OpenTravel Alliance version 2001C Specifications

Working Draft – for Architecture Committee Review

October 29, 2001

(not intended for public distribution)

Table of Contents

123

1 Introduction

1.1 About the OpenTravel Alliance[1]

Created in May of 1999, the OpenTravel Alliance (OTA) is a legally incorporated, non-profit consortium consisting of approximately 150 member companies from all sectors of the travel industry. These sectors include typical travel industry supplier companies from the airline, car rental, hotel and leisure travel verticals, in addition to related companies that provide distribution and technology to support the industry.

The OTA consists of the five following working groups:

- Air Working Group;
- Car Working Group;
- Hotel Working Group;
- Leisure Working Group, and
- Non-Supplier Working Group

Each of the industry travel supplier segments has its own business work group, tasked with the development of technical specifications for its particular vertical, while the non-supplier group works with each of the supplier working groups according to the technology services that their company provides.

OTA's work efforts over the past two years have led to the successful organization of the work groups and, more importantly, immediate development of specifications for customer and company profiles, and the all-important functions of requesting the availability of travel products and obtaining those products by booking a reservation.

An Interoperability Committee reconciles the specifications from all industry segments to provide a consistent approach to both customer and content throughout the travel business. The Interoperability Committee is comprised of elected members from each of the five industry work groups, and is chartered to serve as the technical steering group for OTA.

From time to time, the Interoperability Committee commissions cross-industry teams to address the needs of all the working groups. Over the past two years, a Profile Content team was created to develop the original OTA Version 1 Customer Profile, and the Profile Integration team worked to integrate the HITIS and OTA profiles.

Currently, two ongoing teams are in existence:

- Architecture team;
- Data Content team;

In the creation of these cross-industry teams, OTA realized that there was considerable work to do involving infrastructures and specific vocabularies within each travel industry segment.

1.1.1 Use of XML

The OTA has chosen Extensible Markup Language (XML) as the vehicle for its specifications, determining that XML is the best means for meeting its goals. XML offers a common framework

[1] Thanks go to David Blackistone and Diane Mair, KPMG, for their contributions to this section.

for creating Internet-compatible messages, but is flexible enough to be used between systems elsewhere in the travel industry.

The use of XML will assist in producing industry-wide specifications that will exploit the opportunities available with the advent of universal access to the low-cost, fast, communications infrastructure that has arrived with the Internet. The successful implementation of these new specifications will result in the facilitation of customer and trip-related information amongst all industry trading partners, regardless of how "net savvy" the participant.

The following are several aspects of XML that made it the preferred language in designing a specification for OTA's needs:

- XML allows for the exchange of structured data over the web.
- XML messages can include action elements that instruct receiving systems to undertake certain operations.
- XML allows for the definition of data elements meaningful to trading partners or entire industries.
- XML is a powerful tool for reshaping the way information is transmitted and presented over the Internet.

The result of OTA's stated goals is specifications created on an open platform that will allow trading partners to access standardized information.

1.1.2 Status of the Travel Industry

The travel industry has been a leader in using global electronic distribution of information for decades, but it has developed this technology in a manner that has resulted in several incompatible standards between the different industry members and their lines of business. What has evolved is a series of related, sometimes interconnected, segments trying to provide the user with as much information as possible. Incompatible specifications have made it difficult for various participants in the travel industry to share information and has limited the variety of distribution channels available to travel suppliers. These limitations make it difficult for suppliers to differentiate themselves, reach new customers, create innovative offers or exchange information with their corporate customers.

Customers are also restricted by this situation and are not able to make a completely informed choice in comparing services from different sources. Currently, customers must navigate a complex web of interrelationships to obtain all the information they might desire. Most travelers elect not to do this.

Information is the core of any travel transaction, be it with the airlines, car rental companies, cruise lines, railroads, hotels, or tour operators. The absence of a common standard has fostered distributions through single channels that silo both customer and market data. While the industry has tried to exploit the Internet, it cannot fully do so until a universal standard has been established to exchange information.

1.1.3 Design Goals of OTA

OTA's basic goal is to design industry specifications capable of exploiting the communications systems that are available with Internet connectivity. To achieve these goals, OTA has designed specifications to meet the following criteria:

- *Openness*- OTA specifications are publicly available to all organizations seeking to develop new or enhanced systems. Membership to OTA is open to all organizations interested in developing these specifications.
- *Flexibility*- OTA specifications provides travel service providers with the flexibility they need to develop, test and deploy new services. The specifications outline the minimum necessary functionality to provide reliable interactions between customers and the systems owned and maintained by the companies serving them.
- *Platform Independence*- OTA has developed this specification to work with any equipment or software that can support the common standards used in the specification.
- *Security*- OTA places great importance on the need to protect information from unauthorized access and the need to give the customer control over the creation, update and exchange of data with other parties.
- *Extensibility*- OTA plans to add more services and functionality to this specification in a way that minimizes incompatibility for those implementing this or other early versions. OTA work groups that develop the specifications do so with future transitions in mind.
- *International scope*- The initial specification was written in English; however, OTA intends to extend later versions to provide representation in character sets supporting the Unicode standard. When possible, OTA has designed data elements to meet as many global elements as possible.

1.1.4 OTA's Approach

Industry fragmentation has led to little terminology and technology being universal throughout the travel industry. The specifications developed by OTA remove the constraints imposed by the trade-offs required when computer communications and processing were costly. These specifications represent a different approach to exchanging data elements and messages, and intend to provide each data element with a unique identity or tag within each industry sub-segment.

Historically, developing standards can be characterized by being long in conception, inflexible and retrospective. OTA recognizes these concerns and is proactively trying to address these concerns through rapid development and implementation. It is OTA' s belief that the underlying technology has been designed to accommodate changes that will occur as the travel industry evolves, and it is OTA’s hope that worldwide availability of OTA’s specifications will spur development of new interchanges between partners in the travel industry.

OTA uses the term *specification* rather than *standard*. OTA reserves the word standard for documents that undergo a rigorous, lengthy development and review process, such as those from International Organization of Standards (French acronym ISO) or American National Standards Institute (ANSI). This document, therefore, is called a specification, which takes broad standards such as the Extensible Markup Language (XML) and applies them to particular problems. As a specification, OTA documents focus on travel industry requirements and while developed in an open consensus process, does not represent itself as meeting the stringent requirements of standards.

1.1.5 Specification Components

OTA’s publications consist of several components that should enable companies to take advantage of current tools and methods for implementation.

A full-fledged set of OTA specifications includes:

- *Specification Document*- MS Word or .pdf file that contains the traditional specification information; along with introduction and explanations of the business content.
- *XML Schema and Schema Fragments* – The XML Schema in conformance with the World-Wide Web Consortium (W3C) Recommendation. Schema fragments define the individual modular pieces of a specification for reusability, and the schema at the message level defines the entire specification for that message. *
- *UML Model* – A graphical model of the data relationships, elements and attributes and their descriptions in Rational Rose format.
- *Data Dictionary* – An alphabetical list of the elements and attributes defined in the schemas, showing their parent class, cardinality and relationships to other elements.
- *Appendix* - Revisions Transmittal and Change Log, references to other standards used, DTDs and Schemas. [DTDs and XML schemas available in electronic format]

* While the text of the Specification document makes every attempt to accurately reflect the DTDs and XML Schema listed in the Appendices. In the case of a conflict between the document text, the DTD and the XML Schema, the XML Schema takes precedence.

1.1.6 Future goals for OTA

The ever-evolving travel industry will require OTA to provide for the continued growth in the capabilities and functions for future specifications. OTA's goals are to provide specifications specifically written to address the needs of the industry sub-segments in a manner that is unique within the OTA name-space. Future publications and specifications will focus on the vocabulary and the messaging needs of each industry segment designed by the industry work groups.

The HITIS Integration portion of this specification, once it has gone through its OTA's review process and is recommended for adoption and broad implementation by the travel industry, may be submitted as an ANSI standard as a result of an Agreement with the American Hotel & Motel Association (AH&MA), signed October 18, 2000. This agreement allows the AH&MA, an ANSI-accredited standards developer organization, to submit a specification for approval as an ANSI standard. That process, monitored by the American National Standards Institute, involves the publication of an intended standard for a period of commentary and consists of a canvass of those parties who would be materially affected by the standard.

1.2 OTA Versioning

To provide for both implementation stability and managed growth, the OTA has adopted a scheme for versioning of messages. Infrastructure changes, such as the addition of Header elements or changes in the method used to connect to trading partners and initiate authentication or security-related functions will constitute an Infrastructure version change. For Version 2, OTA assumes the service interface as defined in Chapter 4, Message Structure, of the OTA 2001A specification. That infrastructure is anticipated to change, potentially adopting the Messaging Specification, Version 1.0, of the ebXML Transport, Routing and Packaging working group.

OTA payload messages are intended to communicate the business functionality. Changes to the schema (and retroactively, to the DTD for that schema), or the addition of business content elements to the schema constitute a change in the version of the payload message.

A payload document is identified in the Document Reference of the Manifest element (Version 2 OTA Infrastructure). The Version attribute in the Schema element indicates the version of

message in the payload. The identification of metadata about the version at this level allows a trading partner to immediately identify whether its system can accept the document(s) and process them appropriately, eliminating the need to parse the entire document in order to return an error message about version compatibility.

In order to isolate and contain the scope of change with the addition of new business content and messages, when OTA releases a specification, each new message introduced will be versioned starting with the number 1, and incremented by N+1 with each subsequent revision. This places the level of versioning at the message interaction, or at the level of the action verb that is the root of the payload document.

OTA will identify publications by the year of publication, and sequential publications within a calendar year will be further identified by the letter A, B, C, or D, etc. For example, the first specification published in the year 2001 was 2001A, the second publication 2001B, etc.

- The OTA 2001A publication included OTA Infrastructure, Version 2, and the integrated HITIS / OTA Customer Profile, Version 2.
- The OTA 2001B publication included Version 1 messages for all working groups represented.
- The OTA 2001C publication contains major revisions to infrastructure and best practices along with the publication of additional messages sets.

Each OTA publication will indicate which version of infrastructure is supported, (or publish a specification if the publication includes a change in the infrastructure) and list all messages supported by the publication, with version numbers. This list may include existing messages, new messages, and revisions of existing messages.

With each publication, an updated list of valid enumerations supported for the *Type* attribute (strings that match the root tag of the payload and identify the messages) will be included.

1.3 Versions Supported

The Version numbers of the messages supported by this publication[2] are the following:

1.3.1 Infrastructure Version:

The Version 2 OTA Infrastructure is the current version of infrastructure used for the exchange of OTA messages. OTA Infrastructure, Version 2, is documented in publication 2001A.

As of the date of this publication, OTA payload messages in this document are presumed to use OTA Version 2 Infrastructure.

Note: *A change in infrastructure is anticipated based upon the completion of the ebXML Messaging Specification in early May, 2001.*

1.3.2 UniqueId Types Supported:

Two string values are supported as an enumeration of the *Type*= attribute of the IniqueId.

- *Profile* - Identifies a customer or company profile by UniqueId
- *Reservation* - Indentifies a reservation by UniqueId

[2] This list includes existing messages, new messages, and revisions of existing messages.

1.3.2.1 Version of Payload Messages:

1.3.2.1.1 Existing messages:

Message	*Version*	*Comments*
OTA_CreateProfileRQ	2	
OTA_CreateProfileRS	2	
OTA_ReadRQ	2	Generic infrastructure verb
OTA_ReadProfileRS	2	
OTA_UpdateRQ	2	Generic infrastructure verb
OTA_UpdateRS	2	Generic infrastructure verb
OTA_DeleteRQ	2	Generic infrastructure verb
OTA_DeleteRS	2	Generic infrastructure verb
OTA_CancelRQ	2	Generic infrastructure verb (revised)
OTA_CancelRS	2	Generic infrastructure verb (revised)

**FIXME – revise to include final list of 2001B messages

1.3.2.1.2 New messages:

Message	*Version*	*Comments*
CreateRQ	1	Part of the new session service
CreateRS	1	Part of the new session service
CloseRQ	1	Part of the new session service
CloseRS	1	Part of the new session service

**FIXME – update to include other new published messages (e.g. Golf related).

1.4 The Journey Concept - 2001 and Beyond

The OpenTravel Alliance (OTA) seeks to make better use of Internet technology to provide the travel industry and travelers with a greater array of communications channels and services, and in turn, provide a greater level of service to travelers. OTA's Specifications make important contributions toward this objective by offering specifications for the travel industry on travel planning, availability, package tours, reservations, and fulfillment of services.

As part of these efforts, OTA also seeks to broaden the concept of travel services, to meet the needs of travelers that extend beyond the industry's traditional definition of the term. To reach this goal, OTA has identified the traveler's *journey* as a basic idea upon which to organize further development of its message and service specifications.

1.4.1 Scope of these specifications

OTA has specified a set of business processes for planning travel through the fulfillment and analysis of the travel experience, all of which play a vital role in one's journey. During the planning phase, the functions involve a choice of destination (if one has that choice), with the meeting of personal interests or business objectives, such climate, activities, or recreational services. This planning function can involve searching by name, finding airports or railway stations in the proximity of a geographic location or the final destination, etc., as well as include making a choice among lodging options.

Planning a trip is different from buying other goods or services because of the interrelationship between the individual items in the complete package. Items can be related *commercially* (packages, car rental/flight combinations, etc.) and/ or by *time* and *place*. Unifying this relationship is the fact that they all belong to an individual's trip plan.[3]

The journey also involves taking action on those plans, beginning with determining availability of the services required, covering transportation, lodging, car rental, and travel insurance if required. A distinct feature of the travel business is that one can purchase (traditionally as well as over the Internet) from horizontally as well as vertically oriented businesses. If the traveler is fortunate enough to be planning a trip for a holiday, he/she can take advantage of the planning having largely been done for them in a tour package. Business travelers, on the other hand, may address their needs independently, or with the assistance of travel planning software or the services of a travel agent.

