "GA"/>
```

479 2.3.3 DTD vs. XML Schema (III)

480 2.3.3.1 DTD vs. XML Schema (III-1)

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

483 **Rationale:**

- 484 • Schemas are written in XML syntax, rather than complex SGML regular expression
485 syntax.
- 486 • Because XML Schemas are themselves well-formed XML documents, they can be
487 programmatically generated and validated using a meta-schema -- a schema used to
488 define other schema models.
- 489 • XML schemas have built-in datatypes and an extensible data-typing mechanism. (DTDs
490 only understand markup and character data.)
- 491 • Using an XML syntax to define data model requirements allows for more constraints,
492 strong datatyping, etc.
- 493 • Provides for a consistent Data Repository syntax.

494 2.3.4 Global vs Local Element Types and Elements/Attributes (IV)

495 2.3.4.1 Global vs Local Element Types and Elements/Attributes (IV-1)

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

499 **Rationale:** This approach supports a domain library or repository of reusable XML Schema
500 components. Also, since Schema type names are not contained in XML instance documents, they
501 can be verbose to avoid Schema element type naming collisions.

502 2.3.4.2 Global vs Local Element Types and Elements/Attributes (IV-2)

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

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

511 **Example:**

```
512 <xs:complexType name="AddressType">
513 <xs:sequence>
514 <xs:element name="StreetNmbr" type=" xs:string" minOccurs="0"/>
```

```

515 <xs:element name="BldgRoom" type="PlaceID_Type"
516         minOccurs="0" maxOccurs="unbounded"/>
517 <xs:element name="AddressLine" type="AddressLineType"
518         minOccurs="0" maxOccurs="unbounded"/>
519 <xs:element name="CityName" minOccurs="0">
520   <xs:complexType>
521     <xs:simpleContent>
522       <xs:extension base="xs:string">
523         <xs:attribute name="PostalCode" type="PostalCodeType"/>
524       </xs:extension>
525     </xs:simpleContent>
526   </xs:complexType>
527 </xs:element>
528 <xs:element name="StateProv" type="StateProvinceType" minOccurs="0"/>
529 <xs:element name="CountryName" type="CountryNameType" minOccurs="0"/>
530 <xs:element name="PrivacyDetails" type="Privacy"/>
531 </xs:sequence>
532 </xs:complexType>

```

533 2.3.4.3 Global vs Local Element Types and Elements/Attributes (IV-3)

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

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

540 **Example:**³

```

541 <xs:attributeGroup name="OTA_PayloadStdAttributes">
542   <xs:attribute name="EchoToken" type="OTA_TokenType"/>
543   <xs:attribute name="TimeStamp" type="xs:dateTime"/>
544   <xs:attribute name="Target" default="Production">
545     <xs:simpleType>
546       <xs:restriction base="xs:NMTOKEN">
547         <xs:enumeration value="Test"/>
548         <xs:enumeration value="Production"/>
549       </xs:restriction>
550     </xs:simpleType>
551   </xs:attribute>
552   <xs:attribute name="Version" type="OTA_VersionType"
553     use="required"/>
554   <xs:attribute name="SequenceNmbr" type="xs:integer"/>
555 </xs:attributeGroup>

```

556 2.3.5 Namespaces (V)

557 2.3.5.1 Namespaces (V-1)

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

561 **Rationale:** This approach supports a consistent way to manage and identify OTA’s XML based,
562 transaction assets both internally and externally (via trading partners and global e-business
563 repositories such as UDDI). It also avoids the need for explicit prefixes on both schema and
564 instance docs.

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

565 2.3.5.2 Namespaces (V-2)

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

570 **Rationale:** The same rationale as V-1 above. Also, provides a standard way for "OTA"
 571 namespaced content to be merged with other Industry or Trading Partner namespace content.

572 2.3.5.3 Namespaces (V-3)

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

575 **Rationale:** The same rationale as V-1 above.

576 **Example:**

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

582 2.3.6 Versioning XML Schemas (VI)

583 2.3.6.1 Versioning XML Schemas (VI-1)

584 **Guideline:** The root tag for all XML payload instances should contain a 'version' attribute
 585 (obtained from the 2001C attributeGroup 'PayloadStdAttributes') whose value will mimic the
 586 OTA version release, plus OTA restricted extension meta-data (if extensions are present).⁴
 587 Additionally, the 'schemaLocation' attribute should contain a URI that corresponds to the
 588 location of the schema version (requester's, receiver's or common repository) defining this
 589 particular XML payload instance.

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

593 **Example:**⁵

```
594 '2002A'; '2002B.23'; '2003A.Tabc123'; '2001C.Txyz456'
595 where:
596   '2002A' - is the bi-annual year of release (ie. 2002B, 2003A).
597   '.23' - A numeric reference value '.nn' to the base OTA schema
598           version (ie. 2002A) which contains extensions derived
599           from an XML Schema simpleType or from an existing
600           OTA Schema type (simple or complex).
601
602
```

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

⁵ related to guideline example in section 'VI - 2'.

603
604
605
606
607
608
609

```

'.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.

```

610 2.3.6.2 Versioning XML Schemas (VI-2)

611 **Guideline:** Each version of a schema produced under the 'OTA' namespace must have a unique
612 URI value for the 'version' attribute of the <xs:schema> opening tag. The URI must value
613 must correspond to both the OTA payload message root tag name and to the root tag's version
614 attribute value as shown in the format following example.⁷ Additionally, if ebXML is used as the
615 transport medium, the URI value MUST be duplicated within the ebXML Manifest as the value
616 of its grandchild <eb:schema> element.

617 **Rationale:** Using a version mechanism that parallels the schema-discovery mechanism of
618 validating XML parsers is desirable and is supported by many schema validation tools.
619 Additionally, having a versioning scheme that mimics OTA's specification release methodology
620 reduces the overall work effort of both schema publication and maintainability. Similarly, the
621 'domain_path' recommendations can greatly reduce the work required to maintain and XML
622 Schemas and schema fragments. This feature can be further enhanced by supplying the
623 'domain_path' as schema meta-data in a repository tool. Also, the 2001C Service and Action
624 transaction model can leverage the 'domain_path' as the Service token.

625 **Example:**

626
627
628
629
630
631
632
633
634
635
636
637
638
639
640

```

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

```

641 2.3.7 XML Markup – General (VII)

642 2.3.7.1 XML Markup - General (VII-1)

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

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

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

647 **Example:**

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

651 2.3.7.2 XML Markup - General (VII-2)

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

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

657 **Example:**

```
658 <xs:annotation>
659   <xs:documentation>Privacy sharing control attributes.
660   </xs:documentation>
661 </xs:annotation>
```

662 2.3.7.3 XML Markup - General (VII-3)

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

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

669 2.3.8 OTA General (VIII)

670 2.3.8.1 OTA General (VIII-1)

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

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

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

679 **Example:**⁹

```
680 <OTA_VehAvailRateRQ
681   xmlns="http://www.opentravel.org/OTA"
682   Version="2002A"
683   xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
684   xsi:schemaLocation="http://www.opentravel.org/OTA
685 http://www.opentravel.org/OTA/2002A-REC/VEH-availability/VehAvailRateRQ.xsd">
686   <!-- Payload content... -->
687 </OTA_VehAvailRateRQ>
```

⁹ attribute details are shown in guidelines 'IV-3' and 'VI-2'

688 2.3.8.2 OTA General (VIII-2)

689 **Guideline:** Proprietary trading partner data can be included in an XML instance message within
 690 the <TPA_Extension> global element at OTA sanctioned plug-in points defined in the
 691 schema. This element may also contain the boolean attribute ‘mustProcess’ which notifies that
 692 the message receiver must process the ‘TPA_Extension’ data.

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

695 **Example:** schema fragment:

```
696 <xs:element name="TPA_Extension">
697   <xs:complexType>
698     <xs:complexContent>
699       <xs:extension base="xs:anyType">
700         <xs:attribute name="mustProcess" type="xs:boolean" default="0"/>
701       </xs:extension>
702     </xs:complexContent>
703   </xs:complexType>
704 </xs:element>
705
```

706 **Sample XML:**

```
707 <OTA_VehResRQ xmlns="http://www.opentravel.org/OTA"
708   Version="2002A.Tze123"
709   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
710   xsi:schemaLocation="http://www.opentravel.org/OTA
711 http://www.opentravel.org/OTA/2002A-REC/VEH-booking/VehResRQ-Tze123.xsd">
712   <POS>
713     <Source PseudoCityCode="ABC123" AgentSine="123456789"/>
714     <UniqueId URL="http://switch.com/OTAEngine/"
715       Type="VehResRQ" Id="123456"/>
716     <BookingChannel Type="GDS"/>
717   </Source>
718   <TPA_Extension mustProcess="1">
719     <NegotiatedService Type="TourGuideDriver"/>
720   </TPA_Extension>
721 </POS>
722 <VehRequest>
723   <!--OTA VehRequest content -->
724 </VehRequest>
725 </OTA_VehResRQ>
```

726 2.3.8.3 OTA General (VIII-3)

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

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

731 2.3.8.4 OTA General (VIII-4)

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

734 **Rationale:** Maximizes reuse and avoids definition duplication.

735 2.3.8.5 OTA General (VIII-5)

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

738 **Rationale:** This approach allows OTA defined messages to inter-operate globally more
 739 seamlessly and allows any particular trading partner to locally restrict content values as needed
 740 for unique business requirements.

741 **2.4 Response Message Design**

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

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

749 **2.4.1 Standard Payload Attributes¹⁰**

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

- 752 • *EchoToken*: a sequence number for additional message identification assigned by the
 753 requesting host system. When a request message includes an *EchoToken*, the corresponding
 754 response message MUST include an *EchoToken* with an identical value.
- 755 • *TimeStamp*: indicates the creation date and time of the message in UTC using the following
 756 format specified by ISO 8601: YYYY-MM-DDThh:mm:ssZ with time values using the 24-
 757 hour (military) clock. e.g. 20 November 2000, 1:59:38pm UTC becomes 2000-11-
 758 20T13:59:38Z
- 759 • *Target*: indicates if the message is a test or production message, with a default value of
 760 Production. Valid values: (Test | Production)
- 761 • *Version*: For all OTA versioned messages, the version of the message is indicated by an
 762 integer value.
- 763 • *SequenceNمبر* - This optional attribute is used to identify the sequence number of the
 764 transaction as assigned by the sending system. Allows for an application to process messages
 765 in a certain order, or to request a resynchronization of messages in the event that a system has
 766 been offline and needs to retrieve messages that were missed.

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

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

773 Therefore, every <xxxRS> element MUST have an optional <Success/> element as its first
 774 child. The presence of the empty <Success/> element explicitly indicates that the OTA
 775 message succeeded. In addition to <Success/>, an implementation may return <Warnings>

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.

776 in the event of one or more non-fatal business context errors, OR <Errors> in the event of a
777 failure to process the message altogether.

778 2.4.1.1 Mapping OTA Payload Attributes onto ebXML MS Headers

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

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

783 2.4.2 Design Patterns for Response Messages

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

```
788 <?xml version="1.0" encoding="UTF-8"?>
789 <!-- Created and edited with 'vi' -->
790 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
791           xmlns="http://www.opentravel.org/OTA"
792           targetNamespace="http://www.opentravel.org/OTA"
793           elementFormDefault="qualified">
794
795   <xs:annotation>
796     <xs:documentation xml:lang="en">
797       OTA v2001C Specification - General elements used in response messages
798       Copyright (C) 2001 Open Travel Alliance. All rights reserved.
799     </xs:documentation>
800   </xs:annotation>
801
802   <xs:include schemaLocation="UniqueId.xsd"/>
803
804   <xs:attributeGroup name="OTA_PayloadStdAttributes">
805     <xs:annotation>
```

```

806         <xs:documentation>The OTA_PayloadStdAttributes defines the standard attributes
807         that appear on the root element for all OTA payloads where ebXML is not being used (with the
808         exception of version).</xs:documentation>
809     </xs:annotation>
810     <xs:attribute name="EchoToken" type="xs:string" />
811     <xs:attribute name="TimeStamp" type="xs:string" />
812     <xs:attribute name="Target" default="Production" >
813         <xs:simpleType>
814             <xs:restriction base="xs:NMTOKEN">
815                 <xs:enumeration value="Test" />
816                 <xs:enumeration value="Production" />
817             </xs:restriction>
818         </xs:simpleType>
819     </xs:attribute>
820     <xs:attribute name="Version" type="xs:string" />
821     <xs:attribute name="SequenceNmbr" type="xs:integer" />
822 </xs:attributeGroup>
823
824 <xs:attributeGroup name="ReqRespVersion">
825     <xs:annotation>
826         <xs:documentation>The ReqRespVersion attribute is used to request the version of
827         the payload message desired for the response.</xs:documentation>
828     </xs:annotation>
829     <xs:attribute name="ReqRespVersion" type="xs:string" />
830 </xs:attributeGroup>
831
832 <xs:element name="Success">
833     <xs:complexType>
834         <xs:annotation>
835             <xs:documentation>Standard way to indicate success processing an OTA
836             message</xs:documentation>
837         </xs:annotation>
838     </xs:complexType>
839 </xs:element>
840
841 <xs:element name="Warnings">
842     <xs:complexType>
843         <xs:annotation>
844             <xs:documentation>Indicates successful processing, but warnings
845             occurred.</xs:documentation>
846         </xs:annotation>
847         <xs:sequence>
848             <xs:element ref="Warning" maxOccurs="unbounded" />
849         </xs:sequence>
850     </xs:complexType>
851 </xs:element>
852
853 <xs:element name="Errors">
854     <xs:complexType>
855         <xs:annotation>
856             <xs:documentation>A Warning and an Error element have the same
857             structure.</xs:documentation>
858         </xs:annotation>
859         <xs:sequence>
860             <xs:element ref="Error" maxOccurs="unbounded" />
861         </xs:sequence>
862     </xs:complexType>
863 </xs:element>
864
865 <xs:complexType name="ErrorType">
866     <xs:simpleContent>
867         <xs:restriction base="xs:string">
868             <xs:attribute name="Type" use="required">
869                 <xs:simpleType>
870                     <xs:restriction base="xs:NMTOKEN">
871                         <xs:enumeration value="Unknown" />
872                         <xs:enumeration value="NoImplementation" />
873                         <xs:enumeration value="BizRule" />
874                         <xs:enumeration value="Authentication" />
875                         <xs:enumeration value="AuthenticationTimeout" />
876                         <xs:enumeration value="Authorization" />

```

```

877         <xs:enumeration value="ProtocolViolation"/>
878         <xs:enumeration value="TransactionModel"/>
879         <xs:enumeration value="AuthenticationModel"/>
880         <xs:enumeration value="ReqFieldMissing"/>
881         <xs:enumeration value="TransportFailure"/>
882         <xs:enumeration value="EnvelopeFailure"/>
883     </xs:restriction>
884 </xs:simpleType>
885 </xs:attribute>
886 <xs:attribute name="Code" type="xs:string"/>
887 <xs:attribute name="DocURL" type="xs:string"/>
888 <xs:attribute name="Status" type="xs:string"/>
889 <xs:attribute name="Tag" type="xs:string"/>
890 <xs:attribute name="RecordId" type="xs:string"/>
891 </xs:restriction>
892 </xs:simpleContent>
893 </xs:complexType>
894
895 <xs:element name="Warning" type="ErrorType" />
896
897 <xs:element name="Error" type="ErrorType" />
898 </xs:schema>
899

```

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

```

903 <?xml version="1.0" encoding="UTF-8"?>
904 <!-- Created and edited with 'vi' -->
905 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
906           xmlns="http://www.opentravel.org/OTA"
907           targetNamespace="http://www.opentravel.org/OTA"
908           elementFormDefault="qualified">
909   <xs:include schemaLocation="OTA_v2ent.xsd"/>
910   <xs:include schemaLocation="ExampleElement.xsd"/>
911
912   <xs:element name="ExampleRS">
913     <xs:complexType>
914       <xs:choice>
915         <xs:sequence>
916           <xs:element ref="Success" />
917           <xs:element ref="Warnings" minOccurs="0" />
918           <xs:element ref="Example" />
919         </xs:sequence>
920         <xs:element ref="Errors" />
921       </xs:choice>
922       <xs:attributeGroup ref="OTA_PayloadStdAttributes" />
923     </xs:complexType>
924   </xs:element>
925 </xs:schema>
926

```

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

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

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

- 933 • *Unknown* - Indicates an unknown error. It is recommended that additional information be
 934 provided within the PCDATA, whenever possible.
- 935 • *NoImplementation* - Indicates that the target business system has no implementation for
 936 the intended request. Additional information may be provided within the PCDATA.

- 937 • *BizRule* - Indicates that the XML message has passed a low-level validation check, but
938 that the business rules for the request, such as creating a record with a non-unique
939 identifier, were not met. It is up to each implementation to determine when or if to use
940 this error type or a more specific upper level content error. Additional information may
941 be provided within the PCDATA.
- 942 • *AuthenticationModel* - Indicates the type of authentication requested is not recognized.
943 Additional information may be provided within the PCDATA.
- 944 • *Authentication* - Indicates the message lacks adequate security credentials. Additional
945 information may be provided within the PCDATA.
- 946 • *AuthenticationTimeout* - Indicates that the security credentials in the message have
947 expired. Additional information may be provided within the PCDATA.
- 948 • *Authorization* - Indicates the sender lacks adequate security authorization to perform the
949 request. Additional information may be provided within the PCDATA.
- 950 • *ProtocolViolation* - Indicates that a request was sent within a message exchange that does
951 not align to the message protocols. Additional information may be provided within the
952 PCDATA.
- 953 • *TransactionModel* - Indicates that the target business system does not support the
954 intended transaction-oriented operation. Additional information may be found within the
955 PCDATA.
- 956 • *ReqFieldMissing* - Indicates that an element or attribute that is required by the Schema
957 (or required by agreement between trading partners) is missing from the message.
- 958 • *VersionViolation* – an invalid/unsupported version of a payload was sent

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

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

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

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

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

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

974
975
976
977
978
979
980

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

981 **3 Generic Messages and the Service/Action Model**

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

989 **3.1 The Service/Action Concept**

990 The Service/Action model is conceptually similar to a Web Services model where an
 991 implementation provides one or more high-level services, each of which has one or more
 992 applicable actions (these equate to methods available on services). When used in conjunction with
 993 Request/Response message pairs, the model is conceptually similar to an RPC¹¹, though in fact
 994 the underlying messaging substrate need not rely on RPCs.

995 EbXML provides for the definition of *Service* and *Action* as placeholders for information
 996 pertaining to the intended processing of an ebXML message. From the perspective of the ebXML
 997 message service these values are merely passed through and they are provided to allow parties to
 998 target their messages for processing or action on particular services within their application
 999 systems. *Service* and *Action* take the form of REQUIRED elements within the ebXML message
 1000 header.

1001 When operating over an ebXML substrate, OTA REQUIRES the use of defined values for
 1002 *Service* and *Action* elements and the values defined are analogous in concept to Web Services and
 1003 methods available on a particular web service (think of the ebXML Service being conceptually a
 1004 Web Service and the ebXML Action being analogous to a method within that web service).

1005 **3.1.1 Service/Action Message Mappings**

1006 Within the ebXML Service element there is an optional *type* attribute which parties may use in
 1007 the interpretation of the meaning of service. Within OTA messages flowing over ebXML, the
 1008 value of the type attribute SHALL be "OTA".

1009 The following fragment illustrates the context in which *Service* and *Action* will appear within a
 1010 *MessageHeader*:

1011 **Example 1 - MessageHeader showing a read operation¹² on the AirBooking service**

```
1012 <eb:MessageHeader id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1">
1013   <eb:From>
1014     <eb:PartyId eb:type="urn:duns">008925431</eb:PartyId>
1015   </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

```

1016 <eb:To>
1017 <eb:PartyId eb:type="urn:duns">008296571</eb:PartyId>
1018 </eb:To>
1019 <eb:CPAId>http://opentravel.org/cpa/2001C/defaultcpp.xml</eb:CPAId>
1020 <eb:ConversationId>987654321-123456789</eb:ConversationId>
1021 <eb:Service eb:type="OTA">AirBooking</eb:Service>
1022 <eb:Action>OTA_ReadRQ</eb:Action>
1023 <eb:MessageData>
1024 <eb:MessageId>mid:UUID-2</eb:MessageId>
1025 <eb:Timestamp>2001-10-25T12:19:05Z</eb:Timestamp>
1026 </eb:MessageData>
1027 <eb:QualityOfServiceInfo eb:deliverySemantics="BestEffort" />
1028 </eb:MessageHeader>
1029

```

1030 As such, the following Services are defined within OTA¹³:

<i>OTA Service</i>	<i>Description</i>
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 3 rd 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

1031 **3.2 Unique Identifiers within OTA Messages**

1032 Each record that identifies a unique business document containing travel-related information,
 1033 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.

1034 that creates it with the tag name `<UniqueId>`¹⁴. The unique identifier on the record MUST
 1035 contain both a *Type* and an *Id* attribute. It MAY optionally include a *URL* and an *Instance*
 1036 attribute. The syntax for a `<UniqueId>` is formally defined in the following Schema fragment:

1037 **Schema 1 - <UniqueId>:**

```
1038 <xs:element name="UniqueId">
1039   <xs:complexType>
1040     <xs:attribute name="URL" type="xs:string"/>
1041     <xs:attribute name="Type" type="xs:string" use="required"/>
1042     <xs:attribute name="Id" type="xs:ID" use="required"/>
1043     <xs:attribute name="Instance" type="xs:string"/>
1044   </xs:complexType>
1045 </xs:element>
```

- 1047 • *URL* - This optional attribute is what makes a `<UniqueId>` instance globally unique outside
 1048 the context of a single bilateral conversation between known trading partners. OTA
 1049 RECOMMENDS that the URL be a reference to the public OTA implementation for each
 1050 trading partner. *Note*: In the absence of having a public URL, the reference for this attribute
 1051 could be determined by bilateral agreement.
- 1052 • *Type* - This enumerated attribute references the type of object this `<UniqueId>` refers to,
 1053 and gives this element its generality. By convention, the *Type* attribute value is the same as
 1054 the OTA element tag name for the referenced object, for example, "Profile" or "Reservation".
 1055 As additional message types are defined in future versions of OTA specifications, the *Type*
 1056 attribute enumeration will expand to include additional tag names values where a
 1057 `<UniqueId>` applies.
- 1058 • *Id* - This represents a unique identifying value assigned by the creating system, using the
 1059 XML data type ID. The *Id* attribute might, for example, reference a primary-key value within
 1060 a database behind the creating system's implementation.
- 1061 • *Instance* - This optional attribute represents the record as it exists at a point in time. An
 1062 *Instance* is used in update messages where the sender must assure the server that the update
 1063 sent refers to the most recent modification level of the object being updated. Every time the
 1064 record changes *Instance* assumes a different value.

