



Mail.XML™
Primer: Introduction and Overview

Version 9.0

DRAFT Release

2010 06 15

Working Group Chairs

Angelo Anagnostopoulos, Greyhair Software
Shawn Baldwin, BCC Software

Technical Director

Shariq Mirza, IDEAlliance

Editor:

Dianne Kennedy, IDEAlliance

Copyright (c) International Digital Enterprise Alliance, Inc. [IDEAlliance] (2004 – 2010).
All Rights Reserved.

Mail.dat is a registered trademark of IDEAlliance
Mail.XML is a trademark of IDEAlliance



This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to IDEAlliance, except as needed for the purpose of developing IDEAlliance specifications, in which case the procedures for copyrights defined in the IDEAlliance Intellectual Property Policy document must be followed, or as required to translate it into languages other than English. The limited permissions granted above are perpetual and will not be revoked by IDEAlliance or its successors or assigns.

NO WARRANTY, EXPRESSED OR IMPLIED, IS MADE REGARDING THE ACCURACY, ADEQUACY, COMPLETENESS, LEGALITY, RELIABILITY OR USEFULNESS OF ANY INFORMATION CONTAINED IN THIS DOCUMENT OR IN ANY SPECIFICATION OR OTHER PRODUCT OR SERVICE PRODUCED OR SPONSORED BY IDEALLIANCE. THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN AND INCLUDED IN ANY SPECIFICATION OR OTHER PRODUCT OR SERVICE OF IDEALLIANCE IS PROVIDED ON AN "AS IS" BASIS. IDEALLIANCE DISCLAIMS ALL WARRANTIES OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, ANY ACTUAL OR ASSERTED WARRANTY OF NON-INFRINGEMENT OF PROPRIETARY RIGHTS, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. NEITHER IDEALLIANCE NOR ITS CONTRIBUTORS SHALL BE HELD LIABLE FOR ANY IMPROPER OR INCORRECT USE OF INFORMATION. NEITHER IDEALLIANCE NOR ITS CONTRIBUTORS ASSUME ANY RESPONSIBILITY FOR ANYONE'S USE OF INFORMATION PROVIDED BY IDEALLIANCE. IN NO EVENT SHALL IDEALLIANCE OR ITS CONTRIBUTORS BE LIABLE TO ANYONE FOR DAMAGES OF ANY KIND, INCLUDING BUT NOT LIMITED TO, COMPENSATORY DAMAGES, LOST PROFITS, LOST DATA OR ANY FORM OF SPECIAL, INCIDENTAL, INDIRECT, CONSEQUENTIAL OR PUNITIVE DAMAGES OF ANY KIND WHETHER BASED ON BREACH OF CONTRACT OR WARRANTY, TORT, PRODUCT LIABILITY OR OTHERWISE.

IDEAlliance takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available. IDEAlliance does not represent that it has made any effort to identify any such rights. Information on IDEAlliance's procedures with respect to rights in IDEAlliance specifications can be found at the IDEAlliance website.

Copies of claims of rights made available for publication, assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification, can be obtained from the President of IDEAlliance.

IDEAlliance requests interested parties to disclose any copyrights, trademarks, service marks, patents, patent applications, or other proprietary or intellectual property rights which may cover technology that may be required to implement this specification. Please address the information to the President of IDEAlliance.

Abstract

This document describes the messaging protocol for use by mailers and their consignees. The Mail.XML™ Messaging Protocol defines the roles and responsibilities of Shippers and Consignees and defines the format and method for message exchange. This messaging protocol is designed to be XML and Web-Services compliant and to work with FAST. It is designed work as a module of Mail.XML™ alongside Piece Detail Record (PDR). These two Mail.XML protocols share the same base of simple type definitions.

Mail.XML and Mail.dat are trademarks of IDEAlliance.

About Mail.XML™

Mail.XML™ is bringing a paradigm change to the industry by increasing business function specific B2B (Business to Business) communication within the industry that supports automation and in the end enables cost avoidance and higher profits through improved competence and effectiveness of communication. Mail.XML is designed to increase efficiency and lower costs by removing many manual data entry processes and enabling quick near real time communication between business partners. Mail.XML currently supports container based scheduling, pick up and drop off business processes, as well as identifying different business entities responsible for performing different services such as quality of mailing, address correction, and delivery confirmation on a mailing.