A traveler's journey begins in the planning stages by determining the boundaries of time and location within which the travel needs to be fulfilled. These requirements form the foundation of a Journey structure. To add specific information, one would need to fill in a "*segment*" of the journey, which provides a framework for storing individual bits of trip-related data throughout all phases of the trip planning.

Segment, in its broadest definition, includes a portion of the journey, such as the requirement to have a flight from Point A to Point B, but does not necessarily have to be filled in until a selection has been made for the service that will fulfill the need. A Segment defines a piece of the journey that may include the flight, hotel stay, insurance, kennel stay, etc. A segment would have any other information that would apply to the segment in general, such as a start date or end date. Each segment may be in a different stage of completeness with additional segments being added for return flights, hotel stays etc., as needed. At some point, the journey document, made up of individual segments, can be sent to travel suppliers for a quotation request.[4]

Travel providers are then free to respond to a request. If the request has been directed singularly to a supplier, that supplier (e.g.: an airline) may offer the best deal on a flight, while a travel agent may put together a package that includes hotel and car rental within the constraints of the initial journey.

A "*service*" is the actual product or service that the customer eventually purchases, and it may be desirable to have more than one service in any given segment. For example, only after receiving a first quotation it may become apparent that it is more economical to pay for one or two extra hotel nights in order to get a less expensive flight that stays over the weekend, etc. If the preferred vendor cannot fulfill the request, e.g.: the hotel is not available for the longer length of stay, than an alternative needs to be considered within the same segment.

---

[3] We thank Vincent Buller, formerly with AND Data Solutions, for his contributions to this treatise on the Journey Concept.

[4] We thank Vic Roberts, OAG, for his contributions to the Journey Concept.

The relationships between travel services make trip planning very much an interactive process. Determining availability means considering target dates and pricing options or limitations. While events in a trip may occur in chronological order, several events could be running at the same time. For example, a car rental event can be running simultaneously with a hotel stay. The time relationship does communicate important information, however, such as the customer requires a parking space that may need to be reserved along with the hotel reservation.

The responses of the travel providers will assist in identifying the specific services that will ultimately arrive at the point at which the traveler can make the booking request. The commercial relationships link together the items that were previously defined only as time and geographic location, but now attaches essential trip information such as reservation numbers, electronic tickets, vouchers, etc. After all the travel items are finalized, a travel insurance broker could be given a look at the journey information in order to quote for insurance covering the complete trip.

Travel, like any human activity, however, does not always take a direct path from plans through fulfillment. The traveler may wish to book additional facilities, such as reserve a table in a restaurant near the hotel, or plan an extra step into the journey. Changes in flight arrangements, such as a delay or cancellation, requires an update of hotel and car rental information, and other portions of the trip that could be affected as well. Because the Journey concept is valid in all travel phases, not just the booking and reservation phase, it is a concept that fits the interactive nature of travel.

When the traveler faces the need to make changes in itineraries, this means changes in dates, times, destinations, services, and pricing. The traveler's changes affect not only the travel services directly, but the changes also need to be recorded and synchronized with all of the travel supplier's reservations and information services. For example, a cancellation of a room reservation adds to a property's inventory that needs to be communicated quickly and accurately to the parent company's central reservation service and made available to their web site, as well as those run by the individual property.

Travel services need to record in real time the traveler's engagement of travel services throughout the journey. The documentation of these services becomes part of the travel service's record, either for historical purposes or as part of an audit trail. Companies in the travel industry, like companies anywhere, need to get paid for the services they provide, and the specifications include the identification of payment methods and indicators for the financial settlement. OTA will rely on existing vocabularies in the financial services industry for the semantics and mechanics of financial settlements.

1.4.2 The Programmer's View of the Journey

OTA intends to provide a comprehensive set of journey-oriented, loosely-coupled travel services that will enable nontraditional as well as traditional business services. Because the Journey Concept is valid in all travel phases, it is a concept that fits the iterative process of trip planning.

Currently, the OTA specifications contain independent messages from each industry vertical developed by supplier working groups. To the extent possible, the OTA has compared similar data elements across industry verticals so that the same data tag will be expressed similarly where it is encountered in another set of specifications. However, it is inevitable that specifications developed independently from one another will require a cross-industry comparison of the structure of the messages as well as a process of data normalization that recognizes reusability and provides a mapping between them.[5]

[5] This task may be an ongoing role for the OTA Data Content team.

The current OTA infrastructure is designed to handle multiple related documents by identifying those documents in the Manifest and referencing them in the reply.

Atomic messages have been retained for both the request and reply. Each message is identified in the Document Reference element of the Manifest based upon the string that is the root tag of the document. Although the EchoToken is an attribute on the root element that can be used to tie the response back to the request, the EchoToken in the OTA_v2ent.mod file is optional. Since it is not required, it should not be relied upon to be used or to identify the message in all cases.

Ultimately, it may be possible to develop a structure for a combined message that may include all supplier messages in one payload document. However, complications may arise from a message combining documents intended for separate suppliers as the documents can be encrypted differently depending upon agreements between parties. Multiple documents have been deemed to allow for more flexibility, and require less processing, as a multi-part document must be parsed in order to split out a message and send it to different servers.

1.4.3 The Traveler's View of the Journey - 2001 and Beyond

OTA 's work on the Profile specification defines general traveler preferences such as smoking/ non-smoking, company travel arrangements and fares for business trips, etc. These preferences can be extracted using the journey concept to apply a place and time relationship to describe traveler preferences for a particular trip.

For many customers, the journey encompasses needed services that go well beyond the traditional travel services of airlines, passenger rail, lodging, or car rentals. While these basic services are indeed important to getting travelers safely to their destinations and providing for a reasonable level of comfort, many travelers will need other kinds of assistance in order for their travels to be successful. Here are a few examples:

- Business travelers conducting meetings in distant cities may have literature to distribute to their associates. They could carry the materials with their hand baggage on the airplane or check the box as accompanied baggage. However, many of these business travelers would also pay for a reliable package delivery service that could relieve them (and the air carrier) of an extra piece of baggage and deliver the package to their destination. Why can't business travelers arrange for this service while planning their travel?
- Many travelers headed to downtown locations in large cities rarely need a car. For these travelers who need to get from the airport to downtown, however, the only chance they have to make a decision on ground transportation (shared van, airport bus) is when they get to airport and retrieve their luggage. Why can't travelers make arrangements for ground transportation, perhaps other than taxis, when making the air or hotel arrangements?
- Finding a suitable restaurant is a continuous need of travelers. While some hotels provide concierge services to help travelers with meal reservations, sometimes being able to make those arrangements in advance can help a traveler's business as well as culinary interests. While a few restaurant reservation services have begun appearing on the Web, why can't they be integrated with traditional travel services?
- When travelers have pets, they can leave them with friends or family members, but when these options are not available, what are the choices? Finding a suitable boarding kennel that can take a pets on short notice is a definite need of these travelers. Why not have this service as an option for these travelers?

The examples of the journey concept given above are not exhaustive, but they show the kinds of services that travelers consider part of their travel experiences, yet are rarely offered by

companies in the travel industry. Including these new kinds of services along with the traditional travel services requires communication methods and protocols that cover the traditional travel services, yet can still extend in a consistent and reliable way to include these non-traditional offerings.

To fulfill the promise of the journey concept will mean bringing in many companies not yet involved in the travel industry, which presents a whole new set of challenges. Many of the companies offering these new kinds of services are smaller in size and do not have the extensive resources needed to develop sophisticated information systems. As a result, any framework adopted by OTA to incorporate these services needs to be designed for use by companies of all sizes. Any specifications of this kind also must be available worldwide, since the travel industry and the needs of travelers extend worldwide.[6]

1.4.4 Meeting the journey's data exchange requirements

The Electronic Business XML or ebXML initiative offers a new worldwide specification for electronic business messages and services designed for businesses ranging from the smallest operation to multi-national conglomerates. EbXML, a joint enterprise of the Organization for the Advancement of Structured Information Standards (OASIS) and the United Nation's trade facilitation agency (UN/CEFACT) completed its technical specifications in May 2001.

EbXML is a set of specifications that together enables a modular, yet complete electronic business framework. The ebXML initiative is designed for electronic interoperability, allowing businesses to find each other, agree to become trading partners and conduct business. All of these operations can be performed automatically, minimizing, and in most cases completely eliminating the need for human intervention. This streamlines electronic business through a low cost, open, standard mechanism.

EbXML is designed to meet these needs and is built on three basic concepts: provide an infrastructure that ensures data communication interoperability; provide a semantic framework that ensures commercial interoperability. EbXML also provides a mechanism that allows enterprises to find each other, agree to become trading partners and conduct business with each other.

1. The infrastructure to ensure data communication interoperability is provided through:

- a standard message transport mechanism with a well defined interface, packaging rules, and a predictable delivery and security model
- a 'business service interface' that handles incoming and outgoing messages at either end of the transport

The semantic framework to ensure commercial interoperability is provided through:

- a metamodel for defining business process and information models
- a set of re-useable business logic based on core components that reflect common business processes and XML vocabularies
- a process for defining actual message structures and definitions as they relate to the activities in the Business Process model.

3. The mechanism to allow enterprises to find each other, agree to establish business relationships, and conduct business, is provided through:

[6] Many thanks to Alan Kotok, former OTA Standards Manager, for providing this vision of the future.

- shared repository where enterprises can register and discover each other's business services via partner profile information
- process for defining and agreeing to a formal Collaboration Protocol Agreement (CPA), if so desired or where required.
- shared repository for company profiles, business process models and related message structures [7]

OTA is an early and long-time supporter of ebXML. Representatives from OTA took part in most of the ebXML development meetings, beginning with its organizational session in November 1999. OTA also contributed the business content of its first proof of concept test in May 2000.

With ebXML as a foundation, OTA can build a set a specifications addressing issues of travel planning, availability, package tours, reservations, and fulfillment for travel services and still leave open the opportunity for other kinds of services anywhere in the world to interact with this industry. Once operational, this extensible framework will provide tangible benefits for travelers and open up new opportunities for the industry as a whole.

1.5 Relationships with other industry systems

Electronic Data Interchange (EDI) - EDI is the computer-to-computer exchange of structured data between organizations according to a standard format. EDI can use any transport protocol that allows for exchanges of data between systems and organizations, but it generally uses standard formats such as X12 and UN/EDIFACT that pre-date the Web and Internet. While OTA specifications enable data exchanges among individuals and organizations, and create a common vocabulary of data elements and tags, they do not follow the established EDI standards.

Global Distribution Systems - GDSs centralize, consolidate and deliver travel supplier information for the online booking of reservations. Travel agencies are the primary users of GDSs, but direct-to-customer GDS services have recently become available over the Web. GDSs currently present information only on the companies and information that subscribe to their services and supply data to them. OTA's work can be expected to make it easier for GDSs to extend the range of the content they offer. OTA specifications offer the exchange of messages with profile data for implementation by any travel supplier or data service that adheres to these specifications based on open standards.

1.6 Relationships with other standards

Extensible Markup Language (XML) - OTA specifications use XML to represent the data used for travel transactions, as required by XML Recommendation 1.0 of the World Wide Web Consortium.

World Wide Web - The World Wide Web (WWW or the Web) is a collection of interconnected documents on the Internet that adhere to the accepted conventions for the identification and display of those documents. Most implementations of OTA specifications will use Web-related protocols to exchange XML messages as specified in this document.

---

[7] "Enabling Electronic Business with ebXML," ebXML White Paper, December 2000.

XML Schema - OTA 2001C embraces the Recommendation of the World Wide Web Consortium (W3C) for XML schema (2001) and follows its conventions as guidelines for the implementation of XML messages

Hotel industry specifications - In the hotel industry, two organizations have defined common data elements for transactions among trading partners: the Hotel Electronic Distribution Network Association (HEDNA) and the Hospitality Industry Technology Integration Standards (HITIS). The HITIS standards used the work of the Windows Hospitality Interface Specifications (WHIS) for their base documents in the development of their standards. HEDNA made its descriptive content specifications available to both organizations, and OTA relied upon the work of WHIS and HEDNA in the development of OTA specifications.

1.7 Status of this document

This document is a working draft publication of the OTA 2001C specification to be released for public review in December, 2001. It is anticipated that this document will undergo a period of commentary and resolution of comments may result in revisions prior to final publication. Individuals and companies implementing the specification or who would otherwise like to give their comments on the document should address them by e-mail to OTAcomments@disa.org .

OTA also welcomes comments by fax to +1 703-548-1264 or by mail to:

OpenTravel Alliance, Inc.
333 John Carlyle Street, Suite 600
Alexandria, Va. 22314 USA

2 About this Document

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

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

2.1 Intended Audience

This document serves two distinct audiences:

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

2.2 Relationship with previous OTA standards

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

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

2.3 Relationship with ebXML standards

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

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

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

- to provide a viable and robust infrastructure which is both open and available
- with off-the-shelf implementations on a variety of platforms

- which will allow on-the-wire interoperability between implementations

2.4 Definitions and Conventions

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

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

2.5 Use of Namespaces in this specification

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

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

The following table defines the other namespaces used within this document and their applicable namespaces:

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

3 OTA XML Best Practices

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

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

This document defines the Open Travel Alliance's Best Practices Guidelines all of OTA's XML data assets.