1065 Possible implementation strategies for *Instance* values are:

- 1066 ▪ a timestamp
- 1067 ▪ a monotonically increasing sequence (incremented on each update)
- 1068 ▪ an md5 sum of the binary representation of the object in its persistent store

1069 **3.2.1 Examples of unique identifiers**

1070 A valid unique identifier MAY contain only the *Type* and *Id* (a unique string assigned by the
 1071 system that created it) attributes:

```
1072 <UniqueId Type="Profile" Id="1234567"/>
```

1073 To ensure that a unique identifier is globally unique (in the universal namespace) add a *URL*
 1074 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.

```
1075 <UniqueId URL="http://vmguys.com/OTAengine/" Type="Profile" Id="1234567"/>
1076
```

1077 This *Id* is assured of being globally unique in any namespace as the URL points to a vendor's
 1078 OTA implementation which, in turn, relies on the unique domain name for the vendor assigned
 1079 by a government approved Internet Domain Name registrar. OTA has considered the potential
 1080 effect a name change would have on globally unique identifiers, and noted that a change in URL
 1081 or primary domain name should not present an issue unless another business entity assumes the
 1082 previous URL unaltered.

1083 This approach to unique naming takes into consideration the following benefits:

- 1084 • provides a simple and succinct representation
- 1085 • guaranteed to be a globally unique identifier within the universal namespace (with the use of
 1086 the URL attribute)
- 1087 • becomes applicable and reusable in other OTA specifications

1088 **3.3 Generic Infrastructure Messages**

1089 Defining certain actions at the infrastructure level allows for reuse on multiple services and
 1090 avoids duplication of work between domain-specific working groups. Generic infrastructure
 1091 messages also offer opportunities for software reuse within implementations. The basic
 1092 operations include Create, Read, Update, and Delete – memorably abbreviated CRUD. These four
 1093 verbs provide consistent conventions for basic actions affecting both infrastructure and business
 1094 elements in OTA specifications.

1095 The Create, Read, and Delete actions **MUST** apply only to entire records. Updates allow for
 1096 addressing one or more individual elements, and making changes to part(s) of a record.

1097 **3.3.1 Create messages**

1098 Create messages define an operation that generates a new record with a unique identifier. The
 1099 sequence follows these steps:

- 1100 • Requestor sends a Create request along with the initial data, and optionally a unique
 1101 identifier.
- 1102 • Receiver creates a new record and assigns a unique identifier (e.g. a Profile Id or Reservation
 1103 Id).
- 1104 • Receiver responds with a message providing a unique identifier for the new record created
 1105 and optionally, any data entered by the requestor.

1106 **Example 2 – Create request message:¹⁵**

```
1107 <OTA_CreateProfileRQ Version="2">
1108   <UniqueId Type="Profile"/>
1109   <Profile>
1110     <Customer Gender="Male">
1111       <PersonName>
1112         <GivenName>John</GivenName>
1113         <Surname>Smith</Surname>
1114       </PersonName>
1115       . . . . .
1116     </Customer>
```

¹⁵ We thank Adam Athimuthu for creating the sample “Create” messages.

```
1117     </Profile>
1118 </OTA_CreateProfileRQ>
```

1119 **Example 3 - Create response message:**

```
1120 <OTA_CreateProfileRS Version="2">
1121   <Success />
1122   <UniqueId
1123     URL="http://www.sixcontinents.com/OTAProcessor"
1124     Type="Profile"
1125     Id="1234567"/>
1126 </OTA_CreateProfileRS>
1127
```

1128 There is no generic OTA message for create. Each working group should define new *Create*
1129 messages for their business objects using the patterns outlined above.

1130 **3.3.2 Generic Read message**

1131 The Read infrastructure action defines an operation that opens an existing record and transmits
1132 information contained in that record. The Read operation enables the user to identify a particular
1133 record and retrieve its entire contents. The basic operation has the following steps:

- 1134 • Requestor queries the database where the record resides by sending a Read request message
1135 with the object's unique identifier
- 1136 • Receiver returns the record to the requestor

1137 The use of the OTA <UniqueId> element allows for a generalized read transaction message.
1138 With the object type specified via the *Type* attribute, the action type is identified within a general
1139 read request.

1140 The use of the OPTIONAL OTA <POS> element allows an implementation to determine whether
1141 the remote user has permission to view the object being read.

1142 **Example 4 - OTA ReadRQ message:**

```
1143 <OTA_ReadRQ ReqRespVersion="2">
1144   <UniqueId
1145     URL="http://vmguys.com/OTAEngine/"
1146     Type="Profile"
1147     Id="0507-12345"/>
1148 </OTA_ReadRQ>
1149
```

1150 *ReqRespVersion* - The optional "Request Response Version" attribute allows the sender to
1151 indicate the version desired for the response message. For example, the OTA_ReadRQ message
1152 sample above indicates a request to return an OTA Customer Profile.

1153 This request applies to all types of objects, not just profiles. The type of the generated response
1154 depends, of course, on the *Type* specified in the request; for example, the <OTA_ReadRQ>
1155 shown in the example would generate a <OTA_ReadProfileRS> message that contains an
1156 OTA Profile as a response, as in the example below.¹⁶

1157 This generalization significantly reduces the maintenance burden for individual infrastructure
1158 verbs, with effectively zero loss in semantics.

1159 **Example 5 - Read response message:**

```
1160 <OTA_ReadProfileRS Version="2">
1161   <UniqueId
```

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

```

1162         URL="http://vmguys.com/OTAEngine/"
1163         Type="Profile"
1164         Id="0507-12345"
1165         Instance="20000531172200"/>
1166     <Profile>
1167         ...
1168         ...
1169     </Profile>
1170 </OTA_ReadProfileRS>
1171

```

1172 An *Instance* value returned in a Read response, if not implemented as a timestamp, may specify
 1173 the same instance value to all requestors until the record is changed by a subsequent action to the
 1174 record, such as an update.

1175 The generic *OTA_ReadRQ* message is formally defined by the following schema fragment:

1176 Schema 2 - <OTA_ReadRQ>:

```

1177 <?xml version="1.0" encoding="UTF-8"?>
1178 <!-- Created and edited with 'vi' -->
1179 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
1180           xmlns="http://www.opentravel.org/OTA"
1181           targetNamespace="http://www.opentravel.org/OTA"
1182           elementFormDefault="qualified">
1183
1184     <xs:annotation>
1185       <xs:documentation xml:lang="en">
1186         OTA v2001C Specification - Generic OTA_ReadRQ message definition
1187         Copyright (C) 2001 Open Travel Alliance. All rights reserved.
1188       </xs:documentation>
1189     </xs:annotation>
1190     <xs:include schemaLocation="OTA_v2ent.xsd"/>
1191     <xs:include schemaLocation="OTA_POS.xsd"/>
1192
1193     <xs:element name="OTA_ReadRQ">
1194       <xs:annotation>
1195         <xs:documentation xml:lang="en">
1196           A generic message, available as an action on several OTA services
1197           which requests a server to read and return the document type
1198           identified by the UniqueId element.
1199         </xs:documentation>
1200       </xs:annotation>
1201       <xs:complexType>
1202         <xs:sequence>
1203           <xs:element ref="UniqueId"/>
1204           <xs:element ref="POS" minOccurs="0"/>
1205         </xs:sequence>
1206         <xs:attributeGroup ref="OTA_PayloadStdAttributes"/>
1207         <xs:attributeGroup ref="ReqRespVersion"/>
1208       </xs:complexType>
1209     </xs:element>
1210 </xs:schema>

```

1211 3.3.3 Generic Update message

1212 The Update infrastructure action defines an operation that opens an existing record, identifies the
 1213 information that needs changing, then transmits data corresponding to the appropriate elements in
 1214 the tree, and adds or replaces those data in the record.

1215 Because Update operations are more complex and can affect parts of the record rather than the
 1216 entire record, handling update messages generally can be more difficult. As a result, two
 1217 approaches to updating records are defined in this specification.

1218 The goals considered in the design of the Update operation include:

- 1219 • Minimizing the size of a payload on the wire to represent an update transaction

- 1220 • Defining an explicit representation about what has changed
- 1221 • Defining a representation with a clear and simple conceptual model
- 1222 • Creating a representation that is content-independent and general-purpose in nature so as
- 1223 to be reusable throughout future OTA specifications
- 1224 • Providing a simple-to-implement "replace" option to allow developers to get simpler
- 1225 implementations running quickly - at the expense of the first 2 goals (representation of
- 1226 change and size of message) above

1227 Because data to be modified may be stored in a database and not in an XML document format, it
 1228 may not be possible to reconstruct the original document that transmitted the data. Therefore, it is
 1229 RECOMMENDED that implementations utilizing the partial update process perform a Read
 1230 request to obtain the structure of the XML tree prior to constructing an Update request.

1231 The definition of the Update message is lengthy. An example update request for a customer
 1232 profile modification is shown below. A detailed treatment of update requests, how to generate and
 1233 approaches to processing them is included in an appendix (see section 5 – OTA Update
 1234 Messages).

1235 Example 6 - OTA_UpdateRQ¹⁷:

```

1236 <?xml version="1.0" encoding="UTF-8"?>
1237 <!-- created by 'DiffGen' using VMTTools 0.2 (http://www.vmguy.com/vmtools/) -->
1238 <OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
1239     xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
1240     xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
1241     ReqRespVersion="2">
1242   <UniqueId Type="Profile"
1243     Id="987654321"
1244     URL="http://www.vmguy.com/OTAEngine/"
1245     Instance="20011012160659" />
1246   <Position XPath="/Profile/Customer/AddressInfo/Address/StreetNmbr">
1247     <Attribute Name="PO_Box" Operation="delete" />
1248   </Position>
1249   <Position XPath="/Profile/Customer/TelephoneInfo">
1250     <Attribute Name="PhoneUse" Operation="modify" Value="Home" />
1251   </Position>
1252   <Position XPath="/Profile/Customer/PersonName/MiddleName">
1253     <Subtree Operation="delete" />
1254   </Position>
1255   <Position XPath="/Profile/Customer">
1256     <Subtree Operation="insert" Child="5">
1257       <RelatedTraveler Relation="Child">
1258         <PersonName>
1259           <NamePrefixNamePrefix>Ms.</NamePrefixNamePrefix>
1260           <GivenName>Amy</GivenName>
1261           <MiddleName>E.</MiddleName>
1262           <Surname>Smith</Surname>
1263         </PersonName>
1264       </RelatedTraveler>
1265     </Subtree>
1266     <Attribute Name="Gender" Operation="modify" Value="Male" />
1267   </Position>
1268 </OTA_UpdateRQ>

```

1269 3.3.4 Generic Delete message

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

1270 The Delete infrastructure action defines an operation that identifies an existing record, and
 1271 removes the entire record from the database. The use of the Delete action depends upon the
 1272 business rules of an organization. Alternative strategies, such as mapping a duplicate record to
 1273 another by use of the UniqueId, may be considered.

1274 The requestor MAY also verify the record before deleting it to ensure the correct record has been
 1275 identified prior to deleting it. In this case, the use of the *Instance* attribute may be useful in
 1276 determining whether the record has been updated more recently than the information that is
 1277 intended to be deleted. That choice, again, would be dictated by good business practices.

1278 Steps in the Delete operation include:

- 1279 • Requestor submits a Read request to view the record
- 1280 • Receiver returns the record for the requestor to view
- 1281 • Requestor submits a Delete request.
- 1282 • Receiver removes the record and returns an acknowledgement

1283 The use of the OPTIONAL OTA <POS> element allows an implementation to determine whether
 1284 the remote user has permission to delete the object being read.

1285 **Example 7 - illustrating the Delete process, Read request:**

```
1286 <?xml version="1.0" encoding="UTF-8"?>
1287 <OTA_ReadRQ ReqRespVersion="2">
1288   <UniqueId>
1289     URL=http://vmguys.com/OTAEngine/
1290     Type="Profile"
1291     Id="0507-12345"
1292   </UniqueId>
1293 </OTA_ReadRQ>
```

1294 **Example 8 - illustrating the Delete process Read response:**

```
1295 <?xml version="1.0" encoding="UTF-8"?>
1296 <OTA_ReadProfileRS Version="2">
1297   <UniqueId>
1298     URL="http://vmguys.com/OTAEngine/"
1299     Type="Profile"
1300     Id="0507-12345"
1301     Instance="2"
1302   </UniqueId>
1303   <Profile>
1304     ...
1305   </Profile>
1306 </OTA_ReadProfileRS>
```

1307 **Example 9 - illustrating the Delete request:**

```
1308 <?xml version="1.0" encoding="UTF-8"?>
1309 <OTA_DeleteRQ ReqRespVersion="2">
1310   <UniqueId>
1311     URL="http://vmguys.com/OTAEngine/"
1312     Type="Profile"
1313     Id="0507-12345"
1314     Instance="2"
1315   </UniqueId>
1316 </OTA_DeleteRQ>
```

1317 **Example 10 - illustrating the Delete response:**

```
1318 <?xml version="1.0" encoding="UTF-8"?>
1319 <OTA_DeleteRS Version="2">
1320   <UniqueId>
1321     URL="http://vmguys.org/OTAEngine/"
1322     Type="Profile"
1323     Id="0507-12345"
```

```

1324     Instance="0"/>
1325   </UniqueId>
1326   <Success/>
1327 </OTA_DeleteRS>
1328

```

1329 A generic OTA delete message is formally defined by the following schema fragment:

1330 Schema 3 - <OTA_DeleteRQ>:

```

1331 <?xml version="1.0" encoding="UTF-8"?>
1332 <!-- Created and edited with 'vi' -->
1333 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
1334           xmlns="http://www.opentravel.org/OTA"
1335           targetNamespace="http://www.opentravel.org/OTA"
1336           elementFormDefault="qualified">
1337
1338   <xs:annotation>
1339     <xs:documentation xml:lang="en">
1340 OTA v2001C Specification - Generic OTA_DeleteRQ message definition
1341 Copyright (C) 2001 Open Travel Alliance. All rights reserved.
1342     </xs:documentation>
1343   </xs:annotation>
1344   <xs:include schemaLocation="OTA_v2ent.xsd"/>
1345   <xs:include schemaLocation="OTA_POS.xsd"/>
1346
1347   <xs:element name="OTA_DeleteRQ">
1348     <xs:annotation>
1349       <xs:documentation xml:lang="en">
1350 A generic message, available as an action on several OTA services
1351 which requests a server to delete the business object
1352 identified by the UniqueId element.
1353       </xs:documentation>
1354     </xs:annotation>
1355     <xs:complexType>
1356       <xs:sequence>
1357         <xs:element ref="UniqueId" />
1358         <xs:element ref="POS" minOccurs="0" />
1359       </xs:sequence>
1360       <xs:attributeGroup ref="OTA_PayloadStdAttributes"/>
1361       <xs:attributeGroup ref="ReqRespVersion"/>
1362     </xs:complexType>
1363   </xs:element>
1364 </xs:schema>
1365

```

1366 A response to a delete request follows the normal pattern for OTA responses and is formally
 1367 defined by the following schema fragment:

1368 Schema 4 - <OTA_DeleteRS>:

```

1369 <?xml version="1.0" encoding="UTF-8"?>
1370 <!-- Created and edited with 'vi' -->
1371 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
1372           xmlns="http://www.opentravel.org/OTA"
1373           targetNamespace="http://www.opentravel.org/OTA"
1374           elementFormDefault="qualified">
1375
1376   <xs:annotation>
1377     <xs:documentation xml:lang="en">
1378 OTA v2001C Specification - Generic OTA_DeleteRS message definition
1379 Copyright (C) 2001 Open Travel Alliance. All rights reserved.
1380     </xs:documentation>
1381   </xs:annotation>
1382   <xs:include schemaLocation="OTA_v2ent.xsd"/>
1383
1384   <xs:element name="OTA_DeleteRS">
1385     <xs:annotation>
1386       <xs:documentation xml:lang="en">
1387 Response to a generic OTA_DeleteRQ message, available as an action

```

```

1388 on several OTA services which requests a server to delete the business
1389 object identified by the UniqueId element.
1390     </xs:documentation>
1391     </xs:annotation>
1392     <xs:complexType>
1393       <xs:choice>
1394         <xs:sequence>
1395           <xs:element ref="Success" />
1396           <xs:element ref="Warnings" minOccurs="0" />
1397           <xs:element ref="UniqueId" />
1398         </xs:sequence>
1399         <xs:element ref="Errors" />
1400       </xs:choice>
1401       <xs:attributeGroup ref="OTA_PayloadStdAttributes"/>
1402     </xs:complexType>
1403   </xs:element>
1404 </xs:schema>

```

1405 3.3.5 Generic Cancel Request

1406 OTA has developed a generic cancel message for use within the travel industry because there are
 1407 common patterns between industry verticals for canceling a reservation, and it is anticipated that
 1408 the messages defined here would be used to cancel a reservation made using the specifications in
 1409 this publication.

1410 The basic pattern for all industry verticals is to identify the reservation by some form of unique
 1411 identification number, whether that number is called a Record Locator Number, PNR,
 1412 Reservation ID, Confirmation ID, or called by some another name. While the nomenclature may
 1413 differ from system to system, the convention is universal: 1) identify the ID unique to the system
 1414 assigned to perform the cancellation, and 2) receive a confirmation of the cancellation as a
 1415 response.

1416 If a travel service is not going to be fulfilled, the availability of inventory is affected, and the
 1417 system holding the inventory will want to release that inventory. Typically, the cancel action is
 1418 executed by the receiving system, as it is usually the system that holds the reserved inventory.
 1419 The cancellation allows the inventory to be returned back to the marketplace and enables the
 1420 supplier to resell it as quickly as possible.

1421 A cancellation could conceptually be considered as a special case of a "modify" transaction
 1422 because the record of the reservation is generally not removed immediately from a system, but
 1423 placed into a different status. However, a cancel transaction differs from a modification because
 1424 the entirety of the services booked will not be consumed. The information exchange differs
 1425 because the system doing the cancellation does not have to return the bulk of the reservation
 1426 information as is the case when confirming the modification of reservation information. The
 1427 process becomes much simpler: send the unique identifier to the system with the indication that
 1428 the action to be taken is to perform a cancel; receive a response that the action has taken place.

1429 A cancellation also differs from the generic infrastructure verb, OTA_DeleteRQ. The Delete
 1430 action removes the record out of the database, while in a cancellation, the record of the
 1431 reservation is not necessarily deleted. A cancellation is a business process driven by the business
 1432 rules applied to the reservation, and effectively provides a change of status of that reservation.

1433 3.3.5.1 Consequences of Canceling

1434 When a cancel message is sent, two possibilities exist: the reservation may be canceled without
 1435 penalty, or the cancellation incurs a penalty for doing so. For many travel services, perhaps the
 1436 majority in this day and age of bargains, promotional fares and special rates, some sort of
 1437 restrictions may apply. A cancellation of travel services may result in forfeiture of an amount paid
 1438 for a guarantee, or deposit, etc.

1439 3.3.5.2 Read Request Prior to Cancel

1440 The logical steps that occur for a cancellation recognize that prior to sending a cancellation
 1441 message, the party holding the reservation may wish to retrieve and review that reservation.
 1442 Reasons for doing so include reviewing associated information that may communicate the
 1443 cancellation policy, or simply to confirm the identification of the reservation to be cancelled,
 1444 among a series of reservations held by that party.

1445 The function of retrieving the reservation uses the generic OTA_ReadRQ, specifying the
 1446 UniqueId of the reservation, with Type="Reservation", and optionally supplying a URL to
 1447 identify the party holding the reservation or the machine location where the reservation is being
 1448 stored.

1449 Example 11 - <OTA_ReadRQ> message:

```
1450 <OTA_ReadRQ ReqRespVersion="1">
1451   <UniqueId
1452     URL="http://provider1.com/OTAEngine/"
1453     Type="Reservation"
1454     Id="05071G4325"/>
1455 </OTA_ReadRQ>
1456
```

1457 The response is a specific OTA Read message, containing the reservation information.

1458 Example 12 - <OTA_ReadHotelResRS> message:

```
1459 <OTA_ReadHotelResRS Version="1">
1460   <UniqueId
1461     URL="http://vmguys.com/OTAEngine/"
1462     Type="Reservation"
1463     Id="05071G4325"
1464     Instance="20000531172200"/>
1465   <HotelRes>
1466     ...
1467     ...
1468   </HotelRes>
1469 </OTA_ReadHotelResRS>
1470
```

1471 An optional *Instance* value returned in a Read response, if implemented as a timestamp, may
 1472 indicate the most recent record of the reservation and allow for synchronization if the requesting
 1473 and receiving party do not hold the same reservation.

1474 3.3.5.3 Security considerations

1475 Inherent in the business practices of companies exchanging information is the requirement for
 1476 some kind of qualification that entitles the requestor to receive the reservation information.
 1477 Sufficient information should be sent for the receiving system to be able to recognize the
 1478 requesting system, and to qualify it to receive the information.

1479 Systems may assume a point-to-point connection upon receiving a request to retrieve a booking,
 1480 but may still need to know who is on the terminal, particularly with transactions that may come
 1481 from travel web sites where the individual has the opportunity to log on and control their own
 1482 reservation. The booking engine, or application processing the request, is tasked to qualify the
 1483 requestor so that it can be certain that the party is who they represent themselves to be. This
 1484 usually involves furnishing some kind of information such as a last name, membership number,
 1485 confirmation number, or credit card number (at least the last 4 digits of the credit card number)
 1486 that was used for the reservation. That level of verification may be required even with the
 1487 identification of the source and booking channel supplied by the Point-of-Sale information.

1488 This generic cancel message assumes that the software asking for the read or cancel action has
1489 pre-determined the other party's rights for viewing at either end of the message conversation. The
1490 same considerations for the security of the connection, as well as identification of the requesting
1491 party, is true for transactions involving reading and modifying a profile.

1492 It is common practice for many systems to keep a summary level view of a reservation, retaining
1493 information about the requestor. Therefore, within the remit of information in the request, some
1494 level of security can be applied to view or cancel that information. In many cases, sending the
1495 transaction to retrieve a reservation must be made through the channel or source where the
1496 original booking took place.

1497 Complications arise when, within a booking or channel, all of the other qualifiers might be right,
1498 but the request is not made through the same source. For example, a travel agency branch office
1499 could be handling a request for a customer who made the original reservation through another
1500 location. In that case, the retrieving application is then tasked to take the qualification to the next
1501 level and match the request up to the channel and source to display only those reservations that
1502 were booked through that agency.

1503 Travel suppliers may wish to support returning all reservations that can be identified with their
1504 company, e.g. using identifiable confirmation numbers, regardless of where the reservation was
1505 booked (e.g. through their Central Reservation System, or through a web site, etc.). Conversely,
1506 the customer who had booked a reservation through a specific web site, could not go on another
1507 travel supplier web site, and retrieve a booking made on the competitor's site. The reservation
1508 could only be retrieved through the original source.