The core focus of Mail.XML is communication between industry members and from industry to the final mail processing and delivery organization that delivers the mail to the end consumer, e.g., USPS.

In the next few versions of Mail.XML the focus moves across mailing supply chain channels, and includes advanced functions such as such as payment; automated verification; enabling first, second, and third party communication and incorporating presort planning, printing, and distribution processes.

What's New in Mail.XML Version 9.0

With this release, the Mail.XML Messaging Protocol moves to Version 9.0. This is a MAJOR release that corrects technical errors, implements new message sets for Transportation and Mail.dat interfaces and supports ongoing USPS modernization and automation efforts for 2010 and 2011.

About Schema Modularization

As of the release of MailXML 9.0, Mail.XML messages are now grouped into 6 message types.

- Transportation Messages

- Mailing Messages
- Data Distribution Messages
- Identification Messages
- Supply Chain Messages
- System Messages

The underlying schemas have been modularized to reflect these distinct message sets. In addition three “shared” or “common” schemas have been developed. One stores simple type definitions that can be shared across message sets. A second common module is made up of common elements and complex types that are shared across two or more message sets. The final common module is made up of system messages and the attributes, elements and complex types that are *unique* to these messages.

MailXML schema modules are designed to stand alone, enabling those interested in one or more message sets to develop and validate just those message sets without the requirement to address the entire MailXML specification. Each module will validate by itself or in combination with all other message sets when the entire MailXML Specification is considered. In addition each module will be versioned separately. Version numbers are contained within the namespace designation. Modules will be documented individually as well.

The following MailXML XSD modules/namespaces have been developed:

- **mailxml_tm.xsd**: This module contains all the transportation (or FAST) messages and the attributes, elements and complex types that are *unique* to these messages. **Namespace=mailxml tm:**
- **mailxml_mm.xsd**: This module contains all the mailing messages and the attributes, elements and complex types that are *unique* to these messages. **Namespace=mailxml mm:**
- **mailxml_dd.xsd**: This module contains all the data distribution messages (eDoc) and the attributes, elements and complex types that are *unique* to these messages. **Namespace=mailxml dd:**
- **mailxml_id.xsd**: This module contains all the identification messages and the attributes, elements and complex types that are *unique* to these messages. **Namespace=mailxml id:**
- **mailxml_sc.xsd**: This module contains all the supply chain messages and the attributes, elements and complex types that are *unique* to these messages. **Namespace=mailxml sc:**
- **mailxml.xsd**: This module contains all the system messages and the attributes, elements and complex types that are *unique* to these messages. Note that in order for the <MessageResponseRetrievalResponse to work, all the other message modules are included by the system.xsd
Namespace=mailxml:
- **mailxml_defs.xsd**: This module contains all the common definitions of attributes, elements and complex types that are used across one or more message types. **Namespace=mailxml defs:**
- **mailxml_base.xsd**: This module contains all the simple types that are used anywhere in mailxml. These can be considered a building block for any message group. **Namespace=mailxml base:**

- **mailxml.xsd:** This module contains the system messages of MailXML and is used to build custom profiles for MailXML. *Namespace=mailxml:*

The Mail.XML™ Messaging Roadmap

The Mail.XML Messaging Specification has been organized into a set of documents. This *Schemas Specification* is one document in a set of documents that make up the Mail.XML Specification 9.0. Updates in this Specification are NOT backwardly compatible with previous versions. Other documents in the specification set include:

- *Mail.XML™ 9.0: Primer* provides an introduction and overview of the Mail.XML 8.0 Specification and an overview of the Mail.XML messages.
- *Mail.XML™ 9.0: Transportation Messaging Specification* documents all transportation messages
- *Mail.XML™ 9.0: Mailing Messaging Specification* documents all mailing messages
- *Mail.XML™ 9.0: Data Distribution Messaging Specification* documents all data distribution messages
- *Mail.XML™ 9.0: Identification Messaging Specification* documents all identification messages
- *Mail.XML™ 9.0: Supply Chain Messaging Specification* documents all supply chain messages
- *Mail.XML™ 9.0: System Messaging Specification* documents all supply chain messages
- *Mail.XML™ 9.0: Simple Types Specification* documents all simple types used across MailXML messages
- *Mail.XML™ 9.0: Fault and Return Information Specification* documents all fault and return codes.
- *Mail.XML™ 9.0: Schemas* contains the .XSDs that make up the Mail.dat Transaction Messaging Specification.