3.1 The XML Standard Specifications

Currently, there are several XML related specification recommendations produced by W3C. This document only refers to the W3C recommendations and versions listed below:

- Extensible Markup Language (XML) 1.0 (Second Edition) :

http://www.w3.org/TR/2000/REC-xml-20001006

- XML Schema Parts 0 - 2:

http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

3.2 Best Practices

3.2.1 Scope

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

The general OTA guideline approach is to maximize component (elements/attributes) reuse for the highly diverse and yet closely related travel industry data. The best practice to accomplish this is by creating OTA namespaced, complexType definitions as object containers for elements.

To effectively control the large amount of OTA XML data assets created by multiple vertical working groups, an XML component and schema based repository is required.

3.2.2 XML Component Parts and Role

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

- Tag Naming conventions

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

Each of the six items above play a unique role, supporting a common vocabulary, syntax, and semantic grammar of XML Schema and XML components (elements and attributes) definitions. Also, each of the Standards details its specific role in the rationale section. This document defines OTA standards for all XML data assets.

3.3 OTA XML Guidelines

OTA's XML Guidelines were composed from several internal and external technical sources and experts. This includes the OTA 2001A spec, W3C, OTA Architecture members, Galileo International, OASIS, IxRetail, ebXML as well as other technical sources.

I – 1	*XML Tag Names*
Guideline	Use mixed case tag names, with the leading character of each word in upper case and the remainder in lower case without the use of hyphens between words (a.k.a. "UCC camel case" or "PascalCasing").
Rationale	This format increases readability and is consistent with common industry practices. Examples of UCC camel case names are: "WorkAddress" or "PostalCode".

I – 2	*XML Tag Names*
Guideline	Acronym abbreviations are discouraged, but where needed, use all upper case.
Rationale	In some cases, common acronyms inhibit readability. This is especially true for internationally targeted audiences. However, in practice, business requirements and/or physical limitations may require the need to use acronyms. Examples are: `<BusinessURL>` `<HomeAAA>`

I – 3	*XML Tag Names*
Guideline	Word abbreviations are discouraged, however where needed, word abbreviations should use UCC camel case.
Rationale	Abbreviations may inhibit readability. This is especially true for internationally targeted audiences. However, in practice, business requirements and/or physical limitations may require the need to use acronyms. Examples are: `<ProductInfo>` `<BldgPermit>`

I – 4	*XML Tag Names*
Guideline	Element and attribute names should not exceed 25 characters. Tag names should be spelled out except where they exceed 25 characters, then standardized abbreviations should be applied. `Example: <ShareSyncInd>`
Rationale	This approach can reduce the overall size of a message significantly and limit impact to any bandwidth constraints.

681

I – 5	*XML Tag Names*
Guideline	Where the merger of tag name words and acronyms cause two upper case characters to be adjacent, separate them with an underscore ('_').
Rationale	This technique eliminates or reduces any uncertainty for tag name meaning. `Example: <PO_Box> or <UDDI_Keys>`

I – 6	*XML Tag Names*
Guideline	Use common tag name suffixes for elements defined by similar or common XML Schema type definitions.
Rationale	This approach supports a consistent syntax and semantic meaning for elements and attributes.

	`Example: <ContactAddress>`
	`<HomeAddress>`
	`<WorkAddress>`

682

I – 7	*XML Tag Names – 'Types'*	
Guideline	The OTA approved, defined or derived XML Schema type definitions (includes simpleTypes, complexTypes, attributeGroups and groups) should incorporate the following list of suffixes for naming type labels. However, if a user defined 'simpleType' definition is identical to a built-in XML Schema type, the built-in type definition should be used.	
	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]]]).
	Boolean	Examples: true, false or yes, no or on, off or…
	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.
	Measure	A numeric value that is always associated with a unit of measure.
	Type	An enumerated list of values from which only one value can be chosen at a time.
Rationale	This approach supports a consistent syntax and semantic meaning for XML Schema definitions and does not affect the naming of element and attribute tags in an instance document.	

683

II – 1	***Elements vs. Attributes***
Guideline	For a given OTA data element, the preferred method is to represent that data-element as an attribute. The data-element is represented as an element if and only if: • it is not atomic (i.e. It has attributes or child elements of its own) OR • the anticipated length of the attribute value is greater than 64 characters OR • presence or absence of the attribute represents a semantic 'choice' or branch within the schema OR • the data element repeats (i.e. may have more than one instance)
Rationale	The intention is to create a consistent OTA message design approach and to reduce the overall message size as well as to avoid the potential of tag naming collisions.

684

II – 2	***Elements vs. Attributes***

Guideline	A single XML <element> container should be used for “simpleType” repeating content (via the XML Schema “list” construct) and multiple XML element containers should be used for repeating “complexType” elements (i.e. elements with attributes or child elements). Note – caution should be used for the repeating “string” simple type which may contain space characters.
	Examples:
	`<StateCodes>CO IL FL MI</StateCodes>`
	`<Addresses>`
	`<Address>`

Rationale	The XML Schema "list" construct can significantly reduce the size of an XML message instance.

685

III – 1	*DTD vs. XML Schema*
Guideline	The XML Schema specification standard from W3C should be used to define XML message documents.
Rationale	Schemas are written in XML syntax, rather than complex SGML regular expression syntax. Because XML Schemas are themselves well-formed XML documents, they can be programmatically generated and validated using a meta-schema -- a schema used to define other schema models. XML schemas have built-in datatypes and an extensible data-typing mechanism. (DTDs only understand markup and character data.) Using an XML syntax to define data model requirements allows for more constraints, strong datatyping, etc. Provides for a consistent Data Repository syntax.

686

IV – 1	*Global vs Local Element Types and Elements/Attributes*
Guideline	Define XML Schema element types globally in the namespace for the elements that are likely to be reused (instead of defining the type anonymously in the Element declaration). This applies to both simpleType and complexType element type definitions.
Rationale	This approach supports a domain library or repository of reusable XML Schema components. Also, since Schema type names are not contained in XML instance documents, they can be verbose to avoid Schema element type naming collisions.

687

IV – 2	*Global vs Local Element Types and Elements/Attributes*
Guideline	Define XML Schema elements as nested elements via the 'type' attribute or an inline type definition ('simpleType' or 'complexType') instead of the 'ref' attribute that references a global element.
Rationale	This approach for local element naming reduces the possibility of tag name collisions and allows the creation of short tag names. Globally defined elements should be reserved only for travel domain elements with well-defined meanings; such global names should be constructed with sufficient roots and modifiers to identify their domain of use and avoid, tag naming collisions.

688

IV – 3	*Global vs Local Element Types and Elements/Attributes*
Guideline	Define common attribute parameters globally as a reusable component via the XML Schema 'attributeGroup' element definition.

Rationale	This approach supports a domain library or repository of reusable XML Schema components. Also, since the names used for the XML Schema 'attributeGroup' component is not contained in XML instance documents, they can be verbose to avoid naming collisions of other 'attributeGroup' definitions.

689

V – 1	*Namespaces*
Guideline	All schemas specified as compliant to OTA's XML message specifications shall put the global names they declare in one namespace; this shall be the 'OTA' namespace, which is http://www.opentravel.org/OTA.
Rationale	This approach supports a consistent way to manage and identify OTA's XML based, transaction assets both internally and externally (via trading partners and global e-business repositories such as UDDI). It also avoids the need for explicit prefixes on both schema and instance docs.

690

V – 2	*Namespaces*
Guideline	Each XML instance document produced by the 'OTA' namespaced Schemas should specify a default namespace and that should be the 'OTA' namespace defined above. Also, a namespace prefix of "OTA" is to be reserved for the 'OTA' namespace and used were 'OTA' is required not to be a default namespace to satisfy unique business needs.
Rationale	The same rationale as V-1 above. Also, provides a standard way for "OTA" namespaced content to be merged with other Industry or Trading Partner namespace content.

691

V – 3	*Namespaces*
Guideline	Each XML schema document produced as an 'OTA' namespaced schemas should specify a default namespace and a targetNamespace and both should be the 'OTA' namespace.
Rationale	The same rationale as V-1 above.

692

VI – 1	*Versioning XML Schemas and XML*

Guideline

Each version of a schema produced under the 'OTA' namespace must have a unique URI value for the schemaLocation attribute. The URI must be in a hierarchy approved by OTA's InterOperability committee. Each schemaLocation will be the URI for a UTF-8 file subordinate to:

http://www.opentravel.org/OTA/2002A-REC/domain_path/

where:

- '2002A' – is the bi-annual year of release (ie. 2002B, 2003A).
- 'REC' – is an official OTA XML Schema 'Recommendation'
  - the value of 'CAN' will used here for 'Candidate' Recommendations.

'domain_path' recommendations are:

'OTA-atomic-when-where' 'OTA-atomic-who-what'

'AWG-availability' 'HWG-general'

'AWG-booking' 'HWG-availability'

'AWG-general' 'HWG-booking'

'CWG-availability' 'LWG-general'

'CWG-booking' 'LWG-availability'

'CWG-general' 'LWG-booking'

'TIWG-profile' 'TIWG-itinerary'

'TIWG-generic' (ie. 'POS' or 'Generic Cancel')

Note – a complete schemaLocation URI which identifies an extended base OTA schema (see Guideline VI-2) must have the version extension qualifier added to the schema filename as shown below where '-23' is this qualifier:

http://www.opentravel.org/OTA/2002A-REC/CWG- availability/VehAvailRateRQ-23.xsd

Rationale

Using a version mechanism that parallels the schema-discovery mechanism of validating XML parsers is desirable and is supported by many schema validation tools.

Additionally, having a versioning scheme that mimics OTA's specification release methodology reduces the overall work effort of both schema publication and maintainability (especially when using a repository tool such as XML Cannon).

Similarly, the 'domain_path' recommendations can greatly reduce the work required to maintain and XML Schemas and schema fragments. This feature can be further enhanced by supplying the 'domain_path' as schema meta-data in a repository tool (like XML Cannon).

Also, the 2001C Service and Action transaction model can leverage the 'domain_path' as the Service token.

693

VI – 2	*Versioning XML Schemas and XML*

Guideline	The root tag for all OTA XML payload instances should contain a 'version' attribute (obtained from the 2001C attributeGroup 'PayloadStdAttributes') whose value will mimic the OTA version release plus OTA restricted, extension meta-data (if extensions are present). Similarly, the 'version' attribute of the <xsd:schema> opening tag should be specified and should have the same string value as the a fore mentioned root tag version attribute. Some examples are: `'2002A'; '2002B.23'; '2003A.Tabc123'; '2001C.Txyz456'` where: • '2002A' – is the bi-annual year of release (ie. 2002B, 2003A). • '.23' – A numeric reference value '.nn' to the base OTA schema version (ie. 2002A) which contains extensions derived from an XML Schema simpleType or from an existing OTA Schema type (simple or complex). Note – this extension type is primarily 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 byOTA's InterOperability committee). • '.Tabc123' – This extension type flags the presence of the <TPF_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.
Rationale	This approach supports automated schema discovery and provides sufficient version meta-data for repository maintenance. It also provides a quick and simple way for human or machine users to identify XML message transaction versions.

694

VII – 1	*XML Markup – General*
Guideline	The attribute schemaLocation replaces the DOCTYPE and can be used on elements in instances to name the location of a retrievable schema for that element associated with that namespace.
Rationale	Supports OTA's decision to use XML Schemas, which are not aware of this construct.

695

VII – 2	*XML Markup – General*
Guideline	OTA approved XML Schemas will use the <documentation> sub-element of the <annotation> element for documenting XML Schemas. DTD style 'Comments' usage should be limited.
Rationale	Comments are not part of the core information set of a document and may not be available or in a useful form. However, <documentation> elements are available to users of the Schema.

696

VII – 3	*XML Markup – General*

Guideline	OTA approved XML Schemas will avoid the use of PIs by replacing them with the <appinfo> sub-element of the <annotation> element which supplies this functionality.
Rationale	<appinfo> elements are available to users of the Schema. PIs require knowledge of their notation to parse correctly. Extensions to the XML Schema can be made using <appinfo>. An extension will not change the schema-validity of the document.

VIII – 1	*OTA General*
Guideline	Proprietary trading partner data can be included in an XML instance message within the <TPA_Extension> element (defined type of xsd:anyType) at OTA scantioned plug-in points. This element will also contain the boolean attribute 'mustProcess' which notifies that the message receiver must process the 'TPA_Extension' data.
Rationale	This approach (along with the versioning Guideline of VI-2) provides a standard way for OTA to integrate and manage proprietary trading partner information.

VIII – 2	*OTA General*
Guideline	Whenever possible, OTA schema data types should use the standard built-in simple types defined in the XML Schema specification.
Rationale	Simplifies OTA message implementation because validation tools support built-in XML Schema simple types.

VIII – 3	*OTA General*
Guideline	OTA XML Schemas should avoid rigid type restrictions unless the type is a common industry standard which is unlikely to change.
Rationale	This approach allows OTA defined messages to inter-operate globally more seamlessly and allows any particular trading partner to locally restrict content values as needed for unique business requirements.

3.4 Use of XML Technology

OTA 2001C messages MUST meet the requirements of well-formed XML documents, as specified in XML Recommendation 1.0, published by the W3C. The document is available from the W3C online at http://www.w3.org/TR/1998/REC-xml-19980210 .

OTA messages SHOULD begin with the XML declaration indicating use of XML version 1.0. That declaration reads:

```
<?xml version ="1.0"?>
```

OTA version 1 was based solely on validation to document type definitions (DTDs) while version 2001A and 2001B contained a combination of both DTDs and schemas. As XML Schema is now a recommendation and several XML parsers are available with strong schema support OTA will only publish schemas beginning with this 2001C standard and subsequently.

3.4.1 Recommendations for OTA Instance Documents

**FIXME

3.5 Versioned error messages

The OTA specification provides for error messages as part of the versioned messages, when those errors result from interactions with the trading partner's server. This section outlines specific requirements for the versioned error messages.