1509 *Note:* It is likely that within one company, systems will be able to use the generic Read or Cancel
1510 request for conversations between trusted sources, such as an interface from their web site to a
1511 legacy system. Additional information may be needed to establish a level of confidence between
1512 trading partners. As the nature of XML is to be extensible, partners wishing to expand upon
1513 requirements in their environment may use the <TPA_Extension> defined elsewhere in this
1514 document.¹⁸

1515 The use of the OPTIONAL OTA <POS> element allows an implementation to determine whether
1516 the remote user has permission to cancel a particular booking.

1517 **3.3.5.4 OTA Cancel Messages**

1518 This specification provides a request/response pair of messages to support the functionality of
1519 canceling a reservation.

1520 The following message pair is used on all applicable services to cancel a reservation:

- 1521 • **OTA_CancelRQ** - Identifies the reservation and requests a cancellation.
- 1522 • **OTA_CancelRS** – MAY return a list of rules that govern the cancellation, a cancellation
1523 number upon execution of the cancel action, or Warnings or Errors if the processing of the
1524 request did not succeed.

1525 **3.3.5.4.1 OTA Cancel Request**

1526 The root element of the OTA_CancelRQ contains the standard payload attributes found in all
1527 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>

1528 of the response message. As this is the first publication of the OTA cancel message set, currently
1529 the only valid value is "1".

1530 The cancel request also has an attribute, `CancelType = " "`, that defines the action
1531 requested in the cancel message.

1532 Attributes of `OTA_CancelRQ` are as follows:

- 1533 • ***OTA_PayloadStdAttributes*** - includes the 5 standard attributes on all OTA messages.
- 1534 • ***ReqRespVersion*** - Requests a version of the response message.
- 1535 • ***CancelType*** - An enumerated type indicating the type of request made for the cancellation.
1536 Valid Values are: (Initiate | Ignore | Confirm).
- 1537 ***Initiate*** - Indicates the initial request to cancel a reservation.
- 1538 ***Ignore*** - Indicates a roll-back of the request to cancel, leaving the reservation intact.
- 1539 ***Confirm*** - Indicates a request to complete the cancellation.

1540 **3.3.5.4.2 Cancel Request**

1541 The cancel request is formally defined by the following schema fragment:

1542 **Schema 5 - <OTA_CancelRQ> message:**

```

1543 <?xml version="1.0" encoding="UTF-8"?>
1544 <!-- Created and edited with 'vi' -->
1545 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
1546           xmlns="http://www.opentravel.org/OTA"
1547           targetNamespace="http://www.opentravel.org/OTA"
1548           elementFormDefault="qualified">
1549
1550   <xs:annotation>
1551     <xs:documentation xml:lang="en">
1552       OTA v2001C Specification - Generic OTA_CancelRQ message definition
1553       Copyright (C) 2001 Open Travel Alliance. All rights reserved.
1554     </xs:documentation>
1555   </xs:annotation>
1556   <xs:include schemaLocation="OTA_v2ent.xsd"/>
1557   <xs:include schemaLocation="OTA_POS.xsd"/>
1558
1559   <xs:element name="OTA_CancelRQ">
1560     <xs:annotation>
1561       <xs:documentation xml:lang="en">
1562         A generic message, available as an action on several OTA services which
1563         requests a server to cancel the booking identified by the UniqueId element.
1564       </xs:documentation>
1565     </xs:annotation>
1566
1567     <xs:complexType>
1568       <xs:sequence>
1569         <xs:element ref="UniqueId"/>
1570         <xs:element ref="POS" minOccurs="0"/>
1571       </xs:sequence>
1572       <xs:attributeGroup ref="OTA_PayloadStdAttributes"/>
1573       <xs:attributeGroup ref="ReqRespVersion"/>
1574       <xs:attribute name="CancelType" use="required">
1575         <xs:simpleType>
1576           <xs:restriction base="xs:string">
1577             <xs:enumeration value="Initiate"/>
1578             <xs:enumeration value="Ignore"/>
1579             <xs:enumeration value="Confirm"/>
1580           </xs:restriction>
1581         </xs:simpleType>
1582       </xs:attribute>

```

```

1583     </xs:complexType>
1584   </xs:element>
1585 </xs:schema>

```

1586 **3.3.5.4.3 Cancel Request - Sample XML message**

1587 The following sample message is an example of an initial cancel message:

```

1588 <OTA_CancelRQ xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1589             xsi:noNamespaceSchemaLocation="OTA_CancelRQ.xsd"
1590             Version="1" ReqRespVersion="1"
1591             CancelType="Initiate">
1592   <UniqueId URL=" http://provider1.org/OTAEngine/"
1593           Type="Reservation"
1594           Id="0507G4325"
1595           Instance="2001-06-03T13:09:21" />
1596 </OTA_CancelRQ>

```

1597 **3.3.5.5 Confirmation of Cancellation**

1598 When a cancellation has been executed, the majority of systems will return a cancellation
 1599 number. The identification number supplied serves to confirm that the action has taken place, and
 1600 makes it possible to track monetary ramifications that may be associated with the cancellation.

1601 A single cancel request can result in a transaction that straightforwardly cancels a reservation and
 1602 returns a confirmation that the cancellation has taken place. Alternatively, the request to cancel a
 1603 reservation may violate a business rule, subjecting the party to a cancellation penalty. In that case,
 1604 it is desirable to return information about the cancellation penalty in the response.

1605 The OTA_CancelRQ message allows for the possibility to perform a two-phase transaction to
 1606 inform the party wishing to cancel of the penalties that would be incurred, and to allow them to
 1607 verify their intention to cancel. The CancelType attribute is a flag that can be set to the choices of
 1608 (Initiate | Ignore | Confirm). This enables the message conversation to send in an initial request to
 1609 cancel a reservation, anticipate a response that returns the consequences of doing so, and allows
 1610 for the option of rolling back the transaction and choosing NOT to cancel the reservation.

1611 **3.3.5.5.1 OTA Cancel Response**

1612 The <OTA_CancelRS> message follows the standard design pattern for response messages.¹⁹
 1613 Additionally, it MAY return a collection of cancellation rules, with penalty amounts, if incurred,
 1614 and an indication of the status of the cancellation request, either "Cancelled", "Pending", or
 1615 "Ignored" if the transaction has been rolled back and the reservation remains intact.

1616 Attributes of OTA_CancelRS are as follows:

- 1617 • **OTA_PayloadStdAttributes** - includes the 5 standard attributes on all OTA messages.
- 1618 • **CancelID** - The identification number of the cancellation.
- 1619 • **Status** - An enumerated type indicating the status of the cancellation request. Valid Values
 1620 are: (Pending | Ignored | Canceled).
- 1621 *Pending* - Indicates the initial request to cancel a reservation is pending confirmation to
 1622 complete the cancel action. Cancel rules may have been returned along with the response.
- 1623 *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

1624 *Canceled* - Indicates the cancellation is complete. A cancellation ID may have been returned
1625 along with the response.

1626 By providing the option of a two-step process, individual business rules may determine how their
1627 system processes the initial request. If there are no penalties involved in the cancellation, the
1628 cancel transaction can take place and the response return the cancellation number along with the
1629 status that the reservation has been cancelled.

1630 If the processing system determines that a cancellation policy has been invoked, it may choose to
1631 send back the OTA_CancelRS with the Status="Pending", accompanied by a collection of
1632 cancellation rules, allowing the originating party to determine if the cancellation should proceed.
1633 The originating system would then resend the OTA_CancelRQ. A CancelType="Ignore" would
1634 anticipate a response with the Status "Ignored", thus ending the message conversation with no
1635 action being taken to cancel the reservation.

1636 A CancelType = "Commit" indicates a definitive "Yes" to process the cancellation. This message
1637 would anticipate the response of Status="Cancelled", along with the return of a Cancellation Id,
1638 and that transaction would complete the cancellation process. The cancel RQ is the same message
1639 in each case, with the CancelType attribute indicating the action to be taken on the request.

1640 **3.3.5.5.2 Cancel Rules**

1641 The <OTA_CancelRS> message has two child elements: the reservation record is identified by
1642 the use of the UniqueId element, and the cancellation rules and/ or penalties are communicated by
1643 the use of the CancelRules collection that may return one or many CancelRule elements.

1644 The attributes of the CancelRule element are as follows:

- 1645 • **CancelByDate** - Identifies the date by which a cancellation should be made in order to avoid
1646 incurring a penalty.
- 1647 • **Amount** - The monetary amount incurred as a result of the cancellation.
- 1648 • **CurrencyCode** - The code that identifies the currency of the penalty amount, using ISO 4217.

1649 The data type of the CancelRule element is *xs:string*, which can be used for text describing the
1650 cancellation penalties. Cancel Rule is a repeating element and can be used as many times as
1651 needed to communicate the cancel penalties and rules

1652 **3.3.5.5.3 Cancel Response**

1653 The Cancel response is formally defined by the following schema fragment:

1654 **Schema 6 - <OTA_CancelRS> message:**

```

1655 <?xml version="1.0" encoding="UTF-8"?>
1656 <!-- Created and edited with 'vi' -->
1657 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
1658           xmlns="http://www.opentravel.org/OTA"
1659           targetNamespace="http://www.opentravel.org/OTA"
1660           elementFormDefault="qualified">
1661
1662     <xs:annotation>
1663       <xs:documentation xml:lang="en">
1664         OTA v2001C Specification - Generic OTA_CancelRS message definition
1665         Copyright (C) 2001 Open Travel Alliance. All rights reserved.
1666       </xs:documentation>
1667     </xs:annotation>
1668     <xs:include schemaLocation="OTA_v2ent.xsd"/>
1669
1670     <xs:element name="OTA_CancelRS">
```

```

1671     <xs:complexType>
1672       <xs:choice>
1673         <xs:sequence>
1674           <xs:element ref="Success" />
1675           <xs:element ref="Warnings" minOccurs="0" />
1676           <xs:element ref="UniqueId" />
1677           <xs:element ref="CancelRules"/>
1678         </xs:sequence>
1679         <xs:element ref="Errors" />
1680       </xs:choice>
1681     <xs:attributeGroup ref="OTA_PayloadStdAttributes" />
1682     <xs:attribute name="Status" use="required">
1683       <xs:simpleType>
1684         <xs:restriction base="xs:string">
1685           <xs:enumeration value="Pending"/>
1686           <xs:enumeration value="Ignored"/>
1687           <xs:enumeration value="Canceled"/>
1688         </xs:restriction>
1689       </xs:simpleType>
1690     </xs:attribute>
1691     <xs:attribute name="CancelId" type="xs:string"/>
1692   </xs:complexType>
1693 </xs:element>
1694
1695 <xs:element name="CancelRules">
1696   <xs:complexType>
1697     <xs:sequence>
1698       <xs:element ref="CancelRule" minOccurs="0" maxOccurs="unbounded" />
1699     </xs:sequence>
1700   </xs:complexType>
1701 </xs:element>
1702
1703 <xs:element name="CancelRule">
1704   <xs:complexType>
1705     <xs:simpleContent>
1706       <xs:extension base="xs:string">
1707         <xs:attribute name="CancelByDate" type="xs:string"/>
1708         <xs:attribute name="Amount" type="xs:string"/>
1709         <xs:attribute name="CurrencyCode" type="xs:string"/>
1710       </xs:extension>
1711     </xs:simpleContent>
1712   </xs:complexType>
1713 </xs:element>
1714 </xs:schema>

```

1715 3.3.5.5.4 Cancel Response - Sample XML message

1716 The following sample message is an example of a cancel response message:

```

1717 <OTA_CancelRS xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1718   xmlns="http://www.opentravel.org/OTA"
1719   xsi:schemaLocation="http://www.opentravel.org/OTA OTA_CancelRS.xsd"
1720   Version="2"
1721   Status="Canceled"
1722   CancelId="1236597GQ345">
1723   <Success/>
1724   <UniqueId URL="http://provider1.org/OTAEngine/"
1725     Type="Reservation"
1726     Id="0507G4325"
1727     Instance="2001-06-03T13:09:21" />
1728   <CancelRules>
1729     <CancelRule CancelByDate="2001-05-31T13:19:42-05:00"
1730       Amount="75.00"
1731       CurrencyCode="USD">
1732   Cancellations within 30 days incur a penalty of $75.00</CancelRule>
1733   </CancelRules>
1734 </OTA_CancelRS>

```

1735 **4 OTA Infrastructure**

1736 In this section we explore the revised OTA infrastructure. See section 7 for a list of changes since
1737 the last publication of the infrastructure specification in 2001A.

1738 **4.1 Architecture Overview**

1739 OTA's Infrastructure Architecture borrows heavily from ebXML's Technical Architecture²⁰
1740 [ebTA] and Message Service²¹ [ebMS]. EbXML's Message Service, which is based on SOAP
1741 version 1.1 and the Soap with Attachments informational document, provides the functionality
1742 needed for two or more parties to engage in an "electronic business transaction". Products that
1743 implement the ebMS specification should be capable of the reliable and secure exchange of
1744 business data associated with a "business transaction". It is expected that any company
1745 implementing an OTA compliant system will be capable of supporting synchronous and
1746 asynchronous processing models as required by the various OTA message types described
1747 elsewhere in this document

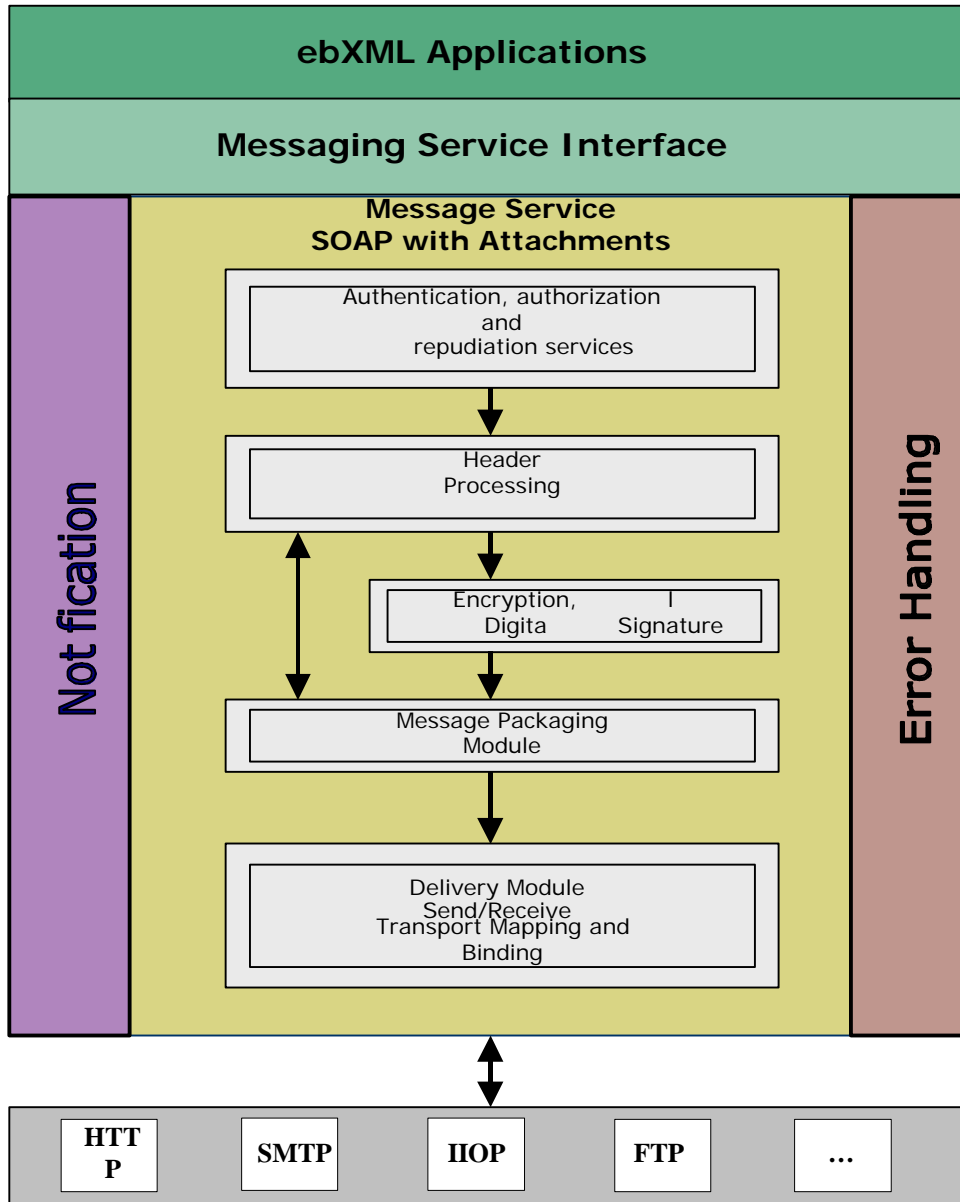
1748 Business Applications interface with an ebMS through a "service interface", which is unique to
1749 each ebMS product. The "service interface" allows applications to request services and receive
1750 notifications from an ebMS product. When an application needs to engage in an electronic
1751 business transaction with a trading partner it must call upon its local ebMS service, through the
1752 service interface, to establish a secure and reliable session with a trading partners ebMS/OTA
1753 system. Once a session is established the ebMS/OTA systems exchange information germane to
1754 the type of "service/action" being requested. A session may include multiple message exchanges
1755 between the ebMS/OTA systems of two or more parties.

1756 . V2001 offers a typical developer working on an OTA-compliant application the option of using
1757 a standard ebMS rather than (as with v2001A) being compelled to design and implement the
1758 OTA infrastructure. It is the ebMS product vendor who is responsible for implementing the
1759 functionality described in the ebMS specification [ebMS]. A high quality ebMS implementation
1760 would also provide facilities for encrypting and digitally signing data, access control,
1761 authentication and authorization, real-time error notifications, logging, auditing and
1762 administrative tasks.

1763 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>

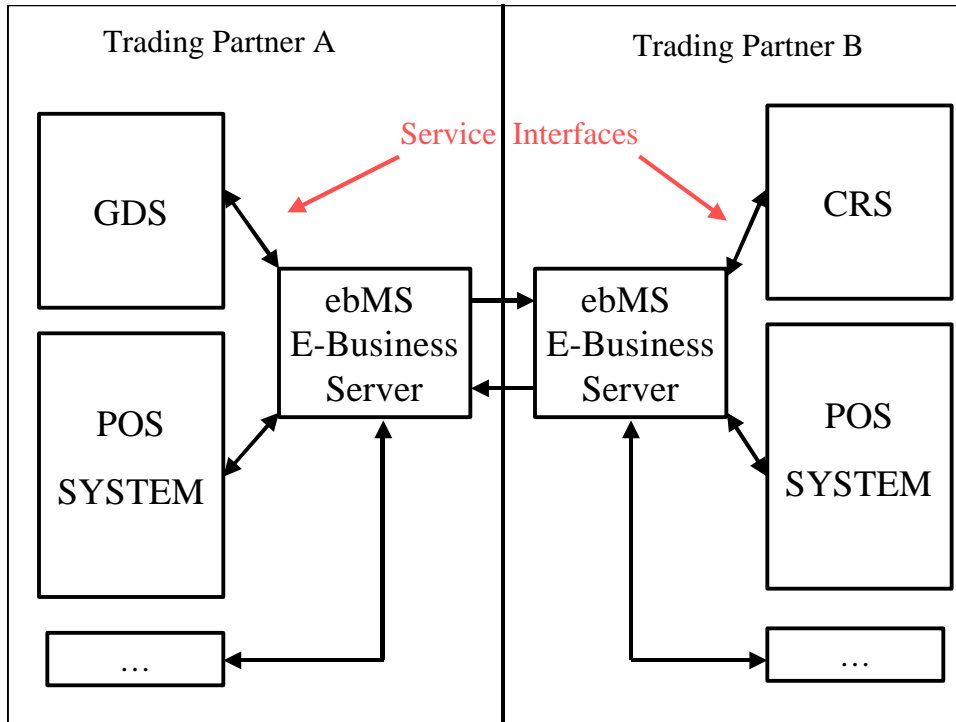


1764

1765 **4.1.1 Reference Model**

1766 An ebMS may serve as a centralized “E-Business Server” to manage all of a company’s
 1767 electronic business transaction exchanges with trading partners. Alternatively ebMS functionality
 1768 may be “integrated“ within each business application that engages in electronic business
 1769 transactions with trading partners. Regardless of which model is used ebMS implementers
 1770 SHOULD provide the same service levels, access control, reliability, availability and security
 1771 required by the electronic business transactions defined by OTA.

1772 Below is a reference model depicting a centralized E-Business Server:

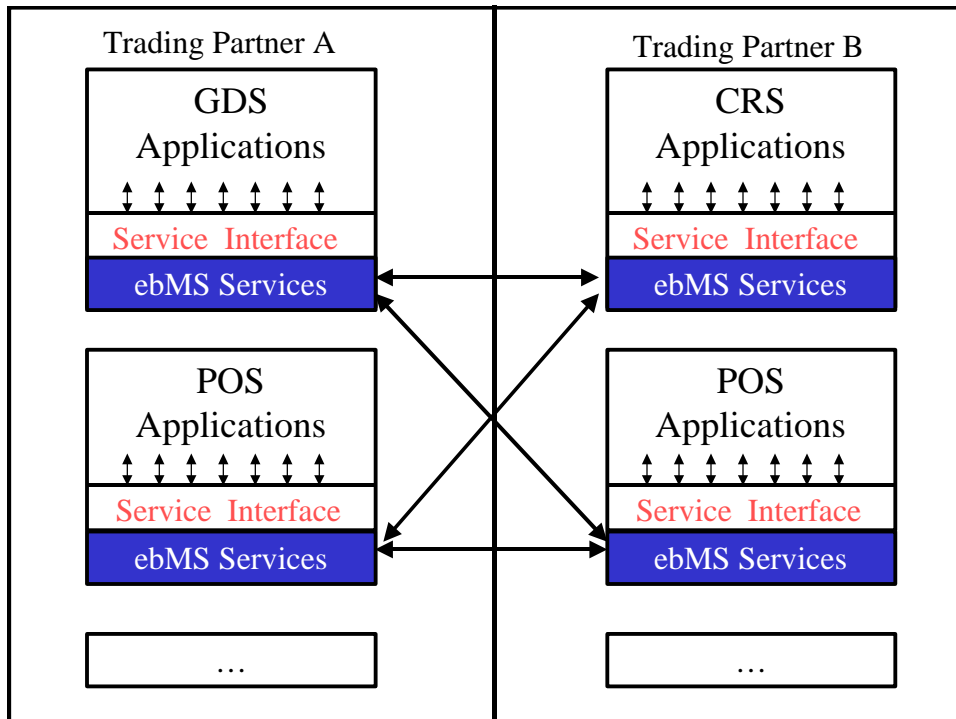


1773

1774

1775

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



1776

1777

4.1.2 Transport Protocols

1778 The ebMS was designed to be transport neutral, however there are certain transport protocols that
1779 are more likely to be used for electronic transactions between trading partners over the Internet,
1780 one such protocol is HTTP (ref: RFC 2616).