Table of Contents

Abstract.....	iii
About Mail.XML™.....	iii
What's New in Mail.XML Version 9.0.....	iii
About Schema Modularization.....	iii
The Mail.XML™ Messaging Roadmap.....	v
1. Introduction.....	1
1.1 Mail.XML™ Design Goals.....	1
1.2 How Mail.XML™ Relates to Other Standards.....	2
1.2.1 Mail.dat®.....	2
1.2.2 XML.....	2
1.2.3 XML Namespaces.....	2
1.2.4 XML Schema.....	2
1.2.5 SOAP.....	3
1.2.6 WSDL.....	3
1.2.7 FAST.....	3
1.2.8 Mail.XML™ Piece Detail Record [PDR].....	3
1.2.9 USPS Wizard Web Service.....	4
1.3 Definitions.....	4
1.3.1 Requirement Wording Note.....	4
1.3.2 Mail.XML Semantic Definitions.....	4
1.4 Defining Mail.XML using an XML-Schema.....	4
1.4.1 Schema Validation.....	5
1.4.2 Security.....	5
1.4.3 Internationalization Issues.....	6
1.4.4 Mail.XML™ Namespaces.....	6
1.4.5 Mail.XML Messaging XSDs.....	6
1.5 Conventions.....	7
2. Mail.XML Messaging Overview.....	9
2.1 Mail.XML Business Scenarios.....	9
2.2 Protocol Overview.....	9
2.2.1 Messages, Requests and Responses.....	9
2.2.2 Request/Response Model.....	10
2.2.3 Shipper/Consignee, Requester/Responder, Sender/Receiver.....	10
2.2.4 Third Party Messages.....	10
2.2.5 Notifications.....	11

2.2.6 Deliveries	11
2.2.7 Large Record Transmission	12
2.2.8 Identifiers for Jobs	12
2.2.9 Grouping Message Types	12
2.2.9.1 Transportation Messages	12
2.2.9.2 Mailing Messages	13
2.2.9.3 Data Distribution Messages	14
2.2.9.4 Identification Messages	14
2.2.9.5 Supply Chain Messages	15
2.2.9.6 System Messages	15
2.2.9.7 Messages Deleted in MailXML 9.0	15
2.3 Mapping Mail.XML to SOAP	16
2.3.1 Mail.XML Outside SOAP	17
2.4 Mail.XML Identifiers.....	17
2.5 Mail.XML Datatyping	20
2.6 Mail.XML Namespaces	21
2.7 Fault and Status Codes.....	22
2.7.1 Tracking ID	22
2.7.2 Fault Codes	22
2.7.3 Fault Description.....	23
2.8 Return Information	23
2.8.1 Consignee Facility	23
2.8.2 Return Codes.....	24
2.8.3 Return Description	24
3 Building Custom Profiles of MailXML.....	25
3.1 Step 1: Specify Module Namespaces.....	25
3.2 Step 2: Import Modules	25
3.3 Step 3: Response Messages	26
4 The Fault Recovery Model	29

1. Introduction

Mail.dat® is an industry standard for efficient communications/data exchange among those providing list processing, mail production, and mail processing services. Mail.dat® has, since its inception been defined, transmitted, and modified as a set of fixed-record files. In the last four years, since the maturation of XML and XML Web Services, the need for more dynamic communications has emerged. In addition, new technologies, which hold bright promise in their flexibility and efficiency, have developed a broader availability in the marketplace. As a result the IDEAlliance has launched a new Mail.XML™ protocol that will mirror Mail.dat® where possible, yet diverge to optimize communications formats and techniques between mailers and the USPS. Today the two sub-committees of Mail.XML, Messaging and PDR have merged and the Mail.XML Specification is developed and advanced by a single IDEAlliance Working Group.