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

3.5.1 Versioned Standard Payload Attributes

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

- *EchoToken*: a sequence number for additional message identification assigned by the requesting host system. When a request message includes an *EchoToken*, the corresponding response message MUST include an *EchoToken* with an identical value.
- *TimeStamp*: indicates the creation date and time of the message in UTC using the following format specified by ISO 8601: YYYY-MM-DDThh:mm:ssZ with time values using the 24-hour (military) clock. e.g.: 20 November 2000, 1:59:38pm UTC becomes 2000-11-20T13:59:38Z
- *Target*: indicates if the message is a test or production message, with a default value of Production. Valid values: (Test | Production)
- *Version*: This optional attribute is NOT used in OTA non-versioned messages. For all OTA versioned messages, the version of the message indicated by an integer value.
- *SequenceNmbr* - This optional attribute is used to identify the sequence number of the transaction as assigned by the sending system. Allows for an application to process messages in a certain order, or to request a resynchronization of messages in the event that a system has been offline and needs to retrieve messages that were missed.

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

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

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

3.5.2 Use of entity module for versioned message responses

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

```
<!-- Copyright (C) 2000 OpenTravel Alliance -->
<!-- OTA v2 Specification - standard parameter entities -->

<!-- The following entity defines standard attributes -->
<!-- that appear on the root element for all OTA v2 payloads. -->
<!ENTITY % OTA_PayloadStdAttributes
   " EchoToken  CDATA #IMPLIED
     TimeStamp  CDATA #IMPLIED
     Target  (Test | Production)'Production'
     Version CDATA #IMPLIED
     SequenceNmbr CDATA #IMPLIED
>
<!-- The following parameter entities represent success and failure -->
<!-- in processing respectively. -->
<!ENTITY % OTA_SucceedDef "(Success|Warnings)">
<!ENTITY % OTA_FailureDef " Errors ">

<!ENTITY % ErrorAttr
" Type (Unknown|
   NoImplementation|
   BizRule|
   Authentication|
   AuthenticationTimeout|
   Authorization|
   ProtocolViolation|
   TransactionModel|
   AuthenticationModel|
   ReqFieldMissing) #REQUIRED
   Code   CDATA #IMPLIED
   DocURL    CDATA #IMPLIED
   Status    CDATA #IMPLIED
   Tag    CDATA #IMPLIED
   RecordId  CDATA #IMPLIED"
>
<!-- Standard way to indicate success processing an OTA message -->
<!ELEMENT Success EMPTY>

<!-- Indicates successful processing, but warnings occurred -->
<!ELEMENT Warnings (Warning+)>

<!-- Indicates errors occurred during processing -->
<!ELEMENT Errors (Error+)>

<!ELEMENT Warning (#PCDATA)>
<!ATTLIST Warning %ErrorAttr;>
<!ELEMENT Error (#PCDATA)>
<!ATTLIST Error %ErrorAttr;>
```

This module may be then included in other payload definition DTDs[8] . An example will help illustrate. In this hypothetical example, we define an <ExampleRS> payload:

```
<!-- Example Response message using standard parameters -->
<!ENTITY % OTA_v2ent SYSTEM "OTA_v2ent.mod">
%OTA_v2eent;
```

[8] N.B. This is the essence of the "schema fragment" method of isolating change within a network of DTD documents forming a specification. Carefully followed, this method allows once-and-only-once definition of all elements and entities.


```
<!ELEMENT ExampleRS ((%OTASucceedDef;,Example)|%OTAFailureDef;)>
<!ATTLIST ExampleRS %OTA_PayloadStdAttributes;>
<!-- Import content for the <Example> element -->
<!ENTITY % ExampleDef SYSTEM "example.dtd">
%ExampleDef;
```

Note how the definition of the <Example> element, the central element in this <ExampleRS> response message, is imported from another reusable DTD ("*example.dtd*" in this case).

The attributes of a <Warning> or <Error> element are identical and defined in the Error Attributes entity, %ErrorAttr; as follows:

Type - The Error element MUST contain the *Type* attribute that uses a recommended set of values to indicate the error type. The validating DTD can expect to accept values that it has NOT been explicitly coded for and process them in an acceptable way. This is facilitated by accepting the *Type* of "*Unknown*". The initial enumeration list MUST contain:

- *Unknown* - Indicates an unknown error. It is recommended that additional information be provided within the PCDATA, whenever possible.
- *NoImplementation* - Indicates that the target business system has no implementation for the intended request. Additional information may be provided within the PCDATA.
- *BizRule* - Indicates that the XML message has passed a low-level validation check, but that the business rules for the request, such as creating a record with a non-unique identifier, were not met. It is up to each implementation to determine when or if to use this error type or a more specific upper level content error. Additional information may be provided within the PCDATA.
- *AuthenticationModel* - Indicates the type of authentication requested is not recognized. Additional information may be provided within the PCDATA.
- *Authentication* - Indicates the message lacks adequate security credentials. Additional information may be provided within the PCDATA.
- *AuthenticationTimeout* - Indicates that the security credentials in the message have expired. Additional information may be provided within the PCDATA.
- *Authorization* - Indicates the sender lacks adequate security authorization to perform the request. Additional information may be provided within the PCDATA.
- *ProtocolViolation* - Indicates that a request was sent within a message exchange that does not align to the message protocols. Additional information may be provided within the PCDATA.
- *TransactionModel* - Indicates that the target business system does not support the intended transaction-oriented operation. Additional information may be found within the PCDATA.
- *Authentication Model* - Indicates that the type of authentication requested is not supported. Additional information may be provided within the PCDATA.
- *ReqFieldMissing* - Indicates that an element of attribute that is required by the DTD or Schema (or required by agreement between trading partners) is missing from the message.

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

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

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

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

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

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

```
<OTA_ReadProfileRS
   EchoToken="12346" TimeStamp="2000-11-20T20:15:21Z" Version="2">
   <Errors>
      <Error Type="BizRule"> UniqueId 09782345768 not found</Error>
   </Errors>
</OTA_ReadProfileRS>
```


4 Generic Messages and the Service/Action Model

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

4.1 The Service/Action Concept

The Service/Action model is conceptually similar to a Web Services model where an implementation provides one or more high-level services, each of which has one or more applicable actions (these equate to methods available on services). When used in conjunction with Request/Response message pairs the model is conceptually similar to an RPC[9], though in fact the underlying messaging substrate need not rely on RPCs.

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

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

4.1.1 Service/Action Message Mappings

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

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

Example 1 - MessageHeader showing a read operation[10] on the AirBooking service

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

[9] 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.

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


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

As such, the following Services are defined within OTA[11]:

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

4.2 Unique Identifiers within OTA Messages

Each record that identifies a unique account of travel information, such as a profile or reservation record, MUST have a unique identifier assigned by the system that creates it with the tag name `<UniqueId>`. The unique identifier on the record MUST contain both a *Type* and an *Id*

[11] Working groups should define additional services as new message sets are defined. See section 7 for detail mappings of messages onto specific actions on these services.

attribute. It MAY optionally include a *URL* and an *Instance* attribute. The syntax for a <UniqueId> is formally defined in the following Schema fragment:

Schema 1 - <UniqueID>:

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

- *URL* - This optional attribute is what makes a <UniqueId> instance globally unique outside the context of a single bilateral conversation between known trading partners. OTA recommends that the URL referenced by the URL identifying the public HTTP v1.1 OTA discovery message implementation for each trading partner. ***Note***: In the absence of having a public URL, the reference for this attribute could be determined by bilateral agreement.
- *Type* - This enumerated attribute references the type of object this <UniqueId> refers to, and gives this element its generality. By convention, the *Type* attribute value is the same as the OTA element tag name for the referenced object, for example, "Profile" or “Reservation”. As additional message types are defined in future versions of OTA specifications, the *Type* attribute enumeration will expand to include additional tag names values where a <UniqueId> applies.
- *Id* - This represents a unique identifying value assigned by the creating system, using the XML data type ID. The *Id* attribute might, for example, reference a primary-key value within a database behind the creating system’s implementation.
- *Instance* - This optional attribute represents the record as it exists at a point in time. An *Instance* is used in update messages where the sender must assure the server that the update sent refers to the most recent modification level of the object being updated.

Possible implementation strategies for *Instance* values are:

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

4.2.1 Examples of unique identifiers

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

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

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

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

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

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

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

4.3 Generic Infrastructure Messages

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

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

4.3.1 Create messages

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

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

Example 2 – Create request message:[12]

```
<OTA_CreateProfileRQ EchoToken="12345" TimeStamp="20001217T182248Z">
   <UniqueId Type="Profile"/>
   <Profile>
      <Customer Gender="Male">
         <PersonName>
            <GivenName>John</GivenName>
            <SurName>Smith</SurName>
         </PersonName>
            .....
      </Customer>
   </Profile>
</OTA_CreateProfileRQ>
```

Example 3 - Create response message:

```
<OTA_CreateProfileRS EchoToken="12345" TimeStamp="20001217T182259Z"
   <Success />
   <UniqueId
      URL="http://www.sixcontinents.com/OTAProcessor"
      Type="Profile"
```

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


```
      Id="1234567"/>
</OTA_CreateProfileRS>
```

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

4.3.2 Generic Read message

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

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

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

Example 4 - OTA ReadRQ message:

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

ReqRespVersion - The optional "Request Response Version" attribute allows the sender to indicate the version desired for the response message. For example, the OTA_ReadRQ message sample above indicates a request to return a 2001A OTA Customer Profile.

This request applies to all types of actions in addition to profiles. The type of the generated response depends, of course, on the *Type* specified in the request; for example, the `<OTA_ReadRQ>` shown in the example would generate a `<OTA_ReadProfileRS>` message that contains a 2001A OTA Profile as a response, as in the example below.

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

This revision affects the original non-versioned version discovery messages published in the OTA version 1 specification, but with minor adjustments implementors of OTA specifications are able to take advantage of the significant long-term benefits that generalization provides.[13]

Example 5 - Read response message:

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

[13] See Chapter 3 - Errors and Non-Versioned Messages


```
            ...
         </Profile>
      </OTA_ReadProfileRS>
```

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

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

Schema 2 - <OTA_ReadRQ>:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
   <xsd:include schemaLocation="OTA_v2ent.xsd"/>
   <xsd:element name="OTA_ReadRQ">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="UniqueId"/>
         </xsd:sequence>
         <xsd:attribute name="EchoToken" type="xsd:string"/>
         <xsd:attribute name="TimeStamp" type="xsd:string"/>
         <xsd:attribute name="Target" use="default" value="Production">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="Test"/>
                  <xsd:enumeration value="Production"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
         <xsd:attribute name="Version" type="xsd:string"/>
         <xsd:attribute name="SequenceNmbr" type="xsd:integer"/>
         <xsd:attribute name="ReqRespVersion" type="xsd:string"/>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

4.3.3 Generic Update message

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

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

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

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

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

The definition of the Update message is lengthy. An example update request for a customer profile modification is shown below. A detailed treatment of update requests, how to generate and approaches to processing them is included in an appendix (see section 6 – OTA Update Messages).

Example 6 - OTA_UpdateRQ[14]:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.2 (http://www.vmguys.com/vmtools/) -->
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
              ReqRespVersion="2"
              Timestamp="2001-10-12T16:07:17EDT">
    <UniqueId Type="Profile"
              Id="987654321"
              URL="http://www.vmguys.com/OTAEngine/"
              Instance="20011012160659" />
    <Position XPath="/Profile/Customer/AddressInfo/Address/StreetNmbr">
        <Attribute Name="PO_Box" Operation="delete" />
    </Position>
    <Position XPath="/Profile/Customer/TelephoneInfo">
        <Attribute Name="PhoneUse" Operation="modify" Value="Home" />
    </Position>
    <Position XPath="/Profile/Customer/PersonName/MiddleName">
        <Subtree Operation="delete" />
    </Position>
    <Position XPath="/Profile/Customer">
        <Subtree Operation="insert" Child="5">
            <RelatedTraveler Relation="Child">
                <PersonName>
                    <NameTitle>Ms.</NameTitle>
                    <GivenName>Amy</GivenName>
                    <MiddleName>E.</MiddleName>
                    <SurName>Smith</SurName>
                </PersonName>
            </RelatedTraveler>
        </Subtree>
        <Attribute Name="Gender" Operation="modify" Value="Male" />
    </Position>
</OTA_UpdateRQ>
```

4.3.4 Generic Delete message

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

The requestor MAY also verify the record before deleting it to ensure the correct record has been identified prior to deleting it. In this case, the use of the *Instance* attribute may also be useful in

[14] Additional examples, including before and after images, may be found in section 6 – OTA Update messages

determining whether the record has been updated more recently than the information that is intended to be deleted. That choice, again, would be dictated by good business practices.

Steps in the Delete operation include:

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

Example 7 - illustrating the Delete process, Read request:

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

Example 8 - illustrating the Delete process Read response:

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

Example 9 - illustrating the Delete request:

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

Example 10 - illustrating the Delete response:

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

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

Schema 3 - <OTA_DeleteRQ>:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
   <xsd:include schemaLocation="OTA_v2ent.xsd"/>
   <xsd:element name="OTA_DeleteRQ">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="UniqueId"/>
         </xsd:sequence>
         <xsd:attribute name="EchoToken" type="xsd:string"/>
         <xsd:attribute name="TimeStamp" type="xsd:string"/>
         <xsd:attribute name="Target" use="default" value="Production">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="Test"/>
                  <xsd:enumeration value="Production"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
         <xsd:attribute name="Version" type="xsd:string"/>
         <xsd:attribute name="SequenceNmbr" type="xsd:integer"/>
         <xsd:attribute name="ReqRespVersion" type="xsd:string"/>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

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

Schema 4 - <OTA_DeleteRS>:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
   <xsd:include schemaLocation="OTA_v2ent.xsd"/>
   <xsd:element name="OTA_DeleteRS">
      <xsd:complexType>
         <xsd:choice>
            <xsd:sequence>
               <xsd:choice>
                  <xsd:element ref="Success"/>
                  <xsd:element ref="Warnings"/>
               </xsd:choice>
               <xsd:element ref="UniqueId"/>
            </xsd:sequence>
            <xsd:element ref="Errors"/>
         </xsd:choice>
         <xsd:attribute name="EchoToken" type="xsd:string"/>
         <xsd:attribute name="TimeStamp" type="xsd:string"/>
         <xsd:attribute name="Target" use="default" value="Production">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="Test"/>
                  <xsd:enumeration value="Production"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
         <xsd:attribute name="Version" type="xsd:string" use="required"/>
         <xsd:attribute name="SequenceNmbr" type="xsd:integer"/>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```


4.3.5 Generic Cancel Request

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

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

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

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

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

1.1. Consequences of Canceling

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

1.2. Read Request Prior to Cancel

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

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

Example 11 - <OTA_ReadRQ> message:

```
   <OTA_ReadRQ EchoToken="12987567" TimeStamp= "20010331T13:19:42:49"
ReqRespVersion="1">
```



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

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

Example 12 - <OTA_ReadHotelResRS> message:Example: - OTA_ReadHotelResRS

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

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

1.3. Security considerations

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

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

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

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

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

level and match the request up to the channel and source to display only those reservations that were booked through that agency.

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

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

1.4. OTA Cancel Messages

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

The following action verbs are used as root elements of a payload document that seeks to cancel a reservation.

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

1.1.1 OTA Cancel Request

The root element of the OTA_CancelRQ contains the standard payload attributes found in all OTA payload documents as well as the attribute *ReqRespVersion*= that requests a specific version of the response message. As this is the first publication of the OTA cancel message set, currently the only valid value is "1".

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

Attributes of OTA_CancelRQ are as follows:

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

  Initiate - Indicates the initial request to cancel a reservation.

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

  Confirm - Indicates a request to complete the cancellation.

1.1.2 Cancel Request

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

Schema 5 - <OTA_CancelRQ> message:


```
**FIXME
not available at time of draft publication
**FIXME
```

The DTD fragment for the OTA_CancelRQ message is as follows:

```
<!ENTITY % OTA_PayloadStdAttributes "OTA_PayloadStdAttributes">
<!ELEMENT OTA_CancelRQ (UniqueId)>
<!ATTLIST OTA_CancelRQ
   %OTA_PayloadStdAttributes;
   ReqRespVersion CDATA #IMPLIED
   CancelType (Initiate | Ignore | Confirm) #REQUIRED
>
<!ELEMENT UniqueId EMPTY>
<!ATTLIST UniqueId
   URL CDATA #IMPLIED
   Type CDATA #REQUIRED
   Id CDATA #REQUIRED
   Instance CDATA #IMPLIED
>
```

1.1.3 Cancel Request - Sample XML message

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

```
<!DOCTYPE OTA_CancelRQ SYSTEM "OTA_CancelRQ.dtd">
<OTA_CancelRQ EchoToken="129845534672175" TimeStamp="2001-03-31T13:19:42-04:49"
Target="Production" Version="1" ReqRespVersion="1" CancelType="Initiate">
   <UniqueId URL="http://provider1.org/OTAEngine/" Type="Reservation"
Id="0507G4325" Instance="2001-06-03T13:09:21"/>
</OTA_CancelRQ>
```

1.5. Confirmation of Cancellation

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

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

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

1.1.4 OTA Cancel Response

The `<OTA_CancelRS>` message uses the external entity <% `OTA_v2ent.mod>` file that defines the root element standard attributes found in all OTA payload documents, and the response options of returning the indication of Success, Warnings or Errors in processing the request. Additionally, it returns a collection of cancellation rules, with penalty amounts, if incurred, and an indication of the status of the cancellation request, either "Cancelled", "Pending", or "Ignored" if the transaction has been rolled back and the reservation remains intact.

Attributes of OTA_CancelRS are as follows:

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

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

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

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

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

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

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

1.1.5 Cancel Rules

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

The attributes of the CancelRule element are as follows:

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

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

1.1.6 Cancel Response

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

Schema 6 - <OTA_CancelRS> message:

```
**FIXME
Schema fragment not available at time of draft publication
**FIXME
```


The DTD fragment for the OTA_CancelRS message is as follows:

```
<!ENTITY % OTA_v2ent SYSTEM "OTA_v2ent.mod">
<!ENTITY % OTA_PayloadStdAttributes "OTA_PayloadStdAttributes">
<!ENTITY % UniqueId "UniqueId"
>
<!ELEMENT OTA_CancelRS (UniqueId, CancelRules)>
<!ATTLIST OTA_CancelRS
   %OTA_PayloadStdAttributes;
   CancelStatus (Pending | Ignored | Canceled) #REQUIRED
   CancelID CDATA #IMPLIED
>
<!ELEMENT CancelRules (CancelRule+)>
<!ELEMENT CancelRule (#PCDATA)>
<!ATTLIST CancelRule
   CancelByDate CDATA #IMPLIED
   Amount CDATA #IMPLIED
   CurrencyCode CDATA #IMPLIED>
```

1.1.7 Cancel Response - Sample XML message

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE OTA_CancelRS SYSTEM "OTA_CancelRS.dtd">
<OTA_CancelRS EchoToken="129@845534672175" TimeStamp="2001-05-23T13:19:42-05:00"
Target="Production" Version="2" Status="Canceled" CancelID="1236597GQ345">
   <UniqueId URL="http://provider1.org/OTAEngine/" Type="Reservation"
Id="0507G4325" Instance="2001-06-03T13:09:21"/>
   <CancelRules>
      <CancelRule CancelByDate="2001-05-31T13:19:42-05:00" CurrencyCode="USD"
Amount="75.00">Cancellations within 30 days incur a penalty of $75.00</CancelRule>
   </CancelRules>
</OTA_CancelRS>
```


5 OTA Infrastructure

In this section we explore the revised OTA infrastructure. See section 8 for a list of changes since the last publication of the infastructure specification in 2001A.

5.1 Architecture Overview

OTA's Infrastructure Architecture borrows heavily from ebXML's Technical Architecture[15] [ebTA] and Message Service[16] [ebMS]. EbXML's Message Service, which is based on SOAP version 1.1 and the Soap with Attachments informational document, provides the functionality needed for two or more parties to engage in an "electronic business transaction". Products that implement the ebMS specification should be capable of the reliable and secure exchange of business data associated with a "business transaction". It is expected that any company implementing an OTA compliant system will be capable of supporting synchronous and asynchronous processing models as required by the various OTA message types described elsewhere in this document.

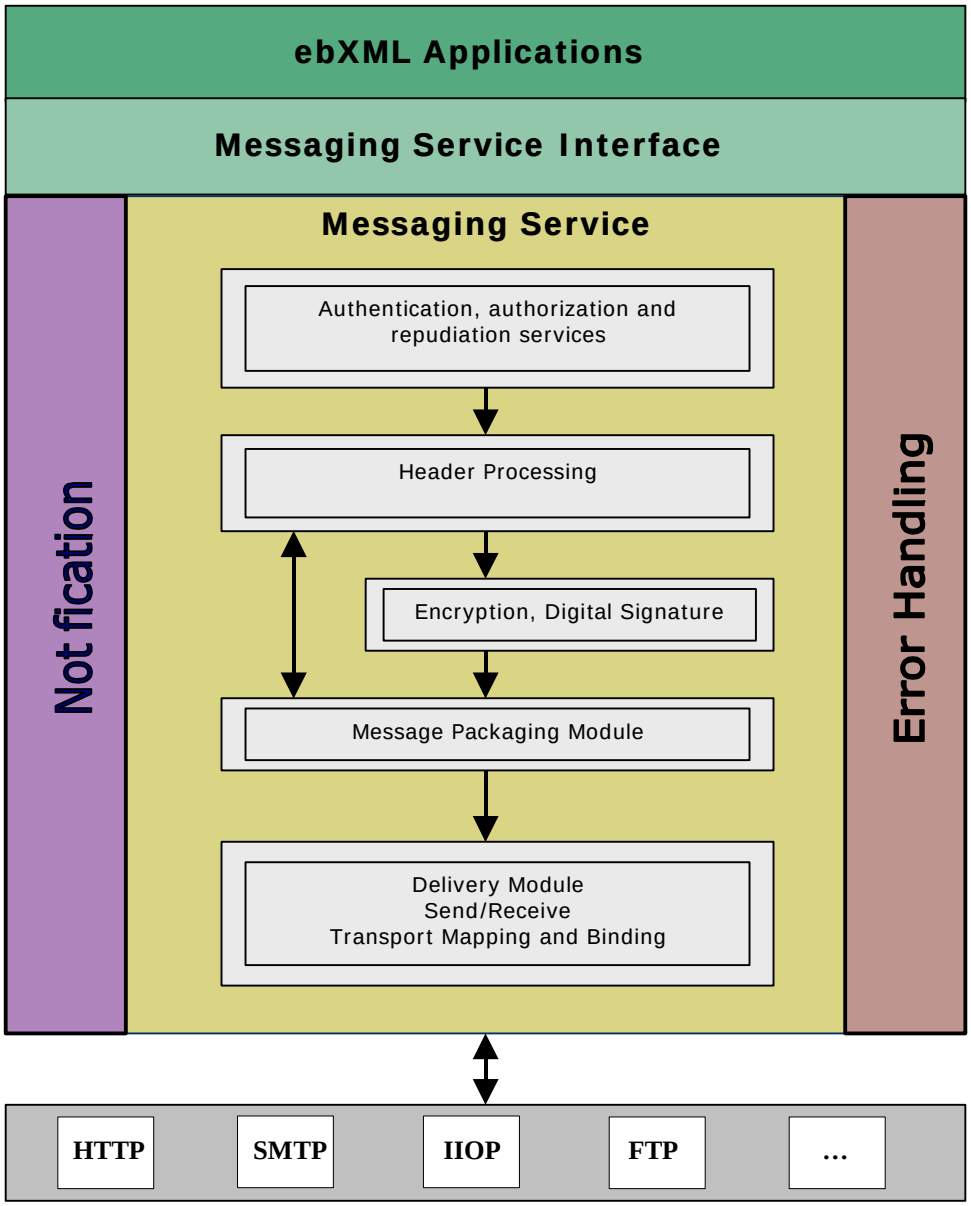
Business Applications interface with an ebMS through a "service interface", which is unique to each ebMS product. The "service interface" allows applications to request services and receive notifications from an ebMS product. When an application needs to engage in an electronic business transaction with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

An ebMS product vendor is responsible for implementing the functionality described in the ebMS specification [ebMS]. A high quality ebMS implementation would also provide facilities for encrypting and digitally signing data, access control, authentication and authorization, real-time error notifications, logging, auditing and administrative tasks.

Following is an architectural diagram of an ebMS:

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

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

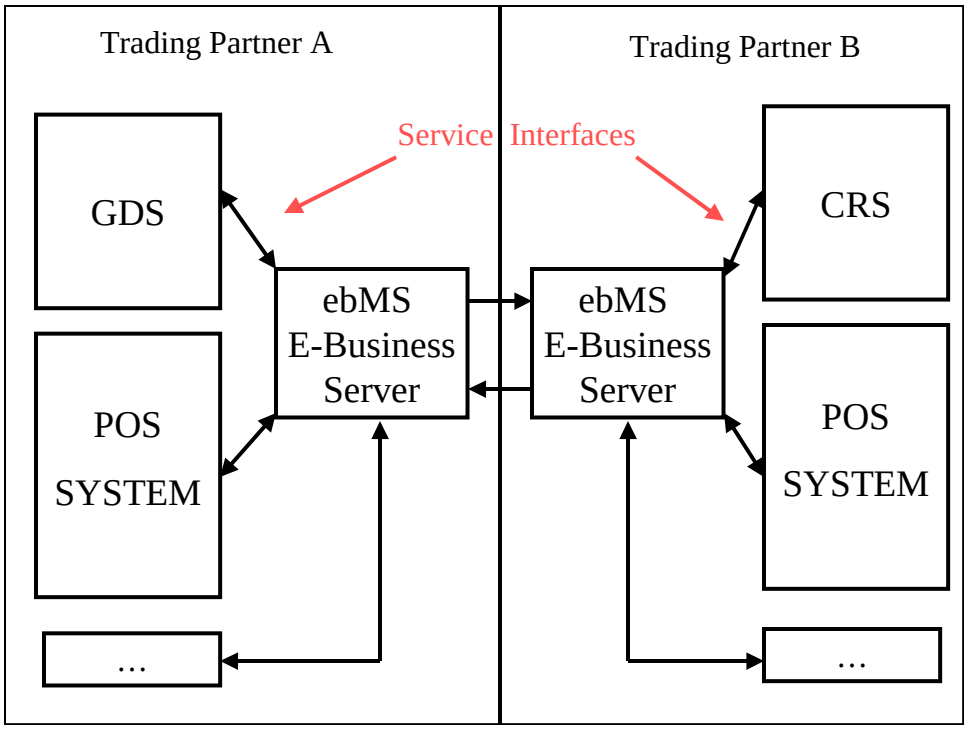


1557

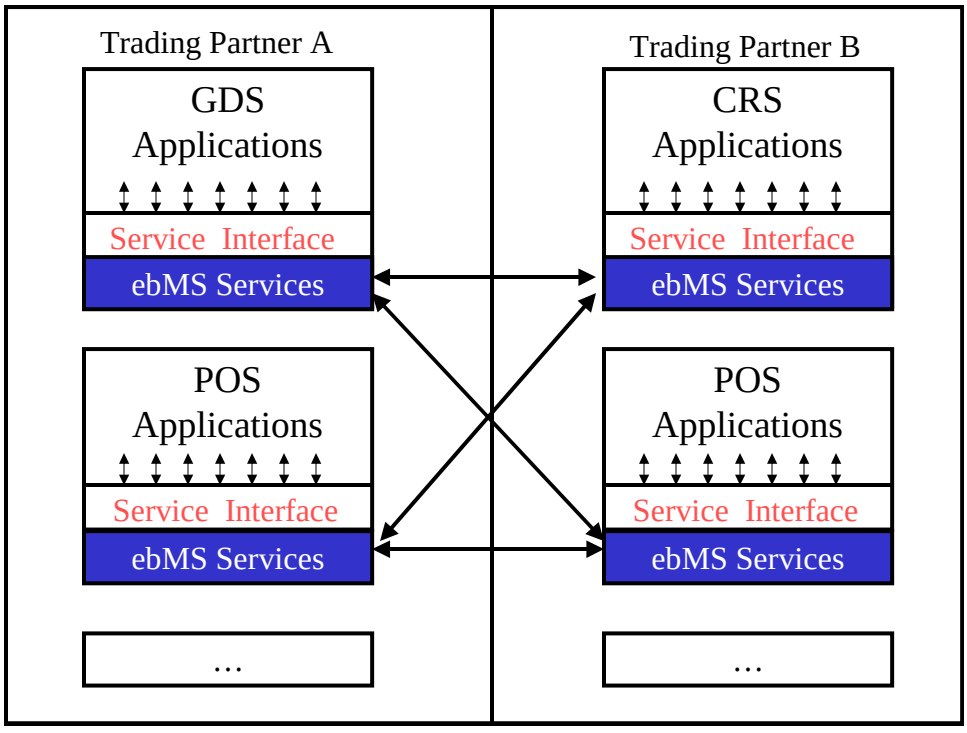
1558 5.1.1 Reference Model

1559 An ebMS may serve as a centralized "E-Business Server" to manage all of a company's
1560 electronic business transaction exchanges with trading partners. Alternatively ebMS functionality
1561 may be "integrated" within each business application that engages in electronic business
1562 transactions with trading partners. Regardless of which model is used implementers SHOULD
1563 provide the same service levels, access control, reliability, availability and security required by
1564 the electronic business transactions defined by OTA.

1565 Below is a reference model depicting a centralized E-Business Server:



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



5.1.2 Transport Protocols

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

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

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

5.1.3 Logging

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

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

5.1.4 Auditing

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

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

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

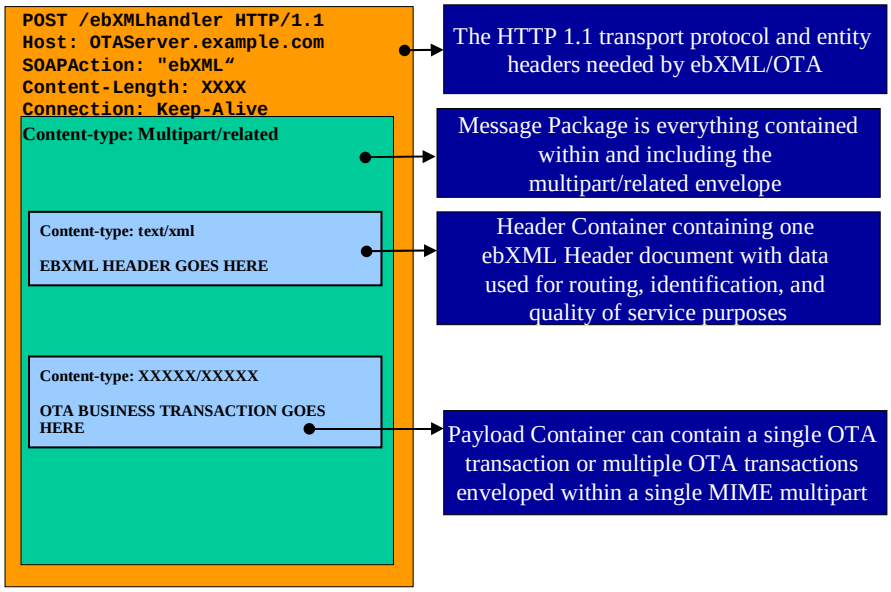
5.2 Message Structure and Packaging

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

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

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

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



5.2.1 The 'Unit-of-work' Concept

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

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

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

5.2.2 Packaging a single unit-of-work

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

```
Content-ID: airlinereservation111@imacompany.com
Content-Type: text/xml
```



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

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

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

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

--BoundarY
Content-ID: hotelreservation111profile@imacompany.com
Content-Type: text/xml

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

--BoundarY--
```

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

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

<OTA_HotelResNotifRQ>
</OTA_HotelResNotifRQ>

--BoundarY
Content-ID: hotelreservation112@imacompany.com
Content-Type: text/xml

<OTA_HotelResNotifRQ>
</OTA_HotelResNotifRQ>

--BoundarY
Content-ID: hotelreservation113@imacompany.com
Content-Type: text/xml

<OTA_HotelResNotifRQ>
</OTA_HotelResNotifRQ>
```



```
--BoundarY--
```

5.3 Classes of Message Delivery

EbXML is built on top of SOAP which does not inherently provide any underlying mechanism for reliable messaging. An ebXML implementation will however provide the ability to use reliable messaging for the transport of messages.

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

5.3.1 Historical Use within the travel industry

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

- Type A – interactive, at most once delivery
- Type B – store-and-forward, 'guaranteed' delivery

5.3.1.1 Type A[17]

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

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

5.3.1.2 Type B[18]

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

5.3.2 EbXML Classes of Delivery

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

---

[17] 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)

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

5.4 EbXML Header Document

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

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

5.4.1 OTA Subset of an ebXML Header Document

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

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

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

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

5.4.1.1 ErrorList Element

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

OTA maintains a list of error codes that may be used in response to any OTA request message. When reporting OTA errors within a response message the Error element used to report an OTA

error condition MUST contain a `codeContext` attribute with the value "http://www.opentravel.org/errorCodes". For example:

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

The following table contains all the possible OTA errors:

errorCode	*Severity*	*ErrorMessages*
SessionFailure	Error	OTA version not supported
SessionFailure	Error	Session has expired
SessionFailure	Error	Session already closed
SessionFailure	Error	Parameter not supported

5.4.1.2 Acknowledgement Element

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

The `Acknowledgement` element contains three REQUIRED attributes:

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

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

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

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

5.4.1.3 Via Element

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

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

REQUIRED attributes:

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

OPTIONAL attribute:

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

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

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

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

5.4.1.4 MessageHeader Element

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

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

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

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

5.4.1.4.1 To element

Contains one child element, PartyId, which contains the DUNS Number of the intended recipient. The PartyId element contains one attribute, "type", containing the value "urn:x12.org:I05:01". This value indicates the data contained in the PartyId element is semantically defined by the X12

organization (x12.org), I05 indicates the particular set of identifiers defined by X12 to identify "parties" and 01 indicates the content of PartyId is a DUNS Number. Following is an example of the To element:

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

5.4.1.4.2 From element

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

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

5.4.1.4.3 CPAId element

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

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

5.4.1.4.4 ConversationId element

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

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

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

5.4.1.4.5 Service and Action elements

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

The Action element is used to provide additional context to the Service element. The Action identifies the protocol "verb". Taken together the Service/Action should provide sufficient identification to invoke proper protocol behavior between parties. The list of possible values for

the Service element, the type attribute and the Action element are listed in section titled “Service and Action Mappings”.

Following is an example:

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

5.4.1.4.6 MessageData element

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

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

5.4.1.4.6.1 MessageId Element

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

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

5.4.1.4.6.2 Timestamp Element

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

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

5.4.1.4.6.3 RefToMessageId Element

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

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

5.4.1.4.6.4 TimeToLive Element

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

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

Both sending and receiving Message Service Handlers (MSH) are expected to generate and report an error condition whenever message delivery/processing has aborted due to expiration. A Sending MSH MUST notify a higher layer application, through its service interface or by some other means, whenever a message request aborts due to expiration. If a Sending MSH has been configured to retry message delivery for messages with BestEffort deliverySemantics (ref: retries in section 10.2.6 of [ebMS], NOTE: this is not the recommended behavior for BestEffort deliverySemantics) then the sending MSH SHOULD attempt to resend a message that aborts due

to expiration. The recommended minimum time for TimeToLive SHOULD be no less than 5 seconds in the future from the date time value contained in the MessageData/Timestamp.

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

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

5.4.1.4.7 QualityOfServiceInfo element

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

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

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

5.5 SOAP Body Elements

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

- **Manifest**

The ebXML Message Service specification defines another element, DeliveryReceipt, which is not used by OTA. Two additional elements, StatusRequest and Status Response are also defined by ebMS, which are presumed to be included in a standard ebXML product and will not be defined here.

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

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

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

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

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

5.6 EbXML Collaboration Protocol Profile

EbXML has defined the Collaboration-Protocol-Profile and Agreement Specification [ebCPP[19]] for the definition of key parameters used by trading partners within the context of their on-the-wire eCommerce conversations.

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

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

<tp:DocExchange tp:docExchangeId = "TYPEA">
  <tp:ebXMLBinding tp:version = "1.0">
     <tp:ReliableMessaging tp:deliverySemantics="BestEffort" tp:idempotency="true">
     </tp:ReliableMessaging>
  </tp:ebXMLBinding>
</tp:DocExchange>

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

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

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


```
                        tp:authenticated="true"
                        tp:authorized="true"/>
</tp:DeliveryChannel>

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

5.7 EbXML Header Examples

5.7.1 Type A Request Message

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

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
                    xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'
                    xmlns:xlink='http://www.w3.org/1999/xlink'>
<SOAP-ENV:Header>
   <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
      <eb:From>
         <eb:PartyId type="urn:x12.org:I05:01">123456789</eb:PartyId>
      </eb:From>
      <eb:To>
         <eb:PartyId type=" urn:x12.org:I05:01">912345678</eb:PartyId>
      </eb:To>
      <eb:CPAId>NULL</eb:CPAId>
   <eb:ConversationId>20001209133003285 72@b2b.company.com
            </eb:ConversationId>
      <eb:Service eb:type="OTA">Profile</eb:Service>
      <eb:Action>OTA_CreateProfileRQ</eb:Action>
      <eb:MessageData>
           <eb:MessageId>mid:20001209-133003-28572@b2b.company.com</eb:MessageId>
           <eb:Timestamp>2001-02-15T11:12:12Z </eb:Timestamp>
           <eb:TimeToLive>2001-02-15T11:12:17Z </eb:TimeToLive>
      </eb:MessageData>
      <eb:QualityofServiceInfo eb:deliverySemantics="BestEffort"/>
   </eb:MessageHeader>
<eb:Via SOAP-ENV:mustUnderstand=”1” SOAP-
ENV:actor=http://schemas.xmlsoap.org/soap/actor/next eb:version=”1.0”
eb:syncReply=”true” ></eb:Via>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
   <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
      <eb:Reference xlink:href="cid:profile111@b2b.company.com"
         xlink:type="simple">
      </eb:Reference>
   </eb:Manifest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.7.2 Type B Request Message

```
Content-ID: <ebxhmheader111@B2B.company.com>
```

```
Content-Type: text/xml; charset="UTF-8"

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
                    xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'
                    xmlns:xlink='http://www.w3.org/1999/xlink'>
<SOAP-ENV:Header>
   <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
      <eb:From>
         <eb:PartyId type="urn:x12.org:I05:01">123456789</eb:PartyId>
      </eb:From>
      <eb:To>
         <eb:PartyId type=" urn:x12.org:I05:01">912345678</eb:PartyId>
      </eb:To>
      <eb:CPAId>NULL</eb:CPAId>
   <eb:ConversationId>20001209133003283572@b2b.company.com
            </eb:ConversationId>
      <eb:Service eb:type="OTA">Profile</eb:Service>
      <eb:Action>OTA_CreateProfileRQ</eb:Action>
      <eb:MessageData>
           <eb:MessageId>mid:20001209-133003-28572@b2b.company.com</eb:MessageId>
           <eb:Timestamp>2001-02-15T11:12:12Z </eb:Timestamp>
      </eb:MessageData>
      <eb:QualityofServiceInfo eb:deliverySemantics="OnceAndOnlyOnce"/>
   </eb:MessageHeader>
<eb:Via SOAP-ENV:mustUnderstand=”1” SOAP-
ENV:actor=http://schemas.xmlsoap.org/soap/actor/next eb:version=”1.0”
eb:syncReply=”false” eb:ackRequested=”Unsigned”></eb:Via>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
   <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
      <eb:Reference xlink:href="cid:profile111@b2b.company.com"
         xlink:type="simple">
      </eb:Reference>
   </eb:Manifest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5.7.3 Type A Response Example

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

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
                    xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'
                    xmlns:xlink='http://www.w3.org/1999/xlink'>
<SOAP-ENV:Header>
   <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
      <eb:From>
         <eb:PartyId type="urn:x12.org:I05:01">912345678</eb:PartyId>
      </eb:From>
      <eb:To>
         <eb:PartyId type=" urn:x12.org:I05:01">123456789</eb:PartyId>
      </eb:To>
      <eb:CPAId>NULL</eb:CPAId>
   <eb:ConversationId>20001209133003283572@b2b.company.com
            </eb:ConversationId>
      <eb:Service eb:type="OTA">Profile</eb:Service>
      <eb:Action>OTA_CreateProfileRS</eb:Action>
      <eb:MessageData>
           <eb:MessageId>mid:20001209-133503-
1111@B2Bserver.imacompany.com</eb:MessageId>
```