1781 EbMS/OTA implementers **MUST** support the HTTP protocol binding specifications defined in
1782 appendix B.2 of the ebMS specification. Additionally, ebMS/OTA implementers **MUST** support
1783 Basic Authentication (ref: RFC 2617) for access control, operating over a secure channel, using
1784 SSL Version 3.0 or TLS (ref: RFC 2246) with 128-bit key sizes for symmetric encryption
1785 algorithms. EbMS implementers **MUST** accept self signed digital certificates, as well as those of
1786 well-known Certificate Authorities (e.g. Verisign, Entrust, Thawte, et al), during the
1787 establishment of an SSL session.

1788 OTA implementers **SHOULD** maintain ebMS systems that are available 24 hours per day, 7 days
1789 per week, to receive and process electronic business transactions from trading partners.

1790 **4.1.3 Logging**

1791 Organizations offering OTA transactions **SHOULD** provide logging capability, regardless of the
1792 type of transaction in the business message (e.g., travel verbs, infrastructure verbs), and trading
1793 partners **MAY** exchange event logs to provide audit trails.

1794 Because of the lack of widely used standards or conventions for defining event logs, OTA does
1795 not require use of a specific log format, nor does the message architecture preclude any logging
1796 capability. Logging capabilities are expected to vary based upon the capabilities of an underlying
1797 ebXML message service implementation.

1798 **4.1.4 Auditing**

1799 EbMS product implementations **SHOULD** maintain audit logs that contain information used to
1800 track and correlate message exchanges between trading partners. The following information
1801 **SHOULD** be stored in an audit log:

- 1802 • Date and time of entry
- 1803 • Sender and Receiver IP addresses
- 1804 • To/PartyId
- 1805 • From/PartyId
- 1806 • Status of the exchange (success/failure)
- 1807 • MessageId
- 1808 • RefToMessageId
- 1809 • CPAId
- 1810 • ConversationId
- 1811 • Service
- 1812 • Action
- 1813 • Via
- 1814 • QualityOfServiceInfo/syncreply value, if present
- 1815 • Contents of Each Reference Element within a manifest
- 1816 • Contents of Acknowledgement element, if present

- 1817 • Contents of Error elements, if present
- 1818 EbMS products SHOULD provide administrative functions to search and view logs.

1819 **4.2 Message Structure and Packaging**

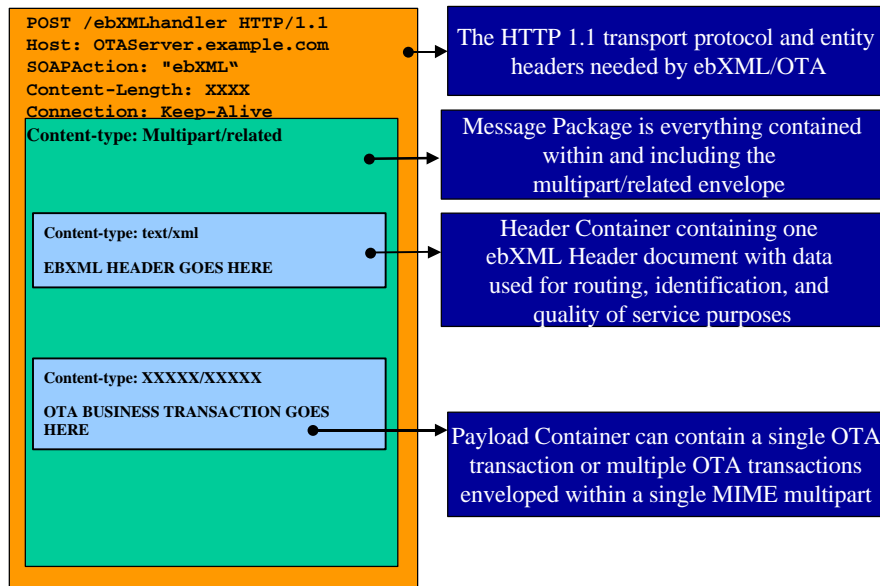
1820 An ebXML Message following OTA’s conventions MUST contain one header container and zero
 1821 or one payload container. Both containers are enveloped by a single MIME/Multipart envelope
 1822 and the entire package is referred to as a *Message Package*.

1823 The two containers within a *Message Package* are described below:

- 1824 • The first MIME part, referred to as the *Header Container*, contains one ebXML
 1825 header document. The header document contains only those elements and attributes
 1826 required by OTA, the details of which are specified elsewhere in this document.
- 1827 • The optional second MIME part, referred to as the *Payload Container*, contains
 1828 application level payloads containing business data germane to the service/action
 1829 identified in the ebXML header. A single payload container may contain one or more
 1830 OTA business transactions.

1831 The general structure and composition of an OTA compliant ebXML Message is described in
 1832 the following figure.

1833



1834

1835 4.2.1 The ‘Unit-of-work’ Concept

1836 EbXML is extremely flexible in how messages can be packaged within payloads and in the
1837 absence of any guidelines implementers may choose substantially different approaches to
1838 packaging. As such OTA strongly recommends that all OTA message payloads within a given
1839 ebXML package pertain to one and only one ‘unit of work’.

1840 The concept of a unit of work is very similar to the concept of transactions in a transaction
1841 processing environment. An easy way to decide whether messages belong to the same unit of
1842 work is by applying the following tests:

- 1843 • Do the messages fall within the boundary of a single transaction?
- 1844 • Do the messages constitute items within the same batch?
- 1845 • A single OTA message by definition matches the unit-of-work criteria

1846 4.2.2 Packaging a single unit-of-work

1847 The “scope” of a unit-of-work is highly dependent on the capabilities of the systems that engage
1848 in an electronic transaction. For example, a reservation system may be capable of performing a
1849 reservation for a single type of transaction (airline reservation), whereas another system may be
1850 capable of performing a reservation for an airline, automobile and hotel in the context of a single
1851 transaction. In the case where a single transaction (e.g. one airline reservation) is the unit-of-work
1852 this data may be packaged as a single MIME body part with a content-type (aka media type)
1853 identifying the data (e.g. application/xml, application/EDI-X12, image/jpeg, etc.) each will have a
1854 corresponding entry in the manifest that references it. Following is an example of a single
1855 transaction packaged within a single payload container:

```
1856 Content-ID: airlinerreservation111@imacompany.com
1857 Content-Type: text/xml
1858
1859 <OTA_AirBookRQ>
1860 <!-- content omitted for brevity... -->
1861 </OTA_AirBookRQ>
```

1862

1863 When a unit-of-work involves multiple transaction documents (e.g. a reservation delivery
1864 notification with a corresponding customer profile) the data may be packaged within a MIME
1865 multipart content-type containing multiple body parts, with each body part containing a single
1866 transaction document. For example the following multipart MIME structure represents a single
1867 unit-of-work that would exist in a single payload container:

```
1868 Content-type: multipart/related; boundary="Boundary"
1869 --Boundary
1870 Content-ID: hotelreservation111@imacompany.com
1871 Content-Type: text/xml
1872
1873 <OTA_HotelResNotifRQ>
1874 <!-- content omitted for brevity... -->
1875 </OTA_HotelResNotifRQ>
1876
1877 --Boundary
1878 Content-ID: hotelreservation111profile@imacompany.com
1879 Content-Type: text/xml
1880
1881 <OTA_CreateProfileRQ>
1882 <!-- content omitted for brevity... -->
1883 </OTA_CreateProfileRQ>
1884
1885 --Boundary--
```

1886

1887 In some cases an application system may operate in a “batch mode” where multiple, related
 1888 transactions may be packaged as a single unit-of-work. For example, many POS systems process
 1889 a batch of transactions during the settlement process. A single “batch” may contain individual
 1890 transactions (e.g. authorizations, credits, voids, etc.) that are processed as a single unit-of-work.
 1891 OTA compliant systems are allowed to package multiple transactions into a “batch” for
 1892 processing as a single unit-of-work. For example a hotel reservation system capable of processing
 1893 a “batch” of reservations may expect to receive the following payload container:

```

1894 Content-type: multipart/related; boundary="Boundary"
1895 --Boundary
1896 Content-ID: hotelreservation111@imacompany.com
1897 Content-Type: text/xml
1898
1899 <OTA_HotelResNotifRQ>
1900 </OTA_HotelResNotifRQ>
1901
1902 --Boundary
1903 Content-ID: hotelreservation112@imacompany.com
1904 Content-Type: text/xml
1905
1906 <OTA_HotelResNotifRQ>
1907 </OTA_HotelResNotifRQ>
1908
1909 --Boundary
1910 Content-ID: hotelreservation113@imacompany.com
1911 Content-Type: text/xml
1912
1913 <OTA_HotelResNotifRQ>
1914 </OTA_HotelResNotifRQ>
1915
1916 --Boundary--

```

1917 **4.3 Classes of Message Delivery**

1918 The EbXML Messaging Service [ebMS] on top of SOAP with attachments which does not
 1919 inherently provide any underlying mechanism for reliable messaging. An ebXML ebMS
 1920 implementation will however provide the ability to use reliable messaging for the transport of
 1921 messages.

1922 Within OTA it is anticipated that there is a need for both reliable and non-reliable messaging,
 1923 depending upon the type of messages being exchanged.

1924 **4.3.1 Historical use within the travel industry**

1925 The travel industry has a long history of automated message exchange dating back to the 1960’s.
 1926 Broadly speaking two classes of message delivery have been used and continue to be used in
 1927 legacy systems:

- 1928 • Type A – interactive, best effort delivery
- 1929 • Type B – store-and-forward, ‘guaranteed’, ordered delivery

1930 **4.3.1.1 Type A²²**

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)

1931 Type A message delivery is typically used for the expedient delivery of interactive messages
 1932 (usually request/response pairs synchronous within a sub-conversation). Upon non-response,
 1933 failure or timeout applications may fall back to alternate messages via type-B.

1934 *NOTE: Type A messages utilize ebXML's Best Effort delivery semantics, which makes no*
 1935 *guarantee as to the delivery of a message, nor does it prevent duplicate messages from being*
 1936 *delivered. It is possible to have multiple deliveries of a single Type A message, which could*
 1937 *result in the receipt and processing of duplicate messages by a receiving Message Service*
 1938 *Handler. It is assumed that a recipient application program, above the Message Service*
 1939 *Handler, will prevent the processing of duplicate Type A messages. This will be relatively easy*
 1940 *to do given timeToLive constraints.*

1941 **4.3.1.2 Type B²³**

1942 Type B message delivery is typically used for 'guaranteed', ordered delivery of messages.
 1943 Delivery of type-B messages is often at a lower priority than type-A messages. Type-B messages
 1944 are used in scenarios where responses may or may not be expected. They are not used for
 1945 synchronous request/response type exchanges.

1946 **4.3.2 EbXML Classes of Delivery**

1947 The ebMS has been designed to meet a spectrum of requirements from one-way, unreliable
 1948 message delivery up-to and including guaranteed, ordered message delivery using synchronous or
 1949 asynchronous exchanges.

1950 **4.4 EbXML Header Document**

1951 The ebMS defines an ebXML header as an XML document, structured according to SOAP
 1952 version 1.1. A SOAP XML document consists of one Envelope Element, which contains one
 1953 mandatory body element and one optional header element. The ebMS mandates that all ebXML
 1954 header documents MUST contain both header and body elements. The following example depicts
 1955 a skeleton SOAP structure:

```
1956 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
1957   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
1958   <SOAP-ENV:Header>
1959   </SOAP-ENV:Header>
1960   <SOAP-ENV:Body>
1961   </SOAP-ENV:Body>
1962 </SOAP-ENV:Envelope>
```

1963 **4.4.1 OTA Subset of an ebXML Header Document**

1964 An ebXML header document contains the information necessary for two communicating
 1965 Message Service Handlers to engage in a conversation involving the exchange and processing of
 1966 an OTA unit-of-work. The ebMS defines several complex header elements that are used for
 1967 identification, routing, Quality of Service, Tracking and Status reporting. OTA does not require
 1968 an implementer to support all of the functionality defined by ebMS. OTA implementers are only
 1969 required to support the functionality needed to support the exchange of type A and B messages.

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

1970 ebXML's Message Service identifies 6 child elements of the SOAP header element and 4 child
 1971 elements of the SOAP body element. For purposes specific to OTA, only the following ebXML
 1972 header elements are allowed to appear as child elements of the SOAP Header:

<i>Element Name</i>	<i>Usage</i>	<i>Message Types</i>
ErrorList	OPTIONAL	A and B, RESPONSE
Acknowledgement	REQUIRED	B, RESPONSE
	OPTIONAL	A, RESPONSE
Via	REQUIRED	A, REQUEST
MessageHeader	REQUIRED	A and B, REQUEST and RESPONSE

1973

1974 **NOTE: the ebXML Message Service 1.0 specification supports two additional elements**
 1975 **within a SOAP-ENV:Header, eb:TraceHeaderList and ds:Signature. These elements are**
 1976 **not required by this version of the OTA specification, but they may be used between**
 1977 **consenting parties as needed.**

1978 4.4.1.1 ErrorList Element

1979 The `ErrorList` element is required when reporting errors pertaining to the contents of an
 1980 ebXML header document (e.g. invalid `ToPartyId`), errors pertaining to payload "faults" (e.g.
 1981 missing required payload), or other types of errors. Further details on the type of ebXML errors
 1982 that can be reported within an `ErrorList` element are defined in section 11 of the ebMS. One
 1983 child element is allowed to appear in an `ErrorList`, the `Error` element. An `ErrorList`
 1984 may contain multiple `Error` elements. The syntax and structure of `ErrorList` and `Error`
 1985 elements are defined in section 8.8 of the ebMS.

1986 OTA maintains a list of error codes that may be used in response to any OTA request message.
 1987 When reporting OTA errors within a response message the `Error` element used to report an OTA
 1988 error condition **MUST** contain a `codeContext` attribute with the value
 1989 "<http://www.opentravel.org/errorCodes>". For example:

```
1990 <eb:ErrorList eb:highestSeverity="Error" eb:version="1.0" SOAP-ENV:mustUnderstand="1">
1991 <eb:Error eb:codeContext="http://www.opentravel.org/errorCodes" eb:errorCode="SessionFailure"
1992 eb:severity="Error">OTA version not supported </eb:Error>
```

1993

1994 The following table contains all the possible OTA errors²⁴:

<i>errorCode</i>	<i>Severity</i>	<i>ErrorMessages</i>
SessionFailure-100	Error	OTA version not supported
SessionFailure-101	Error	Session has expired
SessionFailure-102	Error	Session already closed
SessionFailure-103	Error	Parameter not supported

²⁴ Note these are infrastructure and/or session related errors, not application level errors. Application errors are allowed for in an `<OTA:Errors>` element allowed in the standard design pattern used for all response messages. See section 2.4.2 for details.

1995 **4.4.1.2 Acknowledgement Element**

1996 The Acknowledgement element will only appear as part of an ebXML response message. It is
 1997 used to indicate that a request message has been successfully received by a receiving Server.
 1998 Further details of the Acknowledgement element can be found in section 8.6 of the ebMS.
 1999 The Acknowledgement element used by OTA contains the following attributes and child
 2000 elements:

2001 The Acknowledgement element contains three REQUIRED attributes:

- 2002 • SOAP-ENV:mustUnderstand with a value of “1”
- 2003 • SOAP-ENV:actor with the value “<http://schemas.xmlsoap.org/soap/actor/next>”
- 2004 • eb:version with a value of “1.0”

2005 The Acknowledgement element contains one required element, Timestamp that contains the
 2006 creation date and time of the Acknowledgement formatted in accordance with XML Schema
 2007 timeInstant. Following is an Acknowledgement example:

```
2008 <eb:Acknowledgment SOAP-ENV:mustUnderstand="1" eb:version="1.0"
2009     SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
2010     <eb:Timestamp>2001-03-09T12:22:30Z</eb:Timestamp>
2011 </eb:Acknowledgment>
```

2013 The existence of an Acknowledgement element in a Response indicates that a Request
 2014 message was successfully received. The existence of an ErrorList in a Response indicates
 2015 that a Request has failed.

2016 **4.4.1.3 Via Element**

2017 The Via element is used to indicate the type of response (synchronous/asynchronous) expected
 2018 of a server receiving a request message. Further details of the Via element can be found in
 2019 section 8.7 of the ebMS. It is only used on type A messages, those requiring a synchronous,
 2020 business level response.

2021 The Via element contains four REQUIRED attributes and one optional attribute:

2022 **REQUIRED attributes:**

- 2023 • SOAP-ENV:mustUnderstand with a value of “1”
- 2024 • SOAP-ENV:actor with a value of “<http://schemas.xmlsoap.org/soap/actor/next>”
- 2025 • eb:version with a value of “1.0”
- 2026 • eb:syncReply with a value of “true”

2027 **OPTIONAL attribute:**

- 2028 • eb:ackRequested with a value of “Unsigned”

2029 Type A messages MAY contain an eb:ackRequested=“Unsigned” to indicate that the
 2030 sender requests that the receiving Message Service Handler return an Acknowledgement
 2031 element indicating that a message was successfully received and processed.

2032 Following is an example usage of the Via element:

```
2033 <eb:Via SOAP-ENV:mustUnderstand="1" SOAP-
2034 ENV:actor="http://schemas.xmlsoap.org/soap/actor/next" eb:version="1.0" eb:syncReply="true"
2035 eb:ackRequested="Unsigned"> </eb:Via>
```

2036 4.4.1.4 MessageHeader Element

2037 Each Type A and type B, request and response message MUST contain one MessageHeader
 2038 Element. Each MessageHeader element MUST contain one SOAP-ENV:mustUnderstand
 2039 attribute with a value of “1” (including double quotes) and one eb:version attribute with a value
 2040 of “1.0”. See example below:

```
2041 <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2042 ...
2043 </eb:MessageHeader>
2044
```

2045 OTA utilizes the following child elements of the Message Header element:

<i>Element Name</i>	<i>Usage</i>	<i>Message Types</i>	<i>Purpose</i>
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

2046 4.4.1.4.1 To element

2047 Contains one child element, PartyId, which contains the DUNS Number of the intended recipient.
 2048 The PartyId element contains one attribute, “type”, containing the value “urn:x12.org:I05:01”.
 2049 This value indicates the data contained in the PartyId element is semantically defined by the X12
 2050 organization (x12.org), I05 indicates the particular set of identifiers defined by X12 to identify
 2051 “parties” and 01 indicates the content of PartyId is a DUNS Number. Following is an example of
 2052 the To element:

```
2053 <eb:To>
2054   <eb:PartyId eb:type="urn:x12.org:I05:01">123456789</eb:PartyId>
2055 </eb:To>
```

2056 4.4.1.4.2 From element

2057 Contains one child element, PartyId, which contains the DUNS Number of the sender. The
 2058 PartyId element contains one attribute, “type”, containing the value “urn:x12.org:I05:01”. This
 2059 value indicates the data contained in the PartyId element is semantically defined by the X12
 2060 organization (x12.org), I05 indicates the particular set of identifiers defined by X12 to identify
 2061 “parties” and 01 indicates the content of PartyId is a DUNS Number. Following is an example of
 2062 the From element:

```
2063 <eb:From>
2064   <eb:PartyId eb:type="urn:x12.org:I05:01">987654321</eb:PartyId>
2065 </eb:From>
```

2066 **4.4.1.4.3 CPAId element**

2067 This element is required by ebXML, however its content is determined by mutual consent
 2068 between parties. CPAId may be used to identify a particular agreement (e.g. trading partner
 2069 agreement) that defines the “rules of engagement” two communicating parties agree to abide by.
 2070 If no such agreement exists CPAId MUST contain a constant value of “NULL”, for example:

```
2071 <eb:CPAId>NULL</eb:CPAId>
```

2072 **4.4.1.4.4 ConversationId element**

2073 This element is used to provide context for a particular exchange of messages. It is primarily used
 2074 for session identification. The content of ConversationId is a compound string consisting of
 2075 sid@FQDN. A “sid” is a session identifier. Each party is required to construct unique sid values
 2076 for each new session established. A “Fully Qualified Domain Name” (FQDN) MUST be
 2077 appended to a sid, in order to ensure the uniqueness of ConversationId’s across company
 2078 boundaries.

2079 A ConversationId is obtained by a sender during the establishment of a new session. Refer to the
 2080 section titled “Sessions in OTA” for detailed usage of this element. Following is an example
 2081 ConversationId:

```
2082 <eb:ConversationId>20011017161501-777@B2Bserver.imacompany.com  
2083 </eb:ConversationId>
```

2084 **4.4.1.4.5 Service and Action elements**

2085 The Service and Action elements identify the particular operation being performed, which
 2086 correspond directly to specific protocol behavior. The Service element may contain one of the
 2087 possible values defined in the section titled “Service and Action Mappings”. The service element
 2088 MUST contain a “type” attribute with the fixed value “OTA” when using Services defined by this
 2089 document. Other Services MAY be utilized which are not defined within this document (e.g.
 2090 ebXML Ping message). When using non-OTA defined Services implementers are expected to
 2091 follow the usage guidelines defined for the Service.

2092 The Action element is used to provide additional context to the Service element. The Action
 2093 identifies the protocol “verb”. Taken together the Service/Action should provide sufficient
 2094 identification to invoke proper protocol behavior between parties. The list of possible values for
 2095 the Service element, the type attribute and the Action element are listed in section titled “Service
 2096 and Action Mappings”.