1.1 Mail.XML™ Design Goals

The Mail.XML™ Working Group defined a number of design goals on requirements analysis and much thought and discussion.

Some of the most important design goals are included here for reference:

1. Mail.XML shall be straightforwardly usable over the Internet.
2. Mail.XML shall support a wide variety of applications and not constrain data formats.
3. Mail.XML shall conform to a specific XML syntax and be defined using W3C XML Schema Definition Language (XSD).
4. Mail.XML requirements shall constrain the Transportation appointment booking and delivery verification to practical and implementable mechanisms.
5. Compactness of representation in Mail.XML is of minimal importance. NOTE: this is a statement about low level encoding methodology, e.g., the use of XML in general and the particular choice of tag and attribute names in particular.
6. Mail.XML shall keep protocol and packaging overhead to a minimum. NOTE: this is a statement about protocol overhead in the sense of round trips, complexity, and other high-level performance effects. It is not a contradiction of the previous point. The design achieves its performance objectives by optimizing the high level design of the protocol flow and state management, not by micro optimizing
7. Mail.XML will eliminate element collisions by moving all elements into one or more Mail.XML namespaces.
8. Mail.XML needs to define the characteristics of the communication over SOAP Version 1.2.
9. Mail.XML will be defined as a Web Service using WSDL 1.2 to define the end points of the Mail.XML message-oriented conversation on top of SOAP.

1.2 How Mail.XML™ Relates to Other Standards

Many other standards describe how to transmit data of one form or another between systems. This section briefly discusses some of these protocols and describes their relationship to Mail.XML.

1.2.1 Mail.dat®

Mail.dat® is an industry standard for efficient communications/data exchange among those providing list processing, mail production, and mail processing services. The Mail.XML™ Protocol is designed to provide a Web-Services compliant messaging communication component to Mail.dat® and newer generation Wizard Web Services. The protocol uses field names and values directly from the Mail.dat Specification where ever possible and extends them by defining new fields when necessary.

1.2.2 XML

Mail.XML is an application of the Extensible Mark-up Language (XML 1.1). Basic concepts in Mail.XML are represented using the element/attribute mark-up model of XML. Note, however, that Mail.XML is a *protocol* that encodes messaging conversations, and so in that way differs fundamentally from other pure document applications of XML such as MathML (mathematical formula mark-up language) and SMIL (Synchronized Multimedia Interchange Language).

1.2.3 XML Namespaces

XML Namespaces 1.1 provides a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references. XML Namespaces enable us to define a set of unique element names within a given context. Namespaces prevent element collisions and enable computers to unequivocally determine exact points of reference. Such unique addressing is critical to reliable messaging between Web Services. In Mail.XML, all elements will be moved into a Mail.XML namespace to enable Mail.XML to function as a Web Service.

1.2.4 XML Schema

XML Schema Definition Language 1.1 is a three-part specification from the W3C that provides the capability to specify and constrain XML applications. XML Schema provides a superset of the specification capabilities of the XML DTD. Only XML Schema enables specification of type that is expected by Web services. This difference is

so critical that the SOAP specification specifically states that a SOAP message “MUST NOT” contain a DTD. Since Mail.XML 2.0 is a protocol, it requires features such as type definitions found in XML Schemas but not supported by XML DTDs.

1.2.5 SOAP

SOAP (Simple Object Access Protocol) 1.1 is a key enabler of Web Services through XML. SOAP enables the exchange of XML messages so that services can easily describe their capabilities and allow any other service, application or device on the Internet to easily invoke those capabilities. Mail.XSML, working with SOAP, adds the mechanisms for the management of syndication on the Web. SOAP is being widely used as transport for Web services related RPC. Mail.XML is designed to layer its communications on SOAP. This will enable developers and users to take advantage of their existing communication infrastructure and management services while taking advantage of Mail.XML for their content distribution applications or content subscription activities.