```
            <eb:Timestamp>2001-02-15T11:15:12Z </eb:Timestamp>
                              <eb:RefToMessageId> mid:20001209-133003-
28572@b2b.company.com</eb:RefToMessageId>
      </eb:MessageData>
   </eb:MessageHeader>

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

5.7.4 Type B Response Example

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

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
                    xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'
                    xmlns:xlink='http://www.w3.org/1999/xlink'>
<SOAP-ENV:Header>
   <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
      <eb:From>
         <eb:PartyId type="urn:x12.org:I05:01">912345678</eb:PartyId>
      </eb:From>
      <eb:To>
         <eb:PartyId type=" urn:x12.org:I05:01">123456789</eb:PartyId>
      </eb:To>
      <eb:CPAId>NULL</eb:CPAId>
   <eb:ConversationId>20001209133003328572@b2b.company.com
           </eb:ConversationId>
      <eb:Service eb:type="OTA">Profile</eb:Service>
      <eb:Action>OTA_CreateProfileRS</eb:Action>
      <eb:MessageData>
           <eb:MessageId>mid:20001209-133503-
1111@B2Bserver.imacompany.com</eb:MessageId>
           <eb:Timestamp>2001-02-15T11:15:12Z </eb:Timestamp>
                              <eb:RefToMessageId> mid:20001209-133003-
28572@b2b.company.com</eb:RefToMessageId>
      </eb:MessageData>
   </eb:MessageHeader>
<eb:Acknowledgment SOAP-ENV:mustUnderstand="1" eb:version="1.0"
             SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
          <eb:Timestamp>2001-03-09T12:22:30Z</eb:Timestamp>
</eb:Acknowledgement>

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

5.7.4.1 Reliable Messaging

In the context of OTA, reliable messaging refers to the requirements that a ebXML Message Service Product be capable of reliably delivering and persisting incoming and outgoing payload data until the successful delivery to a trading partner or an internal application. It is expected that all ebXML compliant Message Service products are capable of performing reliable messaging in

accordance with the ebMS specification following the requirements for `reliableMessagingMethod="ebXML"` (the only type supported by OTA at this time and the default method for ebXML) in section 10.2.4.

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

5.7.4.2 Once and Only Once Messaging

ebXML supports two type of delivery Semantics:

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

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

CAUTION: Messages sent using the BestEffort deliverySemantics may result in failed delivery or multiple deliveries of the same message. Implementers should take caution when using type A messages to ensure that failed message delivery or multiple message delivery does not compromise the integrity of a system.

5.7.5 Mapping Class of Delivery to Service/Action Pairs

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

5.8 Sessions in OTA

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

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

5.8.1 What we mean by 'sessions'

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

- They are ongoing
- Session initiation and session termination are relatively low overhead
- Sessions are long-lived (they may last for days, even weeks on end)
- Sessions are durable
- Messages belonging to a session may be identified with that session via a conversation-id
- Session conversation-ids provide a convenient 'hook' for implementation end-points to maintain state (related, of course, to a particular session)

5.8.2 What we do not mean by 'sessions'

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

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

5.8.3 The OTA Session service

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

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

The Session service defines four possible Actions:

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

5.8.3.1 Session/CreateRQ and Session/CreateRS

5.8.3.1.1 Session/CreateRQ

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

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

A recurring OTA_Version element that is used to identify the various OTA versions supported by the sender. A single attribute, preference, containing a numerical value that is used to indicate the senders "preferred" version to be used during this session. The lowest preference value indicates the senders highest preference. A preference value of "0" or the absence of a preference attribute indicates no preference. An example SessionControlRequest document follows:

```
<SessionControlRequest>
<OTA_Version preference="1">2001C</OTA_Version>
<OTA_Version preference="2">2001A</OTA_Version>
</SessionControlRequest>
```


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

5.8.3.1.2 Session/CreateRS

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

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

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

- Approved
- Rejected

A single OTA_Version element identifies the OTA version that both parties agree to follow during the session.

A single ConversationId element containing the value of ConversationId the MUST be used in the MessageHeader/ConversationId element of all subsequent messages exchanged during this session.

Zero or more Reason elements indicating the reason why a Session was rejected. The Reason element MAY only be present when the status attribute of the SessionControlResponse element contains the value "Rejected". When a session has been Rejected the SessionControlResponse document MUST NOT contain a ConversationId element.

Zero or one ServiceSupported element, see section 7 for details of this element.

Following are two example SessionControlResponse documents:

```
<SessionControlResponse status="Approved">
<OTA_Version>2001C</OTA_Version>
<ConversationId>20011027081907-862@host.imacompany.com</ConversationId>
</SessionControlResponse>
```

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

5.8.3.2 Session/CloseRQ and Session/CloseRS

5.8.3.2.1 Session/CloseRQ

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

the value of the session to be closed. This is the same ConversationId value that was contained in the Session/CreateRS SessionControlResponse document when the session was Accepted**.**

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

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

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

There is no payload associated with this message.

5.8.3.2.2 Session/CloseRS

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

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

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

There is no payload associated with this message.

5.8.4 Securing OTA Sessions

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

5.8.4.1 Basic-authorization and SSL

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

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

5.8.4.2 Towards deeper security

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

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

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

6 OTA Update Messages

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

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

6.1 Representing change in XML

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

- library utilities[20] that can compare before and after images of a particular XML document and then generate a succinct representation of the differences (conceptually similar to a GNU *diff* utility)
- a representation for differences that is both human readable and understandable by automated tools
- library utilities that can take an XML document, apply a differences document and generate the same after image (conceptually similar to a GNU *patch* utility)

Applying an Update action involves a tree-to-tree comparison, similar to well-known operations outside the XML arena. This specification assumes that prior to sending an `<OTA_UpdateRQ>` message, the sender has used standard libraries to generate the differences between the 'before' and desired 'after' images of the XML document. Implementors may wish to consult documentation of algorithms for tree-to-tree comparison and correction in computer science publications[21], to provide the background for understanding this XML-specific approach.

6.2 Position Representation with XPath

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

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

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

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

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

It is important to note that when processing a differences document, insertion or deletion of nodes may invalidate subsequent XPath references. This is addressed in the section "Order of Representation and Application" (See Section 6.5).

6.3 Operands

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

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

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

6.4 Operations

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

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

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

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

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

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

Schema 7 - <OTA_UpdateRQ>:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
   <xsd:include schemaLocation="OTA_v2ent.xsd"/>
   <xsd:element name="OTA_UpdateRQ">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="UniqueId"/>
            <xsd:element ref="Position" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="EchoToken" type="xsd:string"/>
         <xsd:attribute name="TimeStamp" type="xsd:string"/>
         <xsd:attribute name="Target" use="default" value="Production">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="Test"/>
                  <xsd:enumeration value="Production"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
         <xsd:attribute name="Version" type="xsd:string"/>
         <xsd:attribute name="SequenceNmbr" type="xsd:integer"/>
         <xsd:attribute name="ReqRespVersion" type="xsd:string"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Position">
      <xsd:complexType>
         <xsd:choice>
            <xsd:sequence>
               <xsd:element ref="Attribute" minOccurs="0" maxOccurs="unbounded"/>
               <xsd:element ref="Element" minOccurs="0" maxOccurs="unbounded"/>
               <xsd:element ref="Subtree" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
```



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



```
            </xsd:simpleType>
         </xsd:attribute>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

6.5 Order of Representation and Application

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

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

Therefore, for purposes of representing differences in an update message, positions and operations are presented in an order which favors a post-order traversal[22]. This ensures that differences may be applied sequentially to a source document to transform it into a target without invalidating the XPath of any subsequent unprocessed difference.

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

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

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

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

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

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

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

- depth-first pre-order traversal presents elements in the order that opening tags occur

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

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

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

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

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

6.6 Update Examples

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

Example 13 - initial "before" document image

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
```



```
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://www.opentravel.org/OTA
OTA_UpdateRQ.xsd"
             ReqRespVersion="2" Timestamp="2001-10-25T13:50:41EDT">
  <UniqueId Type="Profile" Id="9876543210"
              URL="http://www.vmguys.com/OTAEngine" Instance="1" />
  <Position XPath="/Profile">
    <Root Operation="replace">
      <Profile xmlns="">
        <Customer>
          <PersonName NameType="Default">
            <NameTitle>Mr.</NameTitle>
            <GivenName>George</GivenName>
            <MiddleName>A.</MiddleName>
            <SurName>Smith</SurName>
          </PersonName>
          <TelephoneInfo PhoneTech="Voice" PhoneUse="Work">
            <Telephone>
              <AreaCityCode>253</AreaCityCode>
              <PhoneNumber>813-8698</PhoneNumber>
            </Telephone>
          </TelephoneInfo>
          <PaymentForm>...</PaymentForm>
          <AddressInfo>
            <Address>
              <StreetNmbr PO_Box="4321-01">1200 Yakima St</StreetNmbr>
              <BldgRoom>Suite 800</BldgRoom>
              <CityName>Seattle</CityName>
              <StateProv PostalCode="98108">WA</StateProv>
              <CountryName>USA</CountryName>
            </Address>
          </AddressInfo>
        </Customer>
      </Profile>
    </Root>
  </Position>
</OTA_UpdateRQ>
```

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
              ReqRespVersion="2" Timestamp="2001-10-25T13:57:15EDT">
    <UniqueId Type="Profile" Id="9876543210"
              URL="http://www.vmguys.com/OTAEngine" Instance="1" />
    <Position XPath="/Profile/Customer/TelephoneInfo/Telephone/AreaCityCode">
        <Element Operation="modify">253</Element>
    </Position>
</OTA_UpdateRQ>
```


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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
              ReqRespVersion="2" Timestamp="2001-10-25T14:28:58EDT">
  <UniqueId Type="Profile" Id="9876543210"
            URL="http://www.vmguys.com/OTAEngine" Instance="2" />
  <Position XPath="/Profile/Customer">
    <Subtree Operation="insert" Child="5">
      <RelatedTraveler Relation="Child">
        <PersonName>
          <GivenName>Devin</GivenName>
          <MiddleName>R.</MiddleName>
          <SurName>Smith</SurName>
        </PersonName>
      </RelatedTraveler>
    </Subtree>
    <Subtree Operation="insert" Child="6">
      <RelatedTraveler Relation="Child">
        <PersonName>
          <GivenName>Amy</GivenName>
          <MiddleName>E.</MiddleName>
          <SurName>Smith</SurName>
        </PersonName>
      </RelatedTraveler>
    </Subtree>
    <Subtree Operation="insert" Child="7">
      <RelatedTraveler Relation="Child">
        <PersonName>
          <GivenName>Alfred</GivenName>
          <MiddleName>E.</MiddleName>
          <SurName>Newman</SurName>
        </PersonName>
      </RelatedTraveler>
    </Subtree>
  </Position>
</OTA_UpdateRQ>
```

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

```
<Profile>
   <Customer>
      <PersonName NameType="Default">
         <NameTitle>Mr.</NameTitle>
         <GivenName>George</GivenName>
         <MiddleName>A.</MiddleName>
         <SurName>Smith</SurName>
      </PersonName>
      <TelephoneInfo PhoneTech="Voice" PhoneUse="Work"
         <Telephone>
            <AreaCityCode>253</AreaCityCode>
            <PhoneNumber>813-8698</PhoneNumber>
         </Telephone>
      </TelephoneInfo>
      <PaymentForm>
         ...
      </PaymentForm>
      <Address>
         <StreetNmbr POBox="4321-01">1200 Yakima St</StreetNmbr>
```



```
         <BldgRoom>Suite 800</BldgRoom>
         <CityName>Seattle</CityName>
         <StateProv PostalCode="98108">WA</StateProv>
         <CountryName>USA</CountryName>
      </Address>
      <RelatedTraveler Relation="Child"
         <PersonName>
            <GivenName>Devin</GivenName>
            <MiddleName>R.</MiddleName>
            <SurName>Smith</SurName>
         </PersonName>
      </RelatedTraveler>
      <RelatedTraveler Relation="Child"
         <PersonName>
            <GivenName>Amy</GivenName>
            <MiddleName>E.</MiddleName>
            <SurName>Smith</SurName>
         </PersonName>
      </RelatedTraveler>
      <RelatedTraveler Relation="Child"
         <PersonName>
            <GivenName>Alfred</GivenName>
            <MiddleName>E.</MiddleName>
            <SurName>Newman</SurName>
         </PersonName>
      </RelatedTraveler>
   </Customer>
</Profile>
```

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
              ReqRespVersion="2" Timestamp="2001-10-25T15:03:05EDT">
    <UniqueId Type="Profile" Id="9876543210"
              URL="http://www.vmguys.com/OTAEngine" Instance="3" />
    <Position XPath="/Profile/Customer/RelatedTraveler[3]">
        <Subtree Operation="delete" />
    </Position>
    <Position XPath="/Profile/Customer/RelatedTraveler[1]">
        <Subtree Operation="delete" />
    </Position>
</OTA_UpdateRQ>
```

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

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

```
<Profile>
   <Customer>
      <PersonName NameType="Default">
         <NameTitle>Mr.</NameTitle>
         <GivenName>George</GivenName>
```



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

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

This examples changes the TelephoneInfo PhoneUse attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
<OTA UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA UpdateRQ.xsd"
              ReqRespVersion="2" Timestamp="2001-10-25T15:18:24EDT">
    <UniqueId Type="Profile" Id="9876543210"
              URL="http://www.vmguys.com/OTAEngine" Instance="4" />
    <Position XPath="/Profile/Customer/TelephoneInfo">
        <Attribute Name="PhoneUse" Operation="modify" Value="Home" />
    </Position>
</OTA_UpdateRQ>
```

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