2097 Following is an example:

```
2098 <eb:Service type="OTA">Profile</eb:Service>  
2099 <eb:Action>OTA_CreateProfileRQ</eb:Action>
```

2100 **4.4.1.4.6 MessageData element**

2101 • This element is used to provide identification information for each ebXML message
 2102 exchanged between parties. There are three possible child elements under MessageData:

<i>Element Name</i>	<i>Usage</i>	<i>Message Types</i>	<i>Purpose</i>
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

2103

2104 4.4.1.4.6.1 MessageId Element

2105 Contains a unique message identifier formed in conformance with RFC 2392, which defines a
2106 string containing uniqueidentifier@FQDN, for example:

```
2107 <eb:MessageId>mid:abc123@b2bserver.imacompany.com</eb:MessageId>
```

2108 4.4.1.4.6.2 Timestamp Element

2109 Contains the creation date and time of the message formatted in accordance with XML Schema
2110 timeInstant, for example:

```
2111 <eb:TimeStamp>2001-02-15T11:12:12Z</eb:TimeStamp>
```

2112 4.4.1.4.6.3 RefToMessageId Element

2113 Contains the message identifier from the original MessageId that “this” message is in response to,
2114 for example:

```
2115 <eb:RefToMessageId>mid:abc123@b2bserver.imacompany.com</eb:RefToMessageId>
```

2116 4.4.1.4.6.4 TimeToLive Element

2117 Contains the expiration date and time of a message formatted in accordance with XML Schema
2118 timeInstant, for example:

```
2119 <eb:TimeToLive>2001-02-15T11:12:12Z</eb:TimeToLive>
```

2120

2121 Both the sending and receiving ebMS are expected to generate and report an error condition
2122 whenever message delivery/processing has aborted due to expiration. A Sending ebMS MUST
2123 notify a higher layer application, through its service interface or by some other means, whenever
2124 a message request aborts due to expiration. If a Sending ebMS has been configured to retry
2125 message delivery for messages with BestEffort deliverySemantics (ref: retries in section 10.2.6 of
2126 [ebMS], NOTE: this is not the recommended behavior for BestEffort deliverySemantics) then the
2127 sending ebMS SHOULD attempt to resend a message that aborts due to expiration. The
2128 recommended minimum time for TimeToLive SHOULD be no less than 5 seconds in the future
2129 from the date time value contained in the MessageData/TimeStamp.

2130 A request message containing a TimeToLive value is considered satisfied within its allotted time
2131 period if a corresponding response message is received by the sending ebMS before the point in
2132 time identified by the TimeToLive.

2133 Message Service Handlers are expected to maintain time synchronization by synchronizing
2134 systems clocks, at least monthly, with a recognized time source, such as the National Institute of
2135 Standards and Technology NIST-F1 Cesium Fountain Atomic Clock,
2136 <http://nist.time.gov/timezone.cgi?UTC/s/0>

2137 4.4.1.4.7 QualityOfServiceInfo element

2138 This element is used to indicate the need for reliable message exchange between parties. The
2139 ebMS defines three possible attributes for this element, however OTA will only utilize two,
2140 **deliverySemantics** and **messageOrderSemantics**.

2141 This element is used on type A and B Request Messages. Type A messages MUST specify a
2142 deliverySemantics attribute with the value “**BestEffort**”, (including double quotes). Type B

2143 messages MUST specify a deliverySemantics attribute with the value “**OnceAndOnlyOnce**”
 2144 (including double quotes) and messageOrderSemantics containing the value “Guaranteed”.
 2145 Examples below:

2146 TYPE A:

```
2147 <eb:QualityOfServiceInfo eb:deliverySemantics="BestEffort" />
```

2149 TYPE B:

2150

```
2151 <eb:QualityOfServiceInfo eb:deliverySemantics="OnceAndOnlyOnce"  
2152 messageOrderSemantics="Guaranteed" />
```

2153 **4.4.1.4.8 SequenceNumber**

2154 This element indicates the sequence in which messages MUST be processed by a receiving
 2155 Message Service. Each Type B request message MUST contain a SequenceNumber element in
 2156 order to guarantee ordered delivery. This element contains an integer value in the range 0-
 2157 99999999. There is one REQUIRED attribute, “status”, that MUST contain one of the following
 2158 values, “Reset” or “Continue”. Detailed usage of this element may be found in section 8.4.8 of
 2159 [ebMS].

2160 **4.4.1.4.9 Description**

2161 This element contains a human readable description that SHOULD be germane to the
 2162 Service/Action of the message in which it appears. The element MAY be present on any request
 2163 or response message.

2164 **4.5 SOAP Body Elements**

2165 The following ebXML header elements MUST be located within the SOAP Body element, when
 2166 required:

- 2167 • **Manifest**

2168 The Manifest element is used to identify all of the payload documents within in a payload
 2169 container. Each document within a payload container MUST be identified by a corresponding
 2170 Reference element within the Manifest. The Manifest element and at least one Reference element
 2171 MUST be present in a Message Package containing a payload container.

2172 The Reference element is used to identify information needed by a particular service/action pair
 2173 in order to perform proper processing. Conceptually, the Reference element can be thought of as
 2174 identifying the “arguments” needed by a particular Service/Action. A Reference element contains
 2175 two Xlink attributes that provide information about a payload. The two attributes are:

<i>Attribute Name</i>	<i>Purpose</i>	<i>Possible Values</i>
xlink:href	Contains a URI identifying a payload document	Examples: cid:123@imacompany.com http://www.imacompany.com/file1
xlink:type		“simple”

2176

2177 Following is an example of a Manifest element containing one Reference element, indicating the
2178 presence of one payload document:

```
2179 <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2180 <eb:Reference xlink:href="cid:hotelreservation111@example.com"
2181 <xlink:type="simple">
2182 </eb:Reference>
2183 </eb:Manifest>
```

2184 **4.6 EbXML Collaboration Protocol Profile**

2185 EbXML has defined the Collaboration-Protocol-Profile and Agreement Specification [ebCPP²⁵]
2186 for the definition of key parameters used by trading partners within the context of their on-the-
2187 wire eCommerce conversations.

2188 Whereas portions of a valid CPP document are pertinent only to the parties participating within a
2189 particular conversation, other portions of a CPP may be more generally applicable. In the interests
2190 of fostering wire compatibility between implementations the OTA RECOMMENDS the use of
2191 the following CPP fragment within any CPP documents you may define. This OTA fragment
2192 defines transport and delivery channels consistent with the infrastructure defined in this
2193 document:

```
2194 <tp:Transport tp:transportId="HTTPS01">
2195 <tp:SendingProtocol tp:version="1.1">HTTP</tp:SendingProtocol>
2196 <tp:ReceivingProtocol tp:version="1.1">HTTP</tp:ReceivingProtocol>
2197 <tp:Endpoint tp:uri="http://example.com/servlet/ebxmlhandler"
2198 tp:type="allPurpose"/>
2199 <tp:TransportSecurity>
2200 <tp:Protocol tp:version="3.0">SSL</tp:Protocol>
2201 </tp:TransportSecurity>
2202 </tp:Transport>
2203
2204 <tp:DocExchange tp:docExchangeId="TYPEA">
2205 <tp:ebXMLBinding tp:version="1.0">
2206 <tp:ReliableMessaging tp:deliverySemantics="BestEffort" tp:idempotency="true">
2207 </tp:ReliableMessaging>
2208 </tp:ebXMLBinding>
2209 </tp:DocExchange>
2210
2211 <tp:DocExchange tp:docExchangeId="TYPEB">
2212 <tp:ebXMLBinding tp:version="1.0">
2213 <tp:ReliableMessaging tp:deliverySemantics="OnceAndOnlyOnce"
2214 tp:idempotency="true"
2215 tp:messageOrderSemantics="NotGuaranteed">
2216 <tp:Retries>5</tp:Retries>
2217 <tp:RetryInterval>1800</tp:RetryInterval> <!--time in seconds-->
2218 <tp:PersistDuration>24H</tp:PersistDuration>
2219 </tp:ReliableMessaging>
2220 </tp:ebXMLBinding>
2221 </tp:DocExchange>
2222
2223 <tp:DeliveryChannel tp:channelId="TYPEA" tp:transportId="HTTPS01"
2224 tp:docExchangeId="TYPEA">
2225 <tp:Characteristics tp:syncReplyMode="responseOnly"
2226 tp:nonrepudiationOfOrigin="false"
2227 tp:nonrepudiationOfReceipt="false"
2228 tp:secureTransport="true"
2229 tp:confidentiality="false"
2230 tp:authenticated="true"
2231 tp:authorized="true"/>
2232 </tp:DeliveryChannel>
2233
2234 <tp:DeliveryChannel tp:channelId="TYPEB" tp:transportId="HTTPS01"
2235 tp:docExchangeId="TYPEB">
```

²⁵ see: <http://www.ebxml.org/specs/ebCPP.pdf>

```

2236     <tp:Characteristics tp:syncReplyMode="none"
2237                       tp:nonrepudiationOfOrigin="false"
2238                       tp:nonrepudiationOfReceipt="false"
2239                       tp:secureTransport="true"
2240                       tp:confidentiality="false"
2241                       tp:authenticated="true"
2242                       tp:authorized="true" />
2243 </tp:DeliveryChannel>

```

2244 4.7 EbXML Header Examples

2245 4.7.1 Type A Request Message

```

2246 Content-ID: <ebxhmheader111@B2B.company.com>
2247 Content-Type: text/xml; charset="UTF-8"
2248
2249 <?xml version="1.0" encoding="UTF-8"?>
2250 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
2251                   xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
2252                   xmlns:xlink="http://www.w3.org/1999/xlink">
2253   <SOAP-ENV:Header>
2254     <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2255       <eb:From>
2256         <eb:PartyId type="urn:x12.org:I05:01">123456789</eb:PartyId>
2257       </eb:From>
2258       <eb:To>
2259         <eb:PartyId type="urn:x12.org:I05:01">912345678</eb:PartyId>
2260       </eb:To>
2261       <eb:CPAId>NULL</eb:CPAId>
2262       <eb:ConversationId>2000120913300328572@b2b.company.com
2263         <eb:ConversationId>
2264       <eb:Service eb:type="OTA">Profile</eb:Service>
2265       <eb:Action>OTA_CreateProfileRQ</eb:Action>
2266       <eb:MessageData>
2267         <eb:MessageId>mid:20001209-133003-28572@b2b.company.com</eb:MessageId>
2268         <eb:Timestamp>2001-02-15T11:12:12Z </eb:Timestamp>
2269         <eb:TimeToLive>2001-02-15T11:12:17Z </eb:TimeToLive>
2270       </eb:MessageData>
2271       <eb:QualityOfServiceInfo eb:deliverySemantics="BestEffort" />
2272     </eb:MessageHeader>
2273     <eb:Via SOAP-ENV:mustUnderstand="1"
2274       SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next"
2275       eb:version="1.0" eb:syncReply="true" />
2276   </SOAP-ENV:Header>
2277   <SOAP-ENV:Body>
2278     <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2279       <eb:Reference xlink:href="cid:profile111@b2b.company.com"
2280         xlink:type="simple">
2281       </eb:Reference>
2282     </eb:Manifest>
2283   </SOAP-ENV:Body>
2284 </SOAP-ENV:Envelope>

```

2285 4.7.2 Type B Request Message

```

2286 Content-ID: <ebxhmheader111@B2B.company.com>
2287 Content-Type: text/xml; charset="UTF-8"
2288
2289 <?xml version="1.0" encoding="UTF-8"?>
2290 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
2291                   xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
2292                   xmlns:xlink="http://www.w3.org/1999/xlink">
2293   <SOAP-ENV:Header>
2294     <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2295       <eb:From>
2296         <eb:PartyId type="urn:x12.org:I05:01">123456789</eb:PartyId>
2297       </eb:From>
2298     <eb:To>

```

```

2299     <eb:PartyId type=" urn:x12.org:I05:01">912345678</eb:PartyId>
2300   </eb:To>
2301   <eb:CPAId>NULL</eb:CPAId>
2302   <eb:ConversationId>2000120913300328572@b2b.company.com
2303     </eb:ConversationId>
2304   <eb:Service eb:type="OTA">Profile</eb:Service>
2305   <eb:Action>OTA_CreateProfileRQ</eb:Action>
2306   <eb:MessageData>
2307     <eb:MessageId>mid:20001209-133003-28572@b2b.company.com</eb:MessageId>
2308     <eb:Timestamp>2001-02-15T11:12:12Z </eb:Timestamp>
2309   </eb:MessageData>
2310   <eb:QualityOfServiceInfo eb:deliverySemantics="OnceAndOnlyOnce"
2311     eb:messageOrderSemantics="Guaranteed"/>
2312   <eb:SequenceNumber eb:status="Reset">0</eb:SequenceNumber> </eb:MessageHeader>
2313 </SOAP-ENV:Header>
2314 <SOAP-ENV:Body>
2315   <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2316     <eb:Reference xlink:href="cid:profile111@b2b.company.com"
2317       xlink:type="simple">
2318       </eb:Reference>
2319     <eb:Reference xlink:href="cid:profile112@b2b.company.com"
2320       xlink:type="simple">
2321       </eb:Reference>
2322     <eb:Reference xlink:href="cid:profile113@b2b.company.com"
2323       xlink:type="simple">
2324       </eb:Reference>
2325     <eb:Reference xlink:href="cid:profile114@b2b.company.com"
2326       xlink:type="simple">
2327       </eb:Reference>
2328   </eb:Manifest>
2329 </SOAP-ENV:Body>
2330 </SOAP-ENV:Envelope>

```

2331 4.7.3 Type A Response Example

```

2332 Content-ID: <ebxhmheader111@B2Bserver.imacompany.com>
2333 Content-Type: text/xml; charset="UTF-8"
2334
2335 <?xml version="1.0" encoding="UTF-8"?>
2336 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
2337   xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
2338   xmlns:xlink="http://www.w3.org/1999/xlink">
2339 <SOAP-ENV:Header>
2340   <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2341     <eb:From>
2342       <eb:PartyId type="urn:x12.org:I05:01">912345678</eb:PartyId>
2343     </eb:From>
2344     <eb:To>
2345       <eb:PartyId type=" urn:x12.org:I05:01">123456789</eb:PartyId>
2346     </eb:To>
2347     <eb:CPAId>NULL</eb:CPAId>
2348     <eb:ConversationId>2000120913300328572@b2b.company.com
2349       </eb:ConversationId>
2350     <eb:Service eb:type="OTA">Profile</eb:Service>
2351     <eb:Action>OTA_CreateProfileRS</eb:Action>
2352     <eb:MessageData>
2353       <eb:MessageId>mid:20001209-133503-1111@B2Bserver.imacompany.com</eb:MessageId>
2354       <eb:Timestamp>2001-02-15T11:15:12Z </eb:Timestamp>
2355       <eb:RefToMessageId> mid:20001209-133003-
2356 28572@b2b.company.com</eb:RefToMessageId>
2357     </eb:MessageData>
2358   </eb:MessageHeader>
2359 </SOAP-ENV:Header>
2360 <SOAP-ENV:Body>
2361   <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2362     <eb:Reference xlink:href="cid:profile111@b2b.imacompany.com"
2363       xlink:type="simple">
2364       </eb:Reference>
2365   </eb:Manifest>
2366

```

```
2367 </SOAP-ENV:Body>
2368 </SOAP-ENV:Envelope>
```

2369 4.7.4 Type B Response Example

```
2370 Content-ID: <ebxhmheader111@B2Bserver.imacompany.com>
2371 Content-Type: text/xml; charset="UTF-8"
2372
2373 <?xml version="1.0" encoding="UTF-8"?>
2374 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
2375     xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
2376     xmlns:xlink="http://www.w3.org/1999/xlink">
2377 <SOAP-ENV:Header>
2378   <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2379     <eb:From>
2380       <eb:PartyId type="urn:x12.org:I05:01">912345678</eb:PartyId>
2381     </eb:From>
2382     <eb:To>
2383       <eb:PartyId type="urn:x12.org:I05:01">123456789</eb:PartyId>
2384     </eb:To>
2385     <eb:CPAId>NULL</eb:CPAId>
2386     <eb:ConversationId>2000120913300328572@b2b.company.com
2387     </eb:ConversationId>
2388     <eb:Service eb:type="OTA">Profile</eb:Service>
2389     <eb:Action>OTA_CreateProfileRS</eb:Action>
2390     <eb:MessageData>
2391       <eb:MessageId>mid:20001209-133503-1111@B2Bserver.imacompany.com</eb:MessageId>
2392       <eb:Timestamp>2001-02-15T11:15:12Z </eb:Timestamp>
2393       <eb:RefToMessageId> mid:20001209-133003-
2394 28572@b2b.company.com</eb:RefToMessageId>
2395     </eb:MessageData>
2396   </eb:MessageHeader>
2397 <eb:Acknowledgment SOAP-ENV:mustUnderstand="1" eb:version="1.0"
2398     SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next" >
2399   <eb:Timestamp>2001-03-09T12:22:30Z</eb:Timestamp>
2400 </eb:Acknowledgment>
2401
2402 </SOAP-ENV:Header>
2403 <SOAP-ENV:Body>
2404 </SOAP-ENV:Body>
2405 </SOAP-ENV:Envelope>
```

2406 4.7.4.1 Reliable Messaging

2407 In the context of OTA, reliable messaging refers to the requirements that an ebXML Message
 2408 Service Product be capable of reliably delivering and persisting incoming and outgoing payload
 2409 data until the successful delivery to a trading partner or an internal application. It is expected that
 2410 all ebXML compliant Message Service products are capable of performing reliable messaging in
 2411 accordance with the ebMS specification following the requirements for
 2412 `reliableMessagingMethod="ebXML"` (the only type supported by OTA at this time and
 2413 the default method for ebXML) in section 10.2.4.

2414 A message requires reliable delivery if the `deliverySemantics="OnceAndOnlyOnce"`.
 2415 Only type B request messages require reliable delivery.

2416 4.7.4.2 Once and Only Once Messaging

2417 ebXML supports two type of delivery Semantics:

- 2418 1. `OnceAndOnlyOnce`
- 2419 2. `BestEffort` (this is the default when unspecified)

2420 All type A request messages MUST utilize `BestEffort` `deliverySemantics`. All type B request
 2421 messages MUST utilize `OnceAndOnlyOnce` `deliverySemantics` with guaranteed ordered delivery.

2422 The OnceAndOnlyOnce deliverySemantics are used when guaranteed, single delivery of a
 2423 message is required. A Message Service Handler is expected to identify and ignore duplicate
 2424 type B message requests.

2425 **CAUTION: Messages sent using the BestEffort deliverySemantics may result in failed**
 2426 **delivery or multiple deliveries of the same message. Implementers should take caution when**
 2427 **using type A messages to ensure that failed message delivery or multiple message delivery**
 2428 **does not compromise the integrity of a system. This can be easily achieved by using**
 2429 **sequence numbers. Also, guaranteed ordered delivery does NOT insure that data contained**
 2430 **in the payload portion of a message is “processed” in the order they were received.**

2431 **4.7.5 Mapping Class of Delivery to Service/Action Pairs**

2432 In section 7 for each Service/Action pair there is a RECOMMENDED class of delivery.
 2433 Implementations following these recommendations are expected to have wire interoperability.

2434 **4.8 Sessions in OTA**

2435 Previous versions of OTA infrastructure defined both session-oriented and single-shot
 2436 conversations with differing conversational semantics and message control structures for each of
 2437 these.

2438 As there are currently no known use-cases among defined messages for the single-shot messaging
 2439 OTA has defined simpler session-oriented conversations and RECOMMENDS that
 2440 implementations always use a valid session during all conversations.

2441 **4.8.1 What we mean by ‘sessions’**

2442 The term ‘session’ has many different meanings depending on context. Within the context of
 2443 OTA infrastructure session is defined as an authenticated, authorized on-going conversation
 2444 between two implementations. Sessions have the following characteristics:

- 2445 • They are ongoing
- 2446 • Session initiation and session termination are relatively low overhead
- 2447 • Sessions may be long-lived (they may last for days, even weeks on end)
- 2448 • Sessions are leased²⁶
- 2449 • Messages belonging to a session are identified with that session via a conversation-id and
 2450 sequence numbers
- 2451 • Session conversation-ids provide a convenient ‘hook’ for implementation end-points to
 2452 maintain state (related, of course, to a particular session) e.g. security
- 2453 • Sessions follow a client/server model

2454 **4.8.2 What we do not mean by ‘sessions’**

2455 It is important to also explicitly note what we do not mean by the term ‘session’. In particular, the
 2456 concept of OTA sessions is completely orthogonal to the notion of ‘session control’ within

²⁶ 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

2457 booking conversations (those familiar with session control will recall that a ‘session’ is initiated
2458 implicitly and terminated explicitly via an ‘ET’ (End Transaction) message – analogous to a
2459 commit or an ‘IG’ (Ignore) message – analogous to a rollback).

2460 Currently there is no notion of traditional travel industry session control within defined OTA
2461 message sets, however as and when such a notion is defined it will be via message sets and new
2462 actions on corresponding services. This notion of ‘session-control’ belongs at a higher layer much
2463 closer to the application, and is conceptually entirely different from the infrastructure layer
2464 sessions being defined here.

2465 **4.8.3 The OTA Session service**

2466 The OTA session service is used to establish context for either a single request/response message
2467 exchange or a series of request/response message exchanges between parties. A Sender **MUST**
2468 establish a Session with a Receiver before any message exchanges containing OTA business
2469 transactions (units-of-work) will be allowed.

2470 Messages exchanged during a session may be related to a single OTA unit-of-work or multiple,
2471 unrelated units-of-work. OTA’s session service is also used to negotiate and establish operational
2472 parameters to govern communications that occur within a given session.

2473 The Session service defines four possible Actions:

- 2474 1. CreateRQ – sent by session initiator to establish a new session
- 2475 2. CreateRS – sent by recipient of a CreateRQ to grant or deny a session
- 2476 3. CloseRQ – sent by session initiator to close a session
- 2477 4. CloseRS – sent by the recipient of a CloseRQ to indicate session closure

2478 **4.8.3.1 Session/CreateRQ and Session/CreateRS**

2479 **4.8.3.1.1 Session/CreateRQ**

2480 This service/action pair is used to request the start of a new session. The initiator of
2481 Session/CreateRQ **MUST** construct an ebXML message header, following the requirements for
2482 Type A request messages. The ebXML Header MessageHeader/ConversationId **MUST** contain a
2483 value of **NULL**.

2484 A single payload container is used to pass a SessionControlRequest document that defines the
2485 operational parameters requested by the initiator. The SessionControlRequest document is an
2486 XML document containing the following information:

- 2487 • A **REQUIRED**, recurring OTA_Version element that is used to identify the various OTA
2488 versions supported by the sender. A single attribute, preference, containing a numerical
2489 value that is used to indicate the senders “preferred” version to be used during this
2490 session. The lowest preference value indicates the senders highest preference. A
2491 preference value of “0” or the absence of a preference attribute indicates no preference.
- 2492 • An **OPTIONAL** mode attribute that is used to identify “Test” or “Production” mode. If
2493 not specified mode=“Production” is assumed.
- 2494 • A **REQUIRED** version attribute (currently set to “1”)
- 2495 • An **OPTIONAL** LeaseRequestTime element that specifies the amount of time a session is
2496 expected to remain “alive”. When present, the element contains a positive integer value

2497 or “0” (zero). The element also contains one OPTIONAL attribute, cadence, that contains
2498 one of the following values:

- 2499 ○ seconds
- 2500 ○ minutes
- 2501 ○ hours
- 2502 ○ days

2503 When the LeaseRequestTime element contains a value of “0” (zero) or the element is not
2504 present, this is semantically equivalent to “infinite”. If the “cadence” attribute is not
2505 present, the value in LeaseRequestTime element SHALL be interpreted as “seconds”.