1.2.6 WSDL

Web Services Definition Language (WSDL) is an XML based description language that currently describes RPC based end-points. This is currently being developed by W3C for extending RPC to enable messaging-style program end-points. Mail.XML has an XML-based protocol for conversation between client and server. For Mail.XML we are defining Mail.XML end-points with WSDL (either message-oriented or RPC-based or both). This will eliminate the need for Mail.XML client packages. Any WSDL to Java or any other programming language based generator will be able to generate Mail.XML client interfaces in that programming language.

1.2.7 FAST

Facility Access and Shipment Tracking (FAST) is an initiative of the USPS to improve the current drop shipment process and decrease overall dock wait times. FAST will replace the current DSAS System and improve integration with source USPS systems. The Mail.XML Messaging Specification is designed to interface with FAST and development has been closely coordinated with the USPS.

1.2.8 Mail.XML™ Piece Detail Record [PDR]

The Mail.XML Piece Detail Record Specification was initially drafted to describe the Piece Detail Record of Mail.dat® for use by mailers and their consignees. PDR is used for manifest mailings; those working with computer sort. If used, the PDR acts as an extension of the PQT file. With the release of Mail.XML 5.0, PDR has been merged directly into the Mail.XML specification.

1.2.9 USPS Wizard Web Service

The USPS Wizard Web Service (WWS) allows customers to electronically submit postage statement forms via Simple Object Access Protocol (SOAP) over the Internet. The Wizard Web Service is an alternative to using the Postage Statement Wizard® application to create and submit a postage statement, or sending a Mail.dat® file. The WWS also allows mailings to be updated and canceled, though a user name and password are required with all submissions. Clerks may view postage statements generated by the WWS and perform transactions on those postage statements as they would on statements created using the Postage Statement Wizard software. This system is not available for use at acceptance units until beta testing is complete. Mail.XML supports Wizard Web Service as well as the Mail.dat environment.

1.3 Definitions

1.3.1 Requirement Wording Note

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 RFC 2119.

In the HTML version of this specification, those key words are **CAPITALIZED BOLD**. Capitalization is significant; non-capitalized uses of the key words are intended to be interpreted in their normal, informal, English language way. Bold face is not significant, and is used to aid comprehension, but the bold font is non normative and the absence of a bold font **MUST NOT** be given any semantic interpretation.

1.3.2 Mail.XML Semantic Definitions

These definitions are used throughout this document. Readers will most likely not fully understand these definitions without also reading through the specification.

1.4 Defining Mail.XML using an XML-Schema

Mail.XML has selected XML Schema for the definition format. This selection was made so that Mail.XML could work over SOAP and function as a Web service. SOAP uses XML Schema for its definition and specifically disallows specification by XML DTDs for interoperability with SOAP.

1.4.1 Schema Validation

It is important to note that XML Schema is used as the definition format for Mail.XML but that validation against the schema is not strictly required. In fact there are two places where XML Schema validation is implied by Mail.XML:

- A Receiver **MAY** perform validation on incoming Mail.XML messages.
- A Sender **MUST** send only valid Mail.XML messages.

Note, however, that "validation" could in principle be implemented in a variety of ways. A Receiver **MAY** use any alternate representation of Mail.XML syntax, and perform some alternate form of validation against that representation, as long as the results are AS-IF the governing Mail.XML XML Schema had been used.

1.4.2 Security

The Mail.XML protocol deliberately does not address security, because the required levels of security can be achieved via existing and emerging Internet/Web security mechanisms.

In the specific case of digital signatures, non repudiation, and similar concepts, two things have happened that have steered the Authoring Group away from the notion of having digital signatures inside Mail.XML itself:

- Separate efforts are underway to define digital signing standards for XML documents.
- Defining digital signing standards for XML documents is quite tricky, and requires defining a canonical text representation of the documents (because the digital hash functions hash the *textual* representation of a document, not its *logical* representation). The Mail.dat Working Group did not want to define its own, possibly conflicting, canonical representation rules to solve this problem.