```
<Profile>
   <Customer>
      <PersonName NameType="Default">
         <NameTitle>Mr.</NameTitle>
         <GivenName>George</GivenName>
         <MiddleName>A.</MiddleName>
         <SurName>Smith</SurName>
      </PersonName>
      <TelephoneInfo PhoneTech="Voice" PhoneUse="Home">
         <Telephone>
            <AreaCityCode>253</AreaCityCode>
            <PhoneNumber>813-8698</PhoneNumber>
         </Telephone>
```



```
      </TelephoneInfo>
      <PaymentForm>
         ...
      </PaymentForm>
      <AddressInfo>
         <Address>
            <StreetNmbr PO_Box="4321-01">1200 Yakima St</StreetNmbr>
            <BldgRoom>Suite 800</BldgRoom>
            <CityName>Seattle</CityName>
            <StateProv PostalCode="98108">WA</StateProv>
            <CountryName>USA</CountryName>
         </Address>
      </AddressInfo>
      <RelatedTraveler Relation="Child">
            <PersonName>
               <GivenName>Amy</GivenName>
               <MiddleName>E.</MiddleName>
               <SurName>Smith</SurName>
            </PersonName>
      </RelatedTraveler>
   </Customer>
</Profile>
```

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

This operation will delete the PO_Box attribute of StreetNmbr.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
<OTA_UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA_UpdateRQ.xsd"
              ReqRespVersion="2" Timestamp="2001-10-25T15:24:50EDT">
    <UniqueId Type="Profile" Id="9876543210"
              URL="http://www.vmguys.com/OTAEngine" Instance="5" />
    <Position XPath="/Profile/Customer/AddressInfo/Address/StreetNmbr">
        <Attribute Name="PO_Box" Operation="delete" />
    </Position>
</OTA_UpdateRQ>
```

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

```
<Profile>
   <Customer>
      <PersonName NameType="Default">
         <NameTitle>Mr.</NameTitle>
         <GivenName>George</GivenName>
         <MiddleName>A.</MiddleName>
         <SurName>Smith</SurName>
      </PersonName>
      <TelephoneInfo PhoneTech="Voice" PhoneUse="Home">
         <Telephone>
            <AreaCityCode>253</AreaCityCode>
            <PhoneNumber>813-8698</PhoneNumber>
         </Telephone>
      </TelephoneInfo>
      <PaymentForm>
         ...
      </PaymentForm>
      <AddressInfo>
         <Address>
            <StreetNmbr>1200 Yakima St</StreetNmbr>
            <BldgRoom>Suite 800</BldgRoom>
```



```
            <CityName>Seattle</CityName>
            <StateProv PostalCode="98108">WA</StateProv>
            <CountryName>USA</CountryName>
         </Address>
      </AddressInfo>
      <RelatedTraveler Relation="Child">
            <PersonName>
               <GivenName>Amy</GivenName>
               <MiddleName>E.</MiddleName>
               <SurName>Smith</SurName>
            </PersonName>
      </RelatedTraveler>
   </Customer>
</Profile>
```

Example 24 - Compound update using multiple operations

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- created by 'DiffGen' using VMTools 0.3 (http://www.vmguys.com/vmtools/) -->
<OTA UpdateRQ xmlns="http://www.opentravel.org/OTA"
              xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.opentravel.org/OTA OTA UpdateRQ.xsd"
              ReqRespVersion="2" Timestamp="2001-10-25T15:30:51EDT">
    <UniqueId Type="Profile" Id="9876543210"
              URL="http://www.vmguys.com/OTAEngine" Instance="6" />
    <Position XPath="/Profile/Customer/RelatedTraveler/PersonName">
        <Element Operation="insert" Child="1">
            <NameTitle xmlns="">Ms.</NameTitle>
        </Subtree>
    </Position>
    <Position XPath="/Profile/Customer/PersonName/MiddleName">
        <Element Operation="delete" />
    </Position>
    <Position XPath="/Profile/Customer">
        <Attribute Name="Gender" Operation="insert" Value="Male" />
    </Position>
</OTA_UpdateRQ>
```

Example 25 - document image after compound update

```
<Profile>
   <Customer Gender="Male">
      <PersonName NameType="Default">
         <NameTitle>Mr.</NameTitle>
         <GivenName>George</GivenName>
         <SurName>Smith</SurName>
      </PersonName>
      <TelephoneInfo PhoneUse="Home">
         <Telephone PhoneTech="Voice" >
            <AreaCityCode>253</AreaCityCode>
            <PhoneNumber>813-8698</PhoneNumber>
         </Telephone>
      </TelephoneInfo>
      <PaymentForm>
         ...
      </PaymentForm>
      <Address>
         <StreetNmbr>1200 Yakima St</StreetNmbr>
         <BldgRoom>Suite 800</BldgRoom>
         <CityName>Seattle</CityName>
```



```
            <StateProv PostalCode="98108">WA</StateProv>
            <CountryName>USA</CountryName>
         </Address>
         <RelatedTraveler Relation="Child">
            <PersonName>
               <NameTitle>Ms.</NameTitle>
               <GivenName>Amy</GivenName>
               <MiddleName>E.</MiddleName>
               <SurName>Smith</SurName>
            </PersonName>
         </RelatedTraveler>
      </Customer>
</Profile>
```

6.7 Validation of Update Messages

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

6.8 The Simple "Replace" verb

The Replace infrastructure verb defines an operation that updates an existing record by replacing the existing information with a complete overlay image of the document as it would appear after changes are applied. The Replace verb does not require a Read request to obtain an image of the current record prior to sending the message to replace it with the information being transmitted, although this may be desirable from the standpoint of good business practice.

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

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

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

6.9 OTA_UpdateRS – Responding to a generic OTA_UpdateRQ message

An OTA update response is formally defined by the following schema fragment:

Schema 8 - <OTA_UpdateRS>:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
   <xsd:include schemaLocation="Profile.xsd"/>
   <xsd:element name="OTA_UpdateRS">
      <xsd:complexType>
         <xsd:choice>
            <xsd:sequence>
               <xsd:choice>
                  <xsd:element ref="Success"/>
                  <xsd:element ref="Warnings"/>
               </xsd:choice>
               <xsd:element ref="UniqueId"/>
            </xsd:sequence>
            <xsd:element ref="Errors"/>
         </xsd:choice>
         <xsd:attribute name="EchoToken" type="xsd:string"/>
         <xsd:attribute name="TimeStamp" type="xsd:string"/>
         <xsd:attribute name="Target" use="default" value="Production">
            <xsd:simpleType>
               <xsd:restriction base="xsd:NMTOKEN">
                  <xsd:enumeration value="Test"/>
                  <xsd:enumeration value="Production"/>
               </xsd:restriction>
            </xsd:simpleType>
         </xsd:attribute>
         <xsd:attribute name="Version" type="xsd:string"/>
         <xsd:attribute name="SequenceNmbr" type="xsd:integer"/>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```


7 Service and Action Mappings

This section defines all the *Action* values which are permitted within each particular *Service*. In all cases there are RECOMMENDED class of delivery semantics (see section 5.3 for definition of these).

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

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

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

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

7.1 The Profile Service

The following table defines the actions available within this service:

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

7.2 The VehicleBooking Service

The following table defines the actions available within this service:

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

OTA_CancelRS	Type-A	-	OTA_CancelRQ

7.3 The AirBooking Service

The following table defines the actions available within this service:

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

7.4 The TravelInsurance Service

The following table defines the actions available within this service:

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

7.5 The HotelBooking Service

The following table defines the actions available within this service:

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

OTA_CancelRQ	Type-A	OTA_CancelRS	
OTA_CancelRS	Type-A	-	OTA_CancelRQ

7.6 The HotelResNotification Service

The following table defines the actions available within this service:

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

7.7 The HotelPropertyInformation Service

The following table defines the actions available within this service:

HotelPropertyInformation Action	*Class of Delivery*	*Response Expected*	*In Response To*
OTA_CommNotifRQ	Type-A	OTA_CommNotifRS	-
OTA_CommNotifRS	Type-A	-	OTA_CommNotifRQ
OTA_StayInfoNotifRQ	Type-A	OTA_StayInfoNotifRS	-
OTA_StayInfoNotifRS	Type-A	-	OTA_StayInfoNotifRQ
OTA_StatisticsNotifRQ	Type-A	OTA_StatisticsNotifRS	-
OTA_StatisticsNotifRS	Type-A	-	OTA_StatisticsNotifRQ
OTA_StatisticsRQ	Type-A	OTA_StatisticsRS	-
OTA_StatisticsRS	Type-A	-	OTA_StatisticsRQ
OTA_GetMsgRQ	Type-A	OTA_GetMsgRS	-
OTA_GetMsgRS	Type-A	-	OTA_GetMsgRQ
OTA_GetMsgInfoRQ	Type-A	OTA_GetMsgInfoRS	-
OTA_GetMsgInfoRS	Type-A	-	OTA_GetMsgInfoRQ

7.8 The MeetingProfile Service

The following table defines the actions available within this service:

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

OTA_CreateMeetingProfileRS	Type-A	-	OTA_CreateMeetingProfileRQ

7.9 The PackageBooking Service

The following table defines the actions available within this service:

PackageBooking Action	*Class of Delivery*	*Response Expected*	*In Response To*
OTA_PkgAvailabilityRQ	Type-A	-	-
OTA_PkgAccomAvailRS	Type-A	-	-
OTA_PkgTransportAvailRQ	Type-A	-	-
OTA_PkgTransportAvailRS	Type-A	-	-
OTA_PkgAccomFacilitiesRQ	Type-A		
OTA_PkgAccomFacilitiesAvailRS	Type-A		
OTA_PkgCreateBkgRQ	Type-A	OTA_PkgCreateBkgRS	-
OTA_PkgCreateBkgRS	Type-A	-	OTA_PkgCreateBkgRQ
OTA_PkgConfirmBkgRQ	Type-A	OTA_PkgConfirmBkgRS	-
OTA_PkgConfirmBkgRS	Type-A	-	OTA_PkgConfirmBkgRQ

7.10 The Session Service

The following table defines the actions available within this service:

Session Action	*Class of Delivery*	*Response Expected*	*In Response To*
CreateRQ	Type-A	CreateRS	-
CreateRS	Type-A	-	CreateRQ
CloseRQ	Type-A	CloseRS	-
CloseRS	Type-A	-	CloseRQ

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

7.11 Determining Services Supported

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

Example 27 - sample <SessionControlResponse> payload:

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



```
<SessionControlResponse status=”Approved”>
  <OTA_Version>2001C<OTA_Version>
  <ConversationId>20011027081907-862@host.imacompany.com</ConversationId>

  <ServicesSupported xmlns="http://www.opentravel.org/OTA"
    xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.opentravel.org/OTA ServicesSupported.xsd"
   Version="1">

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

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

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

  </ServicesSupported>
</SessionControlResponse>
```

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

7.11.1 The <ServicesSupported> element

The *ServicesSupported* element has the following attributes

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

The *ServicesSupported* element MUST contain a sequence of at least one *Service* element. N.B. *Services* and *Actions* included in a *ServicesSupported* message are always from the perspective of the system sending the message i.e. the message formally defines which services the implementation provides and which actions upon each service it implements and will accept messages for. An implementation specifies the request actions it will accept, but does not need to specify the response actions it will generate (these are implied based upon the Service definition tables earlier in this section).

7.11.2 The <Service> element

The *Service* element has the following required attributes:

- *Type:* A value of “OTA”. This is the value that MUST be used in the *type* attribute of the <*Service*> element on the ebXML header

- *Name*: The service name. This is the value which MUST be used as the content of the <*Service*> element on the ebXML header

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

7.11.3 The <Action> element

The *Action* element has the following required attributes:

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

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

7.11.4 The <Extension> element

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

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

7.11.5 The ServicesSupported Schema

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

Schema 9- ServicesSupported.xsd

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

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

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

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

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

```
      </xsd:annotation>

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

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

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

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

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

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

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



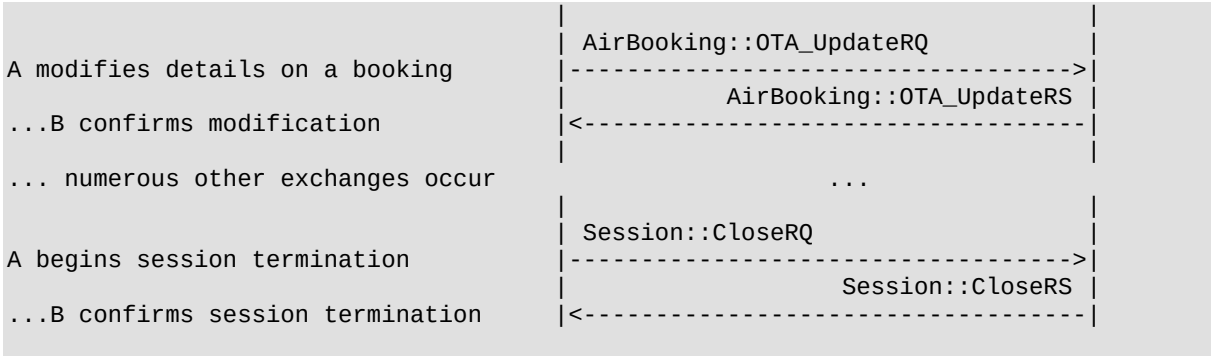
```
   </xsd:element>

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

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

</xsd:schema>
```

7.12 Sample Session Message Flow

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

Example 28 - sample session

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



A modifies details on a booking
AirBooking::OTA_UpdateRQ
AirBooking::OTA_UpdateRS
...B confirms modification
... numerous other exchanges occur
...
A begins session termination
Session::CloseRQ
Session::CloseRS
...B confirms session termination

8 Summary of Infrastructure Changes

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

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