2506 • An OPTIONAL MaxIdleTime element that specifies the amount of time a session may
2507 remain idle before the server system “expires” the session. When present, the element
2508 contains a positive integer value or “0” (zero). The element also contains one
2509 OPTIONAL attribute, cadence, that contains one of the following values:

- 2510 ○ seconds
- 2511 ○ minutes
- 2512 ○ hours
- 2513 ○ days

2514 When the MaxIdleTime element contains a value of “0” (zero) or the element is not
2515 present, this is semantically equivalent to “infinite”. If the “cadence” attribute is not
2516 present, the value in MaxIdleTime element SHALL be interpreted as “seconds”.

2517 An example SessionControlRequest document follows:

```
2518 <SessionControlRequest mode="Production" version="1">
2519 <OTA_Version preference="1">2001C</OTA_Version>
2520 <OTA_Version preference="2">2001A</OTA_Version>
2521 <LeaseRequestTime cadence="seconds">60</LeaseRequestTime>
2522 <MaxIdleTime cadence="minutes">15</MaxIdleTime>
2523 </SessionControlRequest>
```

2524

2525 Additional elements and attributes may be added in a later version of this specification that could
2526 be used to identify OTA capabilities (e.g. supported Transactions) and other operational
2527 parameters.

2528 **4.8.3.1.2 Session/CreateRS**

2529 This service/action pair is sent in response to a Session/CreateRQ to inform the requesting party
2530 the status of the request. The sender of Session/CreateRS MUST construct an ebXML message
2531 header, with a RefToMessageId containing the value of MessageId in the Session/CreateRQ
2532 message, along with the other information required within response messages. The ebXML
2533 Header MessageHeader/ConversationId MUST contain a value of **NULL**.

2534 A single payload container is used to pass a SessionControlResponse document that defines the
2535 status of the request and other operational parameters that will be used during the session. The
2536 SessionControlResponse document is an XML document containing the following attributes and
2537 elements:

2538 A “status” attribute containing one of the following values:

- 2539 • Approved

- 2540 • Rejected
- 2541 • A single OTA_Version element identifies the OTA version that both parties agree to
2542 follow during the session.
- 2543 • A single ConversationId element containing the value of ConversationId that MUST be
2544 used in the MessageHeader/ConversationId element of all subsequent messages
2545 exchanged during this session.
- 2546 • An OPTIONAL LeaseRequestTime element that specifies the amount of time a server
2547 has will allow a session to remain “alive”. This element MUST be present, if the
2548 corresponding Session/CreateRQ message contained a LeaseRequestTime. The element
2549 contains a positive integer value or “0” (zero). The element also contains one
2550 OPTIONAL attribute, cadence, that contains one of the following values:
- 2551 ○ seconds
- 2552 ○ minutes
- 2553 ○ hours
- 2554 ○ days
- 2555 When the LeaseRequestTime element contains a value of “0” (zero) or the element is not
2556 present, this is semantically equivalent to “infinite”. If the “cadence” attribute is not
2557 present, the value in LeaseRequestTime element SHALL be interpreted as “seconds”.
- 2558 • An OPTIONAL MaxIdleTime element that specifies the amount of time a server will
2559 allow a session to remain idle before it “expires” the session. This element MUST be
2560 present, if the corresponding Session/CreateRQ message contained a MaxIdleTime.
2561 When present, the element contains a positive integer value or “0” (zero). The element
2562 also contains one OPTIONAL attribute, cadence, that contains one of the following
2563 values:
- 2564 ○ seconds
- 2565 ○ minutes
- 2566 ○ hours
- 2567 ○ days
- 2568 When the MaxIdleTime element contains a value of “0” (zero) or the element is not present, this
2569 is semantically equivalent to “infinite”. If the “cadence” attribute is not present, the value in
2570 MaxIdleTime element SHALL be interpreted as “seconds”.
- 2571 • Zero or more Reason elements indicating the reason why a Session was rejected. The
2572 Reason element MAY only be present when the status attribute of the
2573 SessionControlResponse element contains the value “Rejected”. When a session has been
2574 Rejected the SessionControlResponse document MUST NOT contain a ConversationId
2575 element.
- 2576 • Zero or one ServiceSupported element, see section 6 for details of this element. If the
2577 value of the status attribute is status=“Approved” a <ServicesSupported>
2578 element MUST be present.

2579 Following are two example SessionControlResponse documents:

```
2580 <SessionControlResponse version="1" status="Approved">
2581   <OTA_Version>2001C</OTA_Version>
2582   <ConversationId>20011027081907-862@host.imacompany.com</ConversationId>
2583   <LeaseRequestTime cadence="seconds">60</LeaseRequestTime>
2584   <MaxIdleTime cadence="minutes">15</MaxIdleTime>
```

```

2585     <ServicesSupported>
2586         <!-- content omitted for brevity -->
2587     </ServicesSupported>
2588 </SessionControlResponse>
2589
2590 <SessionControlResponse version="1" status="Rejected">
2591 <OTA_Version>2001C</OTA_Version>
2592 <Reason>No System Available to Process Request - Scheduled Maintenance</Reason>
2593 </SessionControlResponse>
2594

```

2595 The negotiated parameters returned on the response define the values that will be used for the
 2596 remainder of the session.

2597 **4.8.3.2 Session/CloseRQ and Session/CloseRS**

2598 **4.8.3.2.1 Session/CloseRQ**

2599 This service/action pair is used to request the closure of a session. The initiator of
 2600 Session/CloseRQ MUST construct an ebXML message header, following the requirements for
 2601 Type A request messages. The ebXML Header MessageHeader/ConversationId MUST contain
 2602 the value of the session to be closed. This is the same ConversationId value that was contained in
 2603 the Session/CreateRS SessionControlResponse document when the session was Accepted.

2604 The recipient of a Session/CloseRQ must take steps to ensure that only authorized parties are
 2605 allowed to close a session. Any Session/CloseRQ messages containing a ConversationId that was
 2606 not issued to the party issuing the Session/CloseRQ MUST be responded to with
 2607 Session/CloseRS message containing a <ErrorList><Error> indicating that the CloseRQ
 2608 has failed. and the session MUST remain active, if it is already active.

2609 A SESSION MAY ONLY BE CLOSED BY THE PARTY THAT WAS GRANTED THE
 2610 SESSION AND WITHIN THE CONTEXT OF THE SESSION BEING CLOSED (by using the
 2611 MessageHeader/ConversationId that was issued by the Acceptor of the session). Implementers
 2612 SHOULD maintain some identifying characteristics (e.g. IP address of original Requesting Party)
 2613 as verification before completing a Session/CloseRQ

2614 Once a session has been closed, either due to expiration or during a CloseRQ, no further message
 2615 exchanges are allowed over that session. Any attempt to send a message containing a
 2616 ConversationId for a session that has been closed MUST be responded to with a
 2617 <ErrorList><Error> indicating that the session is no longer active within the appropriate
 2618 response Service/Action message.

2619 There is no payload associated with this message.

2620 **4.8.3.2.2 Session/CloseRS**

2621 This service/action pair is sent in response to a Session/CloseRQ indicating that a session has
 2622 been closed. The sender of Session/CloseRS MUST construct an ebXML message header, with a
 2623 RefToMessageId containing the value of MessageId in the Session/CloseRQ message, along with
 2624 the other information required within response messages. The ebXML Header
 2625 MessageHeader/ConversationId MUST contain the value of the closed session.

2626 A SESSION MAY ONLY BE CLOSED BY THE PARTY THAT WAS GRANTED THE
 2627 SESSION AND WITHIN THE CONTEXT OF THE SESSION BEING CLOSED (identified by
 2628 the ConversationId that was issued by the Acceptor of the session). Implementers SHOULD

2629 maintain some identifying characteristics (e.g. IP address of original Requesting Party) as
 2630 verification before completing a Session/CloseRQ and issuing a Session/CloseRS.

2631 Once a session has been closed, either due to expiration or during a CloseRQ, no further message
 2632 exchanges are allowed over that session. Any attempt to send a message containing a
 2633 ConversationId for a session that has been closed MUST be responded to with a
 2634 <ErrorList><Error> indicating that the session is no longer active within the appropriate
 2635 response Service/Action message.

2636 There is no payload associated with this message.

2637 4.8.3.3 SessionControl Schema definition

2638 The following schema fragment formally defines SessionControl messages:

```

2639 <?xml version="1.0" encoding="UTF-8"?>
2640 <!-- Created and edited with 'vi' -->
2641 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2642   xmlns="http://www.opentravel.org/OTA"
2643   targetNamespace="http://www.opentravel.org/OTA"
2644   elementFormDefault="qualified">
2645
2646   <xs:annotation>
2647     <xs:documentation xml:lang="en">
2648       OTA v2001C Specification - SessionControl message definitions
2649       Copyright (C) 2001 Open Travel Alliance. All rights reserved.
2650     </xs:documentation>
2651   </xs:annotation>
2652
2653   <xs:element name="SessionControlRequest">
2654     <xs:annotation>
2655       <xs:documentation xml:lang="en">
2656         A SessionControlRequest is used to negotiate parameters to apply
2657         to an OTA session.
2658       </xs:documentation>
2659     </xs:annotation>
2660     <xs:complexType>
2661       <xs:sequence>
2662         <xs:element ref="OTA_Version" maxOccurs="unbounded" />
2663         <xs:element ref="LeaseRequestTime" minOccurs="0" />
2664         <xs:element ref="MaxIdleTime" minOccurs="0" />
2665       </xs:sequence>
2666       <xs:attribute name="version" type="versionNumber" use="required" />
2667       <xs:attribute name="mode" default="Production">
2668         <xs:simpleType>
2669           <xs:restriction base="xs:string">
2670             <xs:enumeration value="Test"/>
2671             <xs:enumeration value="Production"/>
2672           </xs:restriction>
2673         </xs:simpleType>
2674       </xs:attribute>
2675     </xs:complexType>
2676   </xs:element>
2677
2678   <xs:element name="OTA_Version" type="xs:string">
2679     <xs:attribute name="preferred" type="xs:integer" />
2680   </xs:element>
2681
2682   <xs:element name="LeaseRequestTime" type="xs:integer">
2683     <xs:attribute name="cadence" type="cadenceType" default="seconds"/>
2684   </xs:element>
2685
2686   <xs:simpleType name="cadenceType">
2687     <xs:restriction base="xs:string">
2688       <xs:enumeration value="seconds"/>
2689       <xs:enumeration value="minutes"/>
2690       <xs:enumeration value="hours"/>
2691       <xs:enumeration value="days"/>
  
```

```

2692     </xs:restriction>
2693 </xs:simpleType>
2694
2695 <xs:element name="MaxIdleTime" type="xs:integer">
2696   <xs:attribute name="cadence" type="cadenceType" default="seconds"/>
2697 </xs:element>
2698
2699 <xs:element name="SessionControlResponse">
2700   <xs:annotation>
2701     <xs:documentation xml:lang="en">
2702 A SessionControlResponse is used to negotiate parameters to apply
2703 to an OTA session.
2704     </xs:documentation>
2705   </xs:annotation>
2706   <xs:complexType>
2707     <xs:sequence>
2708       <xs:element ref="OTA_Version" />
2709       <xs:element ref="ConversationId" />
2710       <xs:element ref="LeaseRequestTime" minOccurs="0" />
2711       <xs:element ref="MaxIdleTime" minOccurs="0" />
2712       <xs:element ref="Reason" minOccurs="0" maxOccurs="unbounded" />
2713       <xs:element ref="ServicesSupported" minOccurs="0" />
2714     </xs:sequence>
2715     <xs:attribute name="version" type="versionNumber" use="required" />
2716     <xs:attribute name="status" use="required">
2717       <xs:simpleType>
2718         <xs:restriction base="xs:string">
2719           <xs:enumeration value="Accepted"/>
2720           <xs:enumeration value="Rejected"/>
2721         </xs:restriction>
2722       </xs:simpleType>
2723     </xs:attribute>
2724   </xs:complexType>
2725 </xs:element>
2726
2727 <xs:element name="ConversationId" type="xs:string" />
2728 <xs:element name="Reason" type="xs:string" />
2729
2730 <xsd:simpleType name="versionNumber">
2731   <xsd:annotation>
2732     <xsd:documentation xml:lang="en">
2733 All OTA version numbers are unsigned integers in the range 1..9999.
2734 Version numbers should begin with the value '1' and be incremented
2735 each time a message is revised.
2736     </xsd:documentation>
2737   </xsd:annotation>
2738
2739   <xsd:restriction base="xsd:unsignedShort">
2740     <xsd:minInclusive value="1"/>
2741     <xsd:maxInclusive value="9999"/>
2742   </xsd:restriction>
2743 </xsd:simpleType>
2744 </xs:schema>

```

2745 4.8.4 Securing OTA Sessions

2746 All data passed over an OTA session MUST be protected from unauthorized parties.
 2747 Additionally, all servers used by OTA implementers MUST maintain access control in order to
 2748 prevent unauthorized use of an OTA system. The security and integrity of an OTA system should
 2749 be a top priority for all OTA implementers.

2750 4.8.4.1 Basic-authorization and SSL

2751 All OTA implementers MUST support, at a minimum, SSL version 3.0 using a minimum key
 2752 size of 128 bits for symmetric cryptographic algorithms and a key size of 2048 bits for
 2753 asymmetric cryptographic algorithms (e.g. public key). Presently, only servers are required to

2754 implement Digital Certificates. Some future version may require clients to implement Digital
 2755 Certificates for authentication purposes during the establishment of a SSL connection. During the
 2756 establishment of an SSL connection OTA servers are required to present a Digital Certificate, as
 2757 part of the SSL handshake. All OTA clients MUST accept self signed Digital Certificates as well
 2758 as those that have been signed by a recognized Certificate Authority (e.g. Verisign, Entrust,
 2759 Thawte, et al).

2760 All OTA implementers MUST use HTTP Basic Authentication (usernames and passwords) (ref:
 2761 RFC 2617) to control access to an OTA system. Any party that fails to provide an authorized
 2762 username/password pair in the Authorization header of an HTTP request when sending an OTA
 2763 Request or Response message using HTTP POST method will likely generate an HTTP 401
 2764 error²⁷. Implementers are expected to secure the transport of sensitive username/password data by
 2765 only transporting this information over an established SSL connection with a server that is known
 2766 to belong to your trading partner.

2767 **4.8.4.2 Towards deeper security**

2768 EbXML offers many levels of security beyond our base recommendation of basic-authorization
 2769 over SSL, including the ability to:

- 2770 • Digitally sign payloads
- 2771 • Encrypt payloads
- 2772 • Digitally sign and encrypt payloads
- 2773 • Require client-side X.509 certificates on HTTP/S sessions

2774 Different commercial ebXML implementations are expected to have differing level of support for
 2775 these additional security features. Beyond our recommendation for a basic authenticated and
 2776 secure channel it is up to trading partners to agree on any deeper level of security to be applied to
 2777 their eCommerce exchanges.

2778 **4.9 Web Services Description for OTA ebXML**

2779 The following web services description language [wsdl]²⁸ fragment formally defines OTA type A
 2780 and type B requests and responses from a web services perspective, should OTA partners want to
 2781 make use of a UDDI registry which supports WSDL:

```

2782 <?xml version="1.0" encoding="utf-8"?>
2783 <definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
2784   xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
2785   xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
2786   xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
2787   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
2788   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
2789   targetNamespace="http://www.ebxml.org/namespaces/messageHeader"
2790   xmlns="http://schemas.xmlsoap.org/wsdl/">
2791
2792   <types>
2793     <schema targetNamespace=http://www.ebxml.org/namespaces/messageHeader"
2794       xmlns="http://www.w3.org/2000/10/XMLSchema">
2795
2796
```

²⁷ The HTTP 401 error code indicates an unauthorized attempt to access a restricted resource

²⁸ [wsdl] – Web Services Description Language – a W3C Technical Recommendation available at:
<http://www.w3c.org/TR/wsdl>

```

2797 <message name="TypeARequest">
2798   <part name="MessageHeader" type="eb:MessageHeader" />
2799   <part name="Via" type="eb:Via" />
2800   <part name="Manifest" type="eb:Manifest" />
2801 </message>
2802
2803 <message name="TypeAResponse">
2804   <part name="MessageHeader" type="eb:MessageHeader" />
2805   <part name="ErrorList" type="eb:ErrorList" />
2806   <part name="Manifest" type="eb:Manifest" />
2807 </message>
2808
2809 <message name="TypeBRequest">
2810   <part name="MessageHeader" type="eb:MessageHeader" />
2811   <part name="Manifest" type="eb:Manifest" />
2812 </message>
2813 <message name="TypeBResponse">
2814   <part name="MessageHeader" type="eb:MessageHeader" />
2815   <part name="Acknowledgement" type="eb:Acknowledgement" />
2816   <part name="ErrorList" type="eb:ErrorList" />
2817 </message>
2818
2819 <portType name="ebXML">
2820   <operation name="TypeARequest">
2821     <input message="eb:TypeARequest" />
2822     <output message="eb:TypeAResponse" />
2823   </operation>
2824   <operation name="TypeBRequest">
2825     <input message="eb:TypeBRequest" />
2826     <output message="eb:TypeBResponse" />
2827   </operation>
2828 </portType>
2829
2830 <binding name="ebxmlhandler" type="eb:ebXML">
2831   <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
2832     style="document" />
2833
2834   <operation name="TypeARequest">
2835     <soap:operation soapAction="ebXML" style="document" />
2836     <input>
2837       <soap:header
2838         namespace="http://www.ebxml.org/namespaces/messageHeader" />
2839       <soap:body
2840         namespace="http://www.ebxml.org/namespaces/messageHeader" />
2841       <mime:mimeXml part="Body" />
2842     </input>
2843     <output>
2844       <soap:header
2845         namespace="http://www.ebxml.org/namespaces/messageHeader" />
2846       <soap:body
2847         namespace="http://www.ebxml.org/namespaces/messageHeader" />
2848       <mime:mimeXml part="Body" />
2849     </output>
2850   </operation>
2851
2852   <operation name="TypeBRequest">
2853     <soap:operation soapAction="ebXML" style="document" />
2854     <input>
2855       <soap:header
2856         namespace="http://www.ebxml.org/namespaces/messageHeader" />
2857       <soap:body
2858         namespace="http://www.ebxml.org/namespaces/messageHeader" />
2859       <mime:mimeXml part="Body" />
2860     </input>
2861     <output>
2862       <soap:header
2863         namespace="http://www.ebxml.org/namespaces/messageHeader" />
2864     </output>
2865   </operation>
2866 </binding>
2867

```

```
2868 <service name="ebxmlhandler">
2869   <port name="ebxml" binding="eb:ebxmlhandler">
2870     <http:address location="http://b2b.company.com/servlet/ebxml" />
2871   </port>
2872 </service>
2873 </definitions>
2874
```

2875 **5 OTA Update Messages**

2876 As described before, OTA update messages were designed with the following goals:

- 2877 • Minimizing the size of a payload on the wire to represent an update transaction
- 2878 • Defining an explicit representation for what has changed
- 2879 • Defining a representation with a clear and simple conceptual model
- 2880 • Creating a representation that is content-independent and general-purpose in nature so as to
2881 be reusable throughout future OTA specifications
- 2882 • Providing a simple-to-implement "replace" option to allow developers to get simpler
2883 implementations running quickly - at the expense of the first 2 goals (representation of
2884 change and size of message) above.

2885 **5.1 Representing change in XML**

2886 Representing change is not a problem unique to the OTA. The approach presented in this
2887 specification assumes use of the following:

- 2888 • library utilities²⁹ that can compare before and after images of a particular XML document
2889 and then generate a succinct representation of the differences (conceptually similar to a
2890 GNU *diff* utility)
- 2891 • a representation for differences that is both human readable and understandable by
2892 automated tools
- 2893 • library utilities that can take an XML document, apply a differences document and
2894 generate the same after image (conceptually similar to a GNU *patch* utility)

2895 Applying an Update action involves a tree-to-tree comparison, similar to well-known operations
2896 outside the XML arena. This specification assumes that prior to sending an <OTA_UpdateRQ>
2897 message, the sender has used standard libraries to generate the differences between the 'before'
2898 and desired 'after' images of the XML document. Implementors may wish to consult
2899 documentation of algorithms for tree-to-tree comparison and correction in computer science
2900 publications³⁰, to provide the background for understanding this XML-specific approach.

2901 **5.2 Position Representation with XPath**

2902 The general concept of an OTA generic update request is to send a <Position> element
2903 followed by one or more operations to be applied at that position. XPath is a well-known W3C
2904 recommendation for representing a node in an XML document. The two best known applications
2905 of XPath are within XSL and XLink (themselves XML recommendations).

2906 Within the OTA update representation, XPath is used to reference a specific element within a
2907 source document. This XPath is encoded within the *XPath* attribute in the <Position>

²⁹ an open source implementation of such a library is available from OTA member VM Systems, Inc. (see <http://www.vmguy.com/vmtools/>)

³⁰ 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.qcis.queensu.ca/pub/reports/1995-372.ps>)

2908 element. In OTA update messages, the *XPath* attribute within any given <Position> will
 2909 always refer to an element, and this explanation therefore is restricted to that proper subset of
 2910 XPath notation which refers to elements. The representation of <Position> followed by one or
 2911 more operations is used, as updates will often consist of more than one operation to be performed
 2912 at the same position. This representation will often be shorter than the alternative of embedding
 2913 an XPath position explicitly within each operation.

2914 It is important to note that when processing a differences document, insertion or deletion of nodes
 2915 may invalidate subsequent XPath references. This is addressed in the section "Order of
 2916 Representation and Application" (See Section 5.5).

2917 **5.3 Operands**

2918 In keeping with the design goals of minimal representation and explicit representation of change,
 2919 changes are represented as occurring at four different levels, using the following operands:

- 2920 • *Attribute* - performs an operation on the attribute *name* of the currently selected Element
- 2921 • *Element* - performs an operation on the structure (i.e. content or children) of the currently
 2922 selected element (but not its attributes)
- 2923 • *Subtree* - allows for the grafting or pruning of an element which has its own children
- 2924 • *Root* - this special attribute is ONLY used for a simple 'replace' alternative where an updated
 2925 representation of the entire object is sent as opposed to sending incremental differences

2926 In the case of <Element> and <Subtree> operands, insert operations are performed on
 2927 children of the selected Element (selection is made via the <Position> *Xpath* attribute).
 2928 Children are always specified via the *Child* attribute, which is a one-relative integer where
 2929 children positions are specified left-to-right, with position one indicating the leftmost child.