Independent of any future XML digital signing standards, Mail.XML implementations can achieve necessary security using a variety of methods, including:

- Encryption can be accomplished at the transport level, e.g., via SSL, PGP, or S/MIME.
- Applications can agree to send digitally signed content as items within the Mail.XML protocol, with verification performed at the application level (above Mail.XML).
- Syndicators and Subscribers can be authenticated using certificates implemented at the transport level.

1.4.3 Internationalization Issues

For the initial version of Transaction Messaging, internationalization was not considered to be a goal. It is likely that internationalization will be considered for future revisions of this specification.

1.4.4 Mail.XML™ Namespaces

In order to enable interoperability in the Web services environment, Mail.XML 9.0 uses eight unique namespaces to support Mail.XML messaging. These Namespaces are discussed in more detail in a following section.

The namespace declaration for Mail.XML V9.0 reads:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:mailxml_defs="http://idealliance.org/maildat/Specs/md091/mailxml9.0/mailxml_defs"
  xmlns:mailxml_base="http://idealliance.org/maildat/Specs/md091/mailxml9.0/base"
  xmlns:mailxml_tm="http://idealliance.org/maildat/Specs/md091/mailxml9.0/mailxml_tm"
  xmlns:mailxml_dd="http://idealliance.org/maildat/Specs/md091/mailxml9.0/mailxml_dd"
  xmlns:mailxml_sc="http://idealliance.org/maildat/Specs/md091/mailxml9.0/mailxml_sc"
  xmlns:mailxml_mm="http://idealliance.org/maildat/Specs/md091/mailxml9.0/mailxml_mm"
  xmlns:mailxml_id="http://idealliance.org/maildat/Specs/md091/mailxml9.0/mailxml_id"
  xmlns:mailxml="http://idealliance.org/maildat/Specs/md091/mailxml80A/mailxml"
  targetNamespace="http://idealliance.org/maildat/Specs/md091/mailxml80A/mailxml"
  elementFormDefault="qualified"
  attributeFormDefault="qualified"
  version="mailxml900607010">
```

1.4.5 Mail.XML Messaging XSDs

This Specification defines eight XML Schemas. As of the release of MailXML 9.0, Mail.XML messages are now grouped into 6 message types.

- Transportation Messages
- Mailing Messages
- Data Distribution Messages
- Identification Messages

- Supply Chain Messages
- System Messages

The underlying schemas have been modularized to reflect these distinct message sets. In addition three “shared” or “common” schemas have been developed. One stores simple type definitions that can be shared across message sets. A second common module is made up of common elements and complex types that are shared across two or more message sets. The final common module is made up of system messages and the attributes, elements and complex types that are *unique* to these messages.

The Mail.XML Schemas can be found on the Mail.XML website. See www.mailxml.org.

1.5 Conventions

This document contains a number of constructs that are identified:

- Examples (XML instance files)
- XML schema fragments
- Figures

In addition, as specific Mail.XML tags are documented, they will be set in a typewriter face with an open bracket, but no closing bracket. Namespace designations will not be used. For example `<DeliveryApptRequest` would be used when discussing the Delivery Appointment Request element/message.

2. Mail.XML Messaging Overview

Two entities are involved in forming a business relationship where Mail.XML is used. The *Shipper* is sending mail that is delivered to the *Consignee*. Mail.XML defines a series of messages that go back and forth between these two parties to coordinate the Transaction of the items to be delivered.

2.1 Mail.XML Business Scenarios

Mail.XML business scenarios are used throughout this specification to illustrate when and how Mail.XML messages are to be used.

2.2 Protocol Overview

The Mail.XML protocol is primarily a request/response protocol that allows for fully symmetric implementations, where both the Shipper and Consignee can initiate requests or send responses.

There are several key concepts that form the foundation of the Mail.XML protocol.

2.2.1 Messages, Requests and Responses

Mail.XML uses *message* exchange as its fundamental protocol model, where a *message* is defined for the purposes of this specification to be a SOAP payload as specified by the Mail.XML 4.1 Specification.).

Mail.XML messages are either in the form of *requests* and *responses*. A *request* asks for the performance of an operation. For example, when a Shipper wishes to request a delivery appointment from the Consignee, the Shipper sends the Consignee a message containing a `<DeliveryApptRequest` request with a shipper appointment request identifier. In this case the response comes from the Consignee and echoes the shipper appointment request identifier as a reference.

2.2.2 Request/Response Model