2930 **5.4 Operations**

2931 Operations are specified on an operand via the Operation attribute. Operations are "*insert*",
 2932 "*modify*" and "*delete*", though the "*modify*" operation is not applicable to the operand
 2933 <Subtree>. The operand <Root> is the only operand to have the special "*replace*" operation.

2934 The following table describes the result of each operation on its associated operand:

Operand	Operation	Description
Attribute	insert	The attribute <i>name</i> with value <i>value</i> will be added to selected element
Attribute	modify	The value of the attribute <i>name</i> will be changed to <i>value</i> on the selected element
Attribute	delete	The attribute <i>name</i> 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 <i>Child</i> (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 <i>Child</i> (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

2935 Although it is possible to replace the entire tree by performing a <Subtree> *delete* operation on
 2936 the root, it is not possible to then insert a new subtree as a replacement. The <Root> *replace*
 2937 operation is provided for this purpose.

2938 The syntax for an <OTA_UpdateRQ> is formally defined in the following schema fragment:

2939 **Schema 7 - <OTA_UpdateRQ>:**

```

2940 <?xml version="1.0" encoding="UTF-8"?>
2941 <!-- Created and edited with 'vi' -->
2942 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2943           xmlns="http://www.opentravel.org/OTA"
2944           targetNamespace="http://www.opentravel.org/OTA"
2945           elementFormDefault="qualified">
2946
2947   <xs:annotation>
2948     <xs:documentation xml:lang="en">
2949       OTA v2001C Specification - Generic OTA_UpdateRQ message definition
2950       Copyright (C) 2001 Open Travel Alliance. All rights reserved.
2951     </xs:documentation>
2952   </xs:annotation>
2953   <xs:include schemaLocation="OTA_v2ent.xsd"/>
2954   <xs:include schemaLocation="OTA_POS.xsd"/>
2955
2956   <xs:element name="OTA_UpdateRQ">
2957     <xs:annotation>
2958       <xs:documentation xml:lang="en">
2959         This element represents the incremental changes to the business document
2960         referred to by UniqueId.
2961       </xs:documentation>
2962     </xs:annotation>
2963     <xs:complexType>
2964       <xs:sequence>
2965         <xs:element ref="UniqueId"/>
2966         <xs:element ref="POS" minOccurs="0"/>
2967         <xs:element ref="Position" maxOccurs="unbounded"/>
2968       </xs:sequence>
2969       <xs:attributeGroup ref="OTA_PayloadStdAttributes"/>
2970       <xs:attributeGroup ref="ReqRespVersion"/>
2971     </xs:complexType>
2972   </xs:element>
2973
2974   <xs:element name="Position">

```

```

2975     <xs:complexType>
2976       <xs:choice>
2977         <xs:sequence>
2978           <xs:element ref="Attribute" minOccurs="0" maxOccurs="unbounded" />
2979           <xs:element ref="Element" minOccurs="0" maxOccurs="unbounded" />
2980           <xs:element ref="Subtree" minOccurs="0" maxOccurs="unbounded" />
2981         </xs:sequence>
2982         <xs:element ref="Root" />
2983       </xs:choice>
2984       <xs:attribute name="XPath" type="xs:string" use="required" />
2985     </xs:complexType>
2986 </xs:element>
2987 <xs:element name="Attribute">
2988   <xs:complexType>
2989     <xs:attribute name="Operation" use="required">
2990       <xs:simpleType>
2991         <xs:restriction base="xs:string">
2992           <xs:enumeration value="insert" />
2993           <xs:enumeration value="modify" />
2994           <xs:enumeration value="delete" />
2995         </xs:restriction>
2996       </xs:simpleType>
2997     </xs:attribute>
2998     <xs:attribute name="Name" type="xs:string" use="required" />
2999     <xs:attribute name="Value" type="xs:string" />
3000   </xs:complexType>
3001 </xs:element>
3002 <xs:element name="Element">
3003   <xs:complexType>
3004     <xs:sequence>
3005       <xs:any />
3006     </xs:sequence>
3007     <xs:attribute name="Operation" use="required">
3008       <xs:simpleType>
3009         <xs:restriction base="xs:string">
3010           <xs:enumeration value="insert" />
3011           <xs:enumeration value="modify" />
3012           <xs:enumeration value="delete" />
3013         </xs:restriction>
3014       </xs:simpleType>
3015     </xs:attribute>
3016     <xs:attribute name="Child" type="xs:string" />
3017   </xs:complexType>
3018 </xs:element>
3019 <xs:element name="Subtree">
3020   <xs:complexType>
3021     <xs:sequence>
3022       <xs:any />
3023     </xs:sequence>
3024     <xs:attribute name="Operation" use="required">
3025       <xs:simpleType>
3026         <xs:restriction base="xs:string">
3027           <xs:enumeration value="insert" />
3028           <xs:enumeration value="delete" />
3029         </xs:restriction>
3030       </xs:simpleType>
3031     </xs:attribute>
3032     <xs:attribute name="Child" type="xs:string" />
3033   </xs:complexType>
3034 </xs:element>
3035 <xs:element name="Root">
3036   <xs:complexType>
3037     <xs:sequence>
3038       <xs:any />
3039     </xs:sequence>
3040     <xs:attribute name="Operation" use="required">
3041       <xs:simpleType>
3042         <xs:restriction base="xs:string">
3043           <xs:enumeration value="replace" />
3044         </xs:restriction>
3045       </xs:simpleType>

```

```

3046         </xs:attribute>
3047     </xs:complexType>
3048 </xs:element>
3049 </xs:schema>

```

3050 **5.5 Order of Representation and Application**

3051 The XPath notation is used to determine the position in an XML document where operations are
 3052 to be applied. As that position is specified via an XPath, it is entirely possible that application of
 3053 an operation will invalidate subsequent XPath expressions if care is not taken explicitly in the
 3054 order of presentation.

3055 An XML document is inherently a tree, and when considering an XML document, the most
 3056 natural order for presentation is a depth-first pre-order traversal (this is the way in which an XML
 3057 document is represented in XML notation). Unfortunately, that order is the one that is most likely
 3058 to invalidate XPaths when applying differences in order.

3059 Therefore, for purposes of representing differences in an update message, positions and
 3060 operations are presented in an order which favors a post-order traversal³¹. This ensures that
 3061 differences may be applied sequentially to a source document to transform it into a target without
 3062 invalidating the XPath of any subsequent unprocessed difference.

3063 A brief illustration will help make this point clearer. Assuming the following abstract source
 3064 document:

```

3065 <A>
3066     <B>
3067         <C />
3068         <D>
3069             <E />
3070         </D>
3071     </B>
3072     <F>
3073         <G>
3074             <H />
3075         </G>
3076         <I>
3077             <J />
3078         </I>
3079     </F>
3080 </A>

```

3081
 3082 The depth-first pre-order traversal presents elements in the following sequence:

```

3083 <A><B><C><D><E><F><G><H><I><J>
3084

```

3085 However, a post-order traversal presents elements in the following alternate sequence:

```

3086 <C><E><D><B><H><G><J><I><F><A>
3087

```

3088 When considering tree-traversals in the context of XML, perhaps the easiest way to think of it is
 3089 as follows:

- 3090 • depth-first pre-order traversal presents elements in the order that opening tags occur
- 3091 • post-order traversal presents elements in the order that element closure occurs (i.e. follow the
 3092 closing tags)

³¹ It should be noted that the fastest known algorithms for tree-to-tree correction operate on a post-order traversal

3093 An additional requirement for order is placed on repeating sequences of elements, as follows:
 3094 When operation(s) are to be performed on more than one element in a repeating sequence the
 3095 <Position> for those elements in the sequence must be presented from right-to- left, i.e. from
 3096 the largest index to the smallest.
 3097 By presenting and applying differences using a post-order traversal representation (and reverse
 3098 index order for repeating sequences) operations applied to the current position cannot break the
 3099 XPath to a subsequent position as later operations are deeper within the tree (closer to the root)
 3100 down any given branch.

3101 **5.6 Update Examples**

3102 Given the difference representation above, the following examples illustrate this technique. These
 3103 examples illustrate a chain of updates, showing before and after images at each step. All these
 3104 changes are designed to be incremental, that is, the "after" image from the previous update
 3105 becomes the "before" image for the next update.

3106 **Example 13 - initial "before" document image**

```

3107 <Profile>
3108   <Customer>
3109     <PersonName NameType="Default">
3110       <NamePrefix>Mr.</NamePrefix>
3111       <GivenName>George</GivenName>
3112       <MiddleName>A.</MiddleName>
3113       <Surname>Smith</Surname>
3114     </PersonName>
3115     <TelephoneInfo PhoneTech="Voice" PhoneUse="Work" >
3116       <Telephone>
3117         <AreaCityCode>206</AreaCityCode>
3118         <PhoneNumber>813-8698</PhoneNumber>
3119       </Telephone>
3120     </TelephoneInfo>
3121     <PaymentForm>
3122       ...
3123     </PaymentForm>
3124     <Address>
3125       <StreetNmbr POBox="4321-01">1200 Yakima St</StreetNmbr>
3126       <BldgRoom>Suite 800</BldgRoom>
3127       <CityName>Seattle</CityName>
3128       <StateProv PostalCode="98108">WA</StateProv>
3129       <CountryName>USA</CountryName>
3130     </Address>
3131   </Customer>
3132 </Profile>
  
```

3133

3134 Let's begin by making a simple change – we'll update the profile to reflect a change in the
 3135 customer's area code from '206' to '253'. The simplest way to implement this is to send the
 3136 entire profile as a change, but this is a large message to send for a simple change of area code (see
 3137 the example below):

3138 **Example 14 - update of AreaCode using <Root Operation="replace"/>**

```

3139 <?xml version="1.0" encoding="UTF-8"?>
3140 <!-- created by 'DiffGen' using VMTTools 0.3 (http://www.vmgays.com/vmtools/) -->
3141 <OTA_UpdaterQ xmlns="http://www.opentravel.org/OTA"
3142   xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
3143   xsi:schemaLocation="http://www.opentravel.org/OTA
3144   OTA_UpdaterQ.xsd"
3145   ReqRespVersion="2">
3146   <UniqueId Type="Profile" Id="9876543210"
3147     URL="http://www.vmgays.com/OTAEngine" Instance="1" />
  
```

```

3148 <Position XPath="/Profile">
3149   <Root Operation="replace">
3150     <Profile xmlns="">
3151       <Customer>
3152         <PersonName NameType="Default">
3153           <NamePrefix>Mr.</NamePrefix>
3154           <GivenName>George</GivenName>
3155           <MiddleName>A.</MiddleName>
3156           <Surname>Smith</Surname>
3157         </PersonName>
3158         <TelephoneInfo PhoneTech="Voice" PhoneUse="Work">
3159           <Telephone>
3160             <AreaCityCode>253</AreaCityCode>
3161             <PhoneNumber>813-8698</PhoneNumber>
3162           </Telephone>
3163         </TelephoneInfo>
3164         <PaymentForm>...</PaymentForm>
3165         <AddressInfo>
3166           <Address>
3167             <StreetNmbr PO_Box="4321-01">1200 Yakima St</StreetNmbr>
3168             <BldgRoom>Suite 800</BldgRoom>
3169             <CityName>Seattle</CityName>
3170             <StateProv PostalCode="98108">WA</StateProv>
3171             <CountryName>USA</CountryName>
3172           </Address>
3173         </AddressInfo>
3174       </Customer>
3175     </Profile>
3176   </Root>
3177 </Position>
3178 </OTA_UpdateRQ>
3179

```

3180 If we use the more succinct representation supported by OTA generic updates this same change
3181 of area code from '206' to '253' can be represented by the much more succinct and expressive
3182 message below:

3183 **Example 15 - change area code use <Element Operation="modify">:**

```

3184 <?xml version="1.0" encoding="UTF-8"?>
3185 <!-- created by 'DiffGen' using VMTTools 0.3 (http://www.vmguy.com/vmtools/) -->
3186 <OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
3187   xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
3188   xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
3189   ReqRespVersion="2">
3190   <UniqueId Type="Profile" Id="9876543210"
3191     URL="http://www.vmguy.com/OTAEngine" Instance="1" />
3192   <Position XPath="/Profile/Customer/TelephoneInfo/Telephone/AreaCityCode">
3193     <Element Operation="modify">253</Element>
3194   </Position>
3195 </OTA_UpdateRQ>
3196

```

3197 Note that the operation "modify" replaces the PCDATA within the element. In order to delete
3198 the data, the update request is sent with empty content in the element. (A "delete" operation
3199 deletes the entire element.)

3200 **Example 16 - update of RelatedTraveler using <Subtree Operation="insert">**

```

3201 <?xml version="1.0" encoding="UTF-8"?>
3202 <!-- created by 'DiffGen' using VMTTools 0.3 (http://www.vmguy.com/vmtools/) -->
3203 <OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
3204   xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
3205   xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
3206   ReqRespVersion="2">
3207   <UniqueId Type="Profile" Id="9876543210"
3208     URL="http://www.vmguy.com/OTAEngine" Instance="2" />
3209   <Position XPath="/Profile/Customer">

```

```

3210 <Subtree Operation="insert" Child="5">
3211 <RelatedTraveler Relation="Child">
3212 <PersonName>
3213 <GivenName>Devin</GivenName>
3214 <MiddleName>R.</MiddleName>
3215 <Surname>Smith</Surname>
3216 </PersonName>
3217 </RelatedTraveler>
3218 </Subtree>
3219 <Subtree Operation="insert" Child="6">
3220 <RelatedTraveler Relation="Child">
3221 <PersonName>
3222 <GivenName>Amy</GivenName>
3223 <MiddleName>E.</MiddleName>
3224 <Surname>Smith</Surname>
3225 </PersonName>
3226 </RelatedTraveler>
3227 </Subtree>
3228 <Subtree Operation="insert" Child="7">
3229 <RelatedTraveler Relation="Child">
3230 <PersonName>
3231 <GivenName>Alfred</GivenName>
3232 <MiddleName>E.</MiddleName>
3233 <Surname>Newman</Surname>
3234 </PersonName>
3235 </RelatedTraveler>
3236 </Subtree>
3237 </Position>
3238 </OTA_UpdateRQ>
3239

```

3240 Example 17 - document image after <Subtree Operation="insert">

```

3241 <Profile>
3242 <Customer>
3243 <PersonName NameType="Default">
3244 <NamePrefix>Mr.</NamePrefix>
3245 <GivenName>George</GivenName>
3246 <MiddleName>A.</MiddleName>
3247 <Surname>Smith</Surname>
3248 </PersonName>
3249 <TelephoneInfo PhoneTech="Voice" PhoneUse="Work">
3250 <Telephone>
3251 <AreaCityCode>253</AreaCityCode>
3252 <PhoneNumber>813-8698</PhoneNumber>
3253 </Telephone>
3254 </TelephoneInfo>
3255 <PaymentForm>
3256 ...
3257 </PaymentForm>
3258 <Address>
3259 <StreetNmbr POBox="4321-01">1200 Yakima St</StreetNmbr>
3260 <BldgRoom>Suite 800</BldgRoom>
3261 <CityName>Seattle</CityName>
3262 <StateProv PostalCode="98108">WA</StateProv>
3263 <CountryName>USA</CountryName>
3264 </Address>
3265 <RelatedTraveler Relation="Child">
3266 <PersonName>
3267 <GivenName>Devin</GivenName>
3268 <MiddleName>R.</MiddleName>
3269 <Surname>Smith</Surname>
3270 </PersonName>
3271 </RelatedTraveler>
3272 <RelatedTraveler Relation="Child">
3273 <PersonName>
3274 <GivenName>Amy</GivenName>
3275 <MiddleName>E.</MiddleName>
3276 <Surname>Smith</Surname>
3277 </PersonName>

```

```

3278     </RelatedTraveler>
3279     <RelatedTraveler Relation="Child"
3280       <PersonName>
3281         <GivenName>Alfred</GivenName>
3282         <MiddleName>E.</MiddleName>
3283         <Surname>Newman</Surname>
3284       </PersonName>
3285     </RelatedTraveler>
3286   </Customer>
3287 </Profile>
3288

```

3289 **Example 18 - update of document using <Subtree Operation="delete"/>**

3290 This operation will delete the first and third Related Travelers. (Note the required order of the
3291 operations):

```

3292 <?xml version="1.0" encoding="UTF-8"?>
3293 <!-- created by 'DiffGen' using VMTTools 0.3 (http://www.vmguy.com/vmtools/) -->
3294 <OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
3295   xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
3296   xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
3297   ReqRespVersion="2">
3298   <UniqueId Type="Profile" Id="9876543210"
3299     URL="http://www.vmguy.com/OTAEngine" Instance="3" />
3300   <Position XPath="/Profile/Customer/RelatedTraveler[3]">
3301     <Subtree Operation="delete" />
3302   </Position>
3303   <Position XPath="/Profile/Customer/RelatedTraveler[1]">
3304     <Subtree Operation="delete" />
3305   </Position>
3306 </OTA_UpdateRQ>
3307

```

3308 **Note:** Use of the <Delete> element removes a designated element from the tree. Any children
3309 of the element removed will move up to the grandparent of the element. If the desired result is the
3310 removal of the element and all of its children, the element <Subtree> paired with the operation
3311 "delete" should be used, as in the example above.

3312 **Example 19 - Document image after <Subtree Operation="delete"/>**

```

3313 <Profile>
3314   <Customer>
3315     <PersonName NameType="Default">
3316       <NamePrefix>Mr.</NamePrefix>
3317       <GivenName>George</GivenName>
3318       <MiddleName>A.</MiddleName>
3319       <Surname>Smith</Surname>
3320     </PersonName>
3321     <TelephoneInfo PhoneTech="Voice" PhoneUse="Work" >
3322       <Telephone>
3323         <AreaCityCode>253</AreaCityCode>
3324         <PhoneNumber>813-8698</PhoneNumber>
3325       </Telephone>
3326     </TelephoneInfo>
3327     <PaymentForm>
3328       ...
3329     </PaymentForm>
3330     <AddressInfo>
3331       <Address>
3332         <StreetNmbr POBox="4321-01">1200 Yakima St</StreetNmbr>
3333         <BldgRoom>Suite 800</BldgRoom>
3334         <CityName>Seattle</CityName>
3335         <StateProv PostalCode="98108">WA</StateProv>
3336         <CountryName>USA</CountryName>
3337       </Address>
3338     </AddressInfo>
3339     <RelatedTraveler Relation="Child" >

```

```

3340         <PersonName>
3341             <GivenName>Amy</GivenName>
3342             <MiddleName>E.</MiddleName>
3343             <Surname>Smith</Surname>
3344         </PersonName>
3345     </RelatedTraveler>
3346 </Customer>
3347 </Profile>
3348

```

3349 **Example 20 - Update document using <Attribute Operation="modify"/>**

3350 This examples changes the TelephoneInfo PhoneUse attribute.

```

3351 <?xml version="1.0" encoding="UTF-8"?>
3352 <!-- created by 'DiffGen' using VMTTools 0.3 (http://www.vmguy.com/vmtools/) -->
3353 <OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
3354             xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
3355             xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
3356             ReqRespVersion="2">
3357     <UniqueId Type="Profile" Id="9876543210"
3358             URL="http://www.vmguy.com/OTAEngine" Instance="4" />
3359     <Position XPath="/Profile/Customer/TelephoneInfo">
3360         <Attribute Name="PhoneUse" Operation="modify" Value="Home" />
3361     </Position>
3362 </OTA_UpdateRQ>
3363

```

3364 **Example 21 - Document image after <Attribute Operation="modify"/>**

```

3365 <Profile>
3366     <Customer>
3367         <PersonName NameType="Default">
3368             <NamePrefix>Mr.</NamePrefix>
3369             <GivenName>George</GivenName>
3370             <MiddleName>A.</MiddleName>
3371             <Surname>Smith</Surname>
3372         </PersonName>
3373         <TelephoneInfo PhoneTech="Voice" PhoneUse="Home">
3374             <Telephone>
3375                 <AreaCityCode>253</AreaCityCode>
3376                 <PhoneNumber>813-8698</PhoneNumber>
3377             </Telephone>
3378         </TelephoneInfo>
3379         <PaymentForm>
3380             ...
3381         </PaymentForm>
3382         <AddressInfo>
3383             <Address>
3384                 <StreetNmbr PO_Box="4321-01">1200 Yakima St</StreetNmbr>
3385                 <BldgRoom>Suite 800</BldgRoom>
3386                 <CityName>Seattle</CityName>
3387                 <StateProv PostalCode="98108">WA</StateProv>
3388                 <CountryName>USA</CountryName>
3389             </Address>
3390         </AddressInfo>
3391         <RelatedTraveler Relation="Child">
3392             <PersonName>
3393                 <GivenName>Amy</GivenName>
3394                 <MiddleName>E.</MiddleName>
3395                 <Surname>Smith</Surname>
3396             </PersonName>
3397         </RelatedTraveler>
3398     </Customer>
3399 </Profile>
3400

```

3401 **Example 22 - Update document using <Attribute Operation="delete"/>**

3402 This operation will delete the PO_Box attribute of StreetNmbr.

```

3403 <?xml version="1.0" encoding="UTF-8"?>
3404 <!-- created by 'DiffGen' using VMTTools 0.3 (http://www.vmguy.com/vmtools/) -->
3405 <OTA_UpdaterRQ xmlns="http://www.opentravel.org/OTA"
3406     xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
3407     xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdaterRQ.xsd"
3408     ReqRespVersion="2">
3409   <UniqueId Type="Profile" Id="9876543210"
3410     URL="http://www.vmguy.com/OTAEngine" Instance="5" />
3411   <Position XPath="/Profile/Customer/AddressInfo/Address/StreetNmbr">
3412     <Attribute Name="PO_Box" Operation="delete" />
3413   </Position>
3414 </OTA_UpdaterRQ>
3415

```

3416 **Example 23 - Document image after <Attribute Operation="delete"/>**

```

3417 <Profile>
3418   <Customer>
3419     <PersonName NameType="Default">
3420       <NamePrefix>Mr.</NamePrefix>
3421       <GivenName>George</GivenName>
3422       <MiddleName>A.</MiddleName>
3423       <Surname>Smith</Surname>
3424     </PersonName>
3425     <TelephoneInfo PhoneTech="Voice" PhoneUse="Home">
3426       <Telephone>
3427         <AreaCityCode>253</AreaCityCode>
3428         <PhoneNumber>813-8698</PhoneNumber>
3429       </Telephone>
3430     </TelephoneInfo>
3431     <PaymentForm>
3432       ...
3433     </PaymentForm>
3434     <AddressInfo>
3435       <Address>
3436         <StreetNmbr>1200 Yakima St</StreetNmbr>
3437         <BldgRoom>Suite 800</BldgRoom>
3438         <CityName>Seattle</CityName>
3439         <StateProv PostalCode="98108">WA</StateProv>
3440         <CountryName>USA</CountryName>
3441       </Address>
3442     </AddressInfo>
3443     <RelatedTraveler Relation="Child">
3444       <PersonName>
3445         <GivenName>Amy</GivenName>
3446         <MiddleName>E.</MiddleName>
3447         <Surname>Smith</Surname>
3448       </PersonName>
3449     </RelatedTraveler>
3450   </Customer>
3451 </Profile>
3452

```

3453 **Example 24 - Compound update using multiple operations**

3454 These compound operations will insert the Gender attribute on Customer, delete the
3455 MiddleName, and insert NamePrefix on the Related Traveler:

```

3456 <?xml version="1.0" encoding="UTF-8"?>
3457 <!-- created by 'DiffGen' using VMTTools 0.3 (http://www.vmguy.com/vmtools/) -->
3458 <OTA_UpdaterRQ xmlns="http://www.opentravel.org/OTA"
3459     xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
3460     xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdaterRQ.xsd"
3461     ReqRespVersion="2">
3462   <UniqueId Type="Profile" Id="9876543210"
3463     URL="http://www.vmguy.com/OTAEngine" Instance="6" />
3464   <Position XPath="/Profile/Customer/RelatedTraveler/PersonName">
3465     <Element Operation="insert" Child="1">

```

```

3466         <NamePrefix xmlns="">Ms.</NamePrefix>
3467     </Element>
3468 </Position>
3469 <Position XPath="/Profile/Customer/PersonName/MiddleName">
3470     <Element Operation="delete" />
3471 </Position>
3472 <Position XPath="/Profile/Customer">
3473     <Attribute Name="Gender" Operation="insert" Value="Male" />
3474 </Position>
3475 </OTA_UpdateRQ>
3476

```

3477 **Example 25 - document image after compound update**

```

3478 <Profile>
3479     <Customer Gender="Male">
3480         <PersonName NameType="Default">
3481             <NamePrefix>Mr.</NamePrefix>
3482             <GivenName>George</GivenName>
3483             <Surname>Smith</Surname>
3484         </PersonName>
3485         <TelephoneInfo PhoneUse="Home">
3486             <Telephone PhoneTech="Voice" >
3487                 <AreaCityCode>253</AreaCityCode>
3488                 <PhoneNumber>813-8698</PhoneNumber>
3489             </Telephone>
3490         </TelephoneInfo>
3491         <PaymentForm>
3492             ...
3493         </PaymentForm>
3494         <Address>
3495             <StreetNmbr>1200 Yakima St</StreetNmbr>
3496             <BldgRoom>Suite 800</BldgRoom>
3497             <CityName>Seattle</CityName>
3498             <StateProv PostalCode="98108">WA</StateProv>
3499             <CountryName>USA</CountryName>
3500         </Address>
3501         <RelatedTraveler Relation="Child">
3502             <PersonName>
3503                 <NamePrefix>Ms.</NamePrefix>
3504                 <GivenName>Amy</GivenName>
3505                 <MiddleName>E.</MiddleName>
3506                 <Surname>Smith</Surname>
3507             </PersonName>
3508         </RelatedTraveler>
3509     </Customer>
3510 </Profile>
3511

```

3512 **5.7 Validation of Update Messages**

3513 The update request message is a valid message within its own structure, but since it identifies
3514 only a portion of the content of an XML tree, it cannot be validated in the context of the business
3515 schema. Validation at the level of the business schema must occur after the update has been
3516 completed. Implementors may wish to validate the image of the document before changes have
3517 been applied and again after the changes have been applied in order to ascertain that the desired
3518 result has been obtained and is valid within the business context.

3519 **5.8 The Simple "Replace" verb**

3520 The Replace infrastructure verb defines an operation that updates an existing record by replacing
3521 the existing information with a complete overlay image of the document as it would appear after
3522 changes are applied. The Replace verb does not require a Read request to obtain an image of the

3523 current record prior to sending the message to replace it with the information being transmitted,
3524 although this may be desirable from the standpoint of good business practice.

3525 The "replace" verb allows implementors greater flexibility in choosing how they wish to perform
3526 updates of information, since difference representation may be bypassed and a simple
3527 replacement image of the document object to be updated is sent. This reduces the complexity of
3528 handling updates for some implementations, but at the expense of size of the messages.

3529 **Example 26 - Document update using <Root Operation="replace">**

```
3530 <OTA_UpdateRQ ReqRespVersion="2">
3531   <UniqueId URL="http://vmguys.com/OTAEngine/"
3532     Type="Profile"
3533     Id="12345678"
3534     Instance="7"/>
3535   <Position XPath="/Profile">
3536     <Root Operation="replace">
3537       <Profile>
3538         ...
3539         <!-- include entired updated 'after' image here -->
3540         ...
3541       </Profile>
3542     </Root>
3543   </Position>
3544 </OTA_UpdateRQ>
```

3545 **5.9 OTA_UpdateRS – Responding to a generic OTA_UpdateRQ** 3546 **message**

3547 An OTA update response follows the design pattern for OTA responses and is formally defined
3548 by the following schema fragment:

3549 **Schema 8 - <OTA_UpdateRS>:**

```
3550 <?xml version="1.0" encoding="UTF-8"?>
3551 <!-- Created and edited with 'vi' -->
3552 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3553   xmlns="http://www.opentravel.org/OTA"
3554   targetNamespace="http://www.opentravel.org/OTA"
3555   elementFormDefault="qualified">
3556
3557   <xs:annotation>
3558     <xs:documentation xml:lang="en">
3559     OTA v2001C Specification - Generic OTA_UpdateRS message definition
3560     Copyright (C) 2001 Open Travel Alliance. All rights reserved.
3561     The message id of the message being responded to is in the ebXML header.
3562     </xs:documentation>
3563   </xs:annotation>
3564   <xs:include schemaLocation="OTA_v2ent.xsd"/>
3565
3566   <xs:element name="OTA_UpdateRS">
3567     <xs:complexType>
3568       <xs:choice>
3569         <xs:sequence>
3570           <xs:element ref="Success"/>
3571           <xs:element ref="Warnings" minOccurs="0"/>
3572           <xs:element ref="UniqueId"/>
3573         </xs:sequence>
3574         <xs:element ref="Errors"/>
3575       </xs:choice>
3576       <xs:attributeGroup ref="OTA_PayloadStdAttributes"/>
3577     </xs:complexType>
3578   </xs:element>
3579 </xs:schema>
```

3580

3581 6 Service and Action Mappings

3582 This section defines all the *Action* values which are permitted within each particular *Service*. In
3583 all cases there are RECOMMENDED ‘class of delivery’ semantics (see section 4.3 for definition
3584 of these).

3585 Each defined OTA message is assigned its own *Action* within a specific *Service*. The generic
3586 messages are assigned as a permissible *Action* within each *Service* for which they are applicable.
3587 Working groups are strongly encouraged to make use of the generic messages wherever possible.

3588 Within OTA *Services* many *Actions* are part of request/response pairs. When this is the case the
3589 corresponding expected response or type of request is indicated.

3590 The criteria for grouping related *Actions* into the same *Service* are as follows:

- 3591 • All actions within a service are logically related and are likely to be implemented by the
3592 same application system
- 3593 • Generic messages are defined as actions on each service to which they apply
- 3594 • Request/response pairs always belong to the same service, though in some circumstances
3595 they may belong to more than one service (e.g. generic messages such as OTA_ReadRQ).

3596 6.1 The Session Service

3597 The following table defines the actions available within this service:

<i>Session Action</i>	<i>Class of Delivery</i>	<i>Response Expected</i>	<i>In Response To</i>
CreateRQ	Type-A	CreateRS	-
CreateRS	Type-A	-	CreateRQ
CloseRQ	Type-A	CloseRS	-
CloseRS	Type-A	-	CloseRQ

3598

3599 During Session creation parties MUST establish which services and actions are supported via a
3600 <ServicesSupported> element on the <SessionControlResponse> message. This
3601 process is outlined in the following section.

3602 6.2 The Profile Service

3603 The following table defines the actions available within this service:

<i>Profile Action</i>	<i>Class of Delivery</i>	<i>Response Expected</i>	<i>In Response To</i>
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

3604 **6.3 The VehicleBooking Service**

3605 The following table defines the actions available within this service:

<i>VehicleBooking Action</i>	<i>Class of Delivery</i>	<i>Response Expected</i>	<i>In Response To</i>
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

3606 **6.4 The AirBooking Service**

3607 The following table defines the actions available within this service:

<i>AirBooking Action</i>	<i>Class of Delivery</i>	<i>Response Expected</i>	<i>In Response To</i>
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_MultipleAirlineAvailRQ
OTA_AirBookRQ	Type-A	OTA_AirBookRS	-
OTA_AirBookRS	Type-A	-	OTA_AirBookRQ
OTA_CancelRQ	Type-A	OTA_CancelRS	-
OTA_CancelRS	Type-A	-	OTA_CancelRQ

3608 **6.5 The TravellInsurance Service**

3609 The following table defines the actions available within this service:

<i>TravellInsurance Action</i>	<i>Class of Delivery</i>	<i>Response Expected</i>	<i>In Response To</i>
OTA_InsuranceQuoteRQ	Type-A	OTA_InsuranceQuoteRS	-
OTA_InsuranceQuoteRS	Type-A	-	OTA_InsuranceQuoteRQ
OTA_InsuranceBookRQ	Type-A	OTA_InsuranceBookRS	-

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

3610 **6.6 The HotelBooking Service**

3611 The following table defines the actions available within this service:

<i>HotelBooking Action</i>	<i>Class of Delivery</i>	<i>Response Expected</i>	<i>In Response To</i>
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

3612 **6.7 The HotelResNotification Service³²**

3613 The following table defines the actions available within this service:

<i>HotelResNotification Action</i>	<i>Class of Delivery</i>	<i>Response Expected</i>	<i>In Response To</i>
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

3614 **6.8 The HotelPropertyInformation Service³³**

3615 The following table defines the actions available within this service:

<i>HotelPropertyInformation Action</i>	<i>Class of Delivery</i>	<i>Response Expected</i>	<i>In Response To</i>
OTA_CommNotifRQ	Type-A	OTA_CommNotifRS	-
OTA_CommNotifRS	Type-A	-	OTA_CommNotifRQ

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

³³ Some of the messages on this service are ideal candidates for a single type-B notification message rather than request/response pairs.

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

3616 **6.9 The MeetingProfile Service**

3617 The following table defines the actions available within this service:

<i>MeetingProfile Action</i>	<i>Class of Delivery</i>	<i>Response Expected</i>	<i>In Response To</i>
OTA_CreateMeetingProfileRQ	Type-A	OTA_CreateMeetingProfileRS	-
OTA_CreateMeetingProfileRS	Type-A	-	OTA_CreateMeetingProfileRQ

3618 **6.10 The PackageBooking Service**

3619 The following table defines the actions available within this service:

<i>PackageBooking Action</i>	<i>Class of Delivery</i>	<i>Response Expected</i>	<i>In Response To</i>
OTA_PkgAvailRQ	Type-A	OTA_PkgAvailRS	-
OTA_PkgAvailRS	Type-A	-	OTA_PkgAvailRQ
OTA_PkgBookRQ	Type-A	OTA_PkgBookRS	-
OTA_PkgBookRS	Type-A	-	OTA_PkgBookRQ

3620 **6.11 The GolfTeeTimes Service**

3621 The following table defines the actions available within this service:

<i>Session Action</i>	<i>Class of Delivery</i>	<i>Response Expected</i>	<i>In Response To</i>
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
-----------------	--------	---	-----------------

3622

3623 **6.12 Determining Services Supported**

3624 A *CreateRS* action on the *Session* service **MUST** be accompanied by a `<ServicesSupported>`
 3625 within a `<SessionControlResponse>` document located in the payload container of the message.
 3626 Upon receipt of *Session CreateRQ* request an implementation responds with a *CreateRS* action
 3627 message with a status attribute `status="Accepted"`. Here is an example of a
 3628 `<SessionControlResponse>` with a `<ServicesSupported>` element:

3629 **Example 27 - sample `<SessionControlResponse>` payload:**

```

3630 <?xml version="1.0" encoding="UTF-8"?>
3631
3632 <SessionControlResponse xmlns="http://www.opentravel.org/OTA"
3633   xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
3634   xsi:schemaLocation="http://www.opentravel.org/OTA SessionControlResponse.xsd"
3635   Version="1" status="Approved">
3636   <OTA_Version>2001C</OTA_Version>
3637   <ConversationId>20011027081907-862@host.imacompany.com</ConversationId>
3638
3639   <ServicesSupported>
3640     <Service Type="OTA" Name="HotelBooking">
3641       <Action Name="OTA_HotelSearchRQ" Version="2"/>
3642       <Action Name="OTA_HotelAvailRQ" Version="2"/>
3643       <Action Name="OTA_HotelResRQ" Version="2">
3644         <Extension Name="VMStayAmenities" Version="1" Required="no" />
3645       </Action>
3646     </Service>
3647
3648     <Service Type="OTA" Name="Profile">
3649       <Action Name="OTA_CreateProfileRQ" Version="2">
3650         <Extension Name="DeltaSkymiles" Version="1" Required="no" />
3651       </Action>
3652       <Action Name="OTA_ReadRQ" Version="2"/>
3653       <Action Name="OTA_UpdateRQ" Version="2"/>
3654     </Service>
3655
3656     <Service Type="OTA" Name="Session">
3657       <Action Name="CreateRQ" Version="1" />
3658       <Action Name="CloseRQ" Version="1" />
3659     </Service>
3660   </ServicesSupported>
3661 </SessionControlResponse>
  
```

3662

3663 In this example, an implementation supports two services (*HotelBooking* and *Profile*) in addition
 3664 to the *Session* service itself.

3665 **6.12.1 The `<ServicesSupported>` element**

3666 The *ServicesSupported* element has the following attributes

- 3667 • *Version* : The version of this message. Currently a value of 1.

3668 The *ServicesSupported* element **MUST** contain a sequence of at least one *Service* element. N.B.
 3669 *Services* and *Actions* included in a *ServicesSupported* message are always from the perspective of
 3670 the system sending the message i.e. the message formally defines which services the
 3671 implementation provides and which actions upon each service it implements and will accept
 3672 messages for. An implementation specifies the request actions it will accept, but does not need to

3673 specify the response actions it will generate (these are implied based upon the Service definition
3674 tables earlier in this section).

3675 **6.12.2 The <Service> element**

3676 The *Service* element has the following required attributes:

- 3677 • *Type*: A value of “OTA”. This is the value that **MUST** be used in the *type* attribute of the
3678 <eb:Service> element on the <eb:MessageHeader>
- 3679 • *Name*: The service name. This is the value which **MUST** be used as the content of the
3680 <eb:Service> element on the <eb:MessageHeader>

3681 The *Service* element **MUST** contain a sequence of at least one *Action* element.

3682 **6.12.3 The <Action> element**

3683 The *Action* element has the following required attributes:

- 3684 • *Name*: The action name. This is the value that **MUST** be used as the content of the
3685 <eb:Action> element on the <eb:MessageHeader>. This name also usually corresponds
3686 directly to one of the defined OTA message types (except in the case of the *Session*
3687 service)
- 3688 • *Version*: The version of the OTA message supported. Implementations that support more
3689 than one version of a given message can indicate this by including additional <Action>
3690 elements with the same *Name* but different values for *Version*

3691 The *Action* element **MAY** contain a sequence of *Extension* elements.

3692 **6.12.4 The <Extension> element**

3693 An Extension indicates a bilaterally agreed upon message extension between two trading
3694 partners, using the <TPA_Extension> mechanism. The *Extension* element has the following
3695 attributes:

- 3696 • *Name*: The name of an extension (required)
- 3697 • *Version*: the version of that extension (required)
- 3698 • *Required*: a yes/no value indicating whether the extension is mandatory (defaults to ‘no’)

3699 **6.12.5 The ServicesSupported Schema**

3700 The following schema fragment formally defines the <ServicesSupported> element:

3701 **Schema 9- ServicesSupported.xsd**

```
3702 <?xml version="1.0" encoding="UTF-8"?>
3703
3704 <!-- Created and edited with 'vi' -->
3705
3706 <xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3707           xmlns="http://www.opentravel.org/OTA"
3708           targetNamespace="http://www.opentravel.org/OTA"
3709           elementFormDefault="qualified">
3710
3711   <xs:annotation>
3712     <xs:documentation xml:lang="en">
3713     OTA v2001C Specification - ServicesSupported message definition
```

```

3714 Copyright (C) 2001 Open Travel Alliance. All rights reserved.
3715     </xs:documentation>
3716 </xs:annotation>
3717
3718     <xs:simpleType name="versionNumber">
3719       <xs:annotation>
3720         <xs:documentation xml:lang="en">
3721 All OTA version numbers are unsigned integers in the range 1..9999.
3722 Version numbers should begin with the value '1' and be incremented
3723 each time a message is revised.
3724         </xs:documentation>
3725       </xs:annotation>
3726
3727       <xs:restriction base="xs:unsignedShort">
3728         <xs:minInclusive value="1"/>
3729         <xs:maxInclusive value="9999"/>
3730       </xs:restriction>
3731     </xs:simpleType>
3732
3733     <xs:element name="ServicesSupported">
3734       <xs:annotation>
3735         <xs:documentation xml:lang="en">
3736 Generated and sent during OTA session negotiation to indicate the services
3737 a given implementation supports.
3738         </xs:documentation>
3739       </xs:annotation>
3740
3741       <xs:complexType>
3742         <xs:sequence>
3743           <xs:element ref="Service" minOccurs="1" maxOccurs="unbounded" />
3744         </xs:sequence>
3745         <xs:attribute name="Version" type="versionNumber" />
3746       </xs:complexType>
3747     </xs:element>
3748
3749     <xs:element name="Service">
3750       <xs:annotation>
3751         <xs:documentation xml:lang="en">
3752 Indicates a defined OTA Service which this implementation supports,
3753 providing one or more of the standard actions defined on that service.
3754         </xs:documentation>
3755       </xs:annotation>
3756
3757       <xs:complexType>
3758         <xs:sequence>
3759           <xs:element ref="Action" minOccurs="1" maxOccurs="unbounded" />
3760         </xs:sequence>
3761         <xs:attribute name="Type" use="required">
3762           <xs:simpleType>
3763             <xs:restriction base="xs:string">
3764               <xs:enumeration value="OTA"/>
3765             </xs:restriction>
3766           </xs:simpleType>
3767         </xs:attribute>
3768         <xs:attribute name="Name" type="xs:string" use="required" />
3769       </xs:complexType>
3770     </xs:element>
3771
3772     <xs:element name="Action">
3773       <xs:annotation>
3774         <xs:documentation xml:lang="en">
3775 Indicates an Action upon a defined OTA Service which this implementation
3776 supports. For most OTA Services the Action name is equivalent to the action
3777 verb and root tag of the primary payload document.
3778         </xs:documentation>
3779       </xs:annotation>
3780
3781       <xs:complexType>
3782         <xs:sequence>
3783           <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded" />
3784         </xs:sequence>

```

```

3785         <xs:attribute name="Name" type="xs:string" use="required" />
3786         <xs:attribute name="Version" type="versionNumber" use="required" />
3787     </xs:complexType>
3788 </xs:element>
3789
3790 <xs:element name="Extension">
3791     <xs:annotation>
3792         <xs:documentation xml:lang="en">
3793 Indicates an extension point within a standard OTA action that this
3794 implementation understands.
3795         </xs:documentation>
3796     </xs:annotation>
3797
3798     <xs:complexType>
3799         <xs:attribute name="Name" type="xs:string" use="required" />
3800         <xs:attribute name="Version" type="versionNumber" use="required" />
3801         <xs:attribute name="Required" default="no">
3802             <xs:simpleType>
3803                 <xs:restriction base="xs:string">
3804                     <xs:enumeration value="yes"/>
3805                     <xs:enumeration value="no"/>
3806                 </xs:restriction>
3807             </xs:simpleType>
3808         </xs:attribute>
3809     </xs:complexType>
3810 </xs:element>
3811 </xs:schema>
3812
3813

```

3814 **6.13 Sample Session Message Flow**

3815 The example below shows the messages exchanged during a sample session, including session
3816 initiation and termination.

3817 **Example 28 - sample session**

3818 Comments:	System A	System B
3819 =====	=====	=====
3820 A initiates SSL connection with B	HTTP POST with Authorization	
3821 Containing Username/password data		
3822 While creating a new session	Session::CreatorRQ	
3823 A initiates a new session with B	----->	
3824	Session::CreatorRS status=Accepted	
3825 ...B accepts the session	<-----	
3826	AirBooking::OTA_MultiAirAvailRQ	
3827	----->	
3828 A checks air availability	AirBooking::OTA_MultiAirAvailRS	
3829 ...B responds with availability	<-----	
3830	AirBooking::OTA_AirBookRQ	
3831 A makes an air booking request	----->	
3832	AirBooking::OTA_AirBookRS	
3833 ...B responds	<-----	
3834	AirBooking::OTA_MultiAirAvailRQ	
3835	----->	
3836 A makes a different avail. request	AirBooking::OTA_MultiAirAvailRS	
3837 ...B responds	<-----	
3838	AirBooking::OTA_CancelRQ	
3839 A initiates cancellation of a booking	----->	
3840	AirBooking::OTA_CancelRS	
3841 ...B confirms cancellation	<-----	
3842	AirBooking::OTA_UpdateRQ	
3843 A modifies details on a booking	----->	
3844		
3845		
3846		
3847		
3848		



3859 **7 Summary of Infrastructure Changes**

3860 This section is informative and provides a description of the changes in infrastructure from
3861 previously published OTA standards.

- 3862 • Movement from published DTDs to published XML schemas for specification of message
3863 syntax and the underlying reference model
- 3864 • A mapping to ebXML 1.0 Transport, Routing and Packaging as a RECOMMENDED
3865 infrastructure substrate for OTA implementations
- 3866 • Elimination of the previously mandatory <Control> payload in favor of equivalent
3867 capabilities provided by underlying infrastructure
- 3868 • Definition of the Service/Action concept and mapping of each defined OTA message as an
3869 <Action> on at least one <Service>
- 3870 • Concrete definition of the notion of OTA sessions and the definition of a Session
3871 <Service> which controls session setup and termination
- 3872 • Elimination of the previous non-versioned VersionDiscovery mechanism in favor of
3873 <ServicesSupported> negotiation during session establishment
- 3874 • Elimination of the <Sendby> semantics allowed for in the previous <Control> section.
3875 OTA STRONGLY RECOMMENDS all message exchanges occur within the context of a
3876 valid session
- 3877 • Support for <ReplyTo> <CCTo> and the OrigBodyReq attribute has been dropped from
3878 this specification. These elements, or a mechanism providing similar functionality will be
3879 considered during a future specification when a valid use-case dictates (this kind of
3880 functionality may be provided by a publish-subscribe messaging model)