A request/response pair describes every logical operation in Mail.XML. All operations are forced to fit this model; thus, a valid Mail.XML protocol session always comprises an even number of messages when it is in the idle state (i.e., there is a matching response for every request).

2.2.3 Shipper/Consignee, Requester/Responder, Sender/Receiver

The Shipper and Consignee assume several different roles during Mail.XML protocol operations: Shipper versus Consignee, Requester versus Responder, and Sender versus Receiver.

The definition of Shipper and Consignee is based on the business relationships: the Shipper distributes mail to the Consignee. These terms are capitalized throughout this specification wherever they refer specifically to the roles of the parties in a Mail.XML relationship.

For some message sets the designations of Shipper/Consignee do not apply. These messages may be initiated and responded to by third parties such as consolidators or FAST.

The definition of Requester/Responder is based on who initiates the Mail.XML operation. The initiator is the *Requester*, and the other party, who performs the operation, is the *Responder*. It is possible for a Shipper to be either a Requester or a Responder, depending on the particular operation. The same is true for a Consignee.

Finally, the concept of Sender and Receiver are used in this specification to describe the relationship with respect to the transmission of a single message. A message travels from *Sender* to *Receiver* (and this thus forms the definition of Sender and Receiver).

Note that a Mail.XML operation inherently consists of a Request/Response pair. Thus, the Requester starts out being a Sender, sending a message, containing a request, to the Receiver. The request will be a SOAP message. The Receiver of this first message becomes the Responder. When the Responder has performed the operation and wishes to return the results, the Responder becomes the Sender of a message containing the response, and the initial Requester is now the Receiver.

2.2.4 Third Party Messages

Typically the Shipper and Consignee are the primary sender/receiver of messages during Mail.XML protocol operations. But messages may also be initiated by third parties or partners. This is particularly true when querying as to status or when updating

information originally sent by the mail owner or shipper. A new facility has therefore been added to each request that will identify the party (and the software) who is submitting a request. The business model is that the party receiving such a message will have implemented security operations to verify that the third party has the clearance to be provided a response. Security remains outside the Mail.XML protocol, but these fields are included to facility messaging security through implementation of security business rules.

2.2.5 Notifications

Mail.XML allows for USPS to “push” messages that are “Notifications” to outside parties. The notifications are intended to prompt a query from the recipient in order to receive the full set of information from the postal service.

Examples of Notifications in Mail.XML include:

- By For Conflict Notification
- Consignee Goods Receipt Notification
- Delivery Point Validation Notification
- Full Service Address Correction Notification
- Full Service Nixie Detail Notification
- Full Service Container Visibility Notification
- Start The Clock Notification
- Unscheduled Consignee Goods Receipt Notification

The standard response upon the receipt of a Notification is the <NotificationResponse.

2.2.6 Deliveries

Mail.XML allows for USPS to “push” messages or make “Deliveries” to outside parties. The delivery is a “push” of a full set of information from the postal service. If the delivery is made up of a large amount of data, it may be divided into a series of deliveries.

Examples of Deliveries in Mail.XML include:

- By For Conflict Delivery
- Consignee Goods Receipt Delivery
- Delivery Point Validation Delivery
- Full Service Address Correction Delivery
- Full Service Nixie Detail Delivery
- Full Service Container Visibility Delivery
- Start The Clock Delivery
- Unscheduled Consignee Goods Receipt Delivery

The standard response to a Notification is the <DeliveryResponse

2.2.7 Large Record Transmission

Sometimes records returned to a Query or Delivered are too large to efficiently deliver as a single message. To facilitate transmission of a large record over a series of messages, attributes have been added so that transmissions can be counted and the receiver will be able to determine when the transmission of the record is complete.

2.2.8 Identifiers for Jobs

For many of the Mail.XML messages a choice of identifiers for jobs is offered. If working in the Mail.dat environment, identifiers are the User License Code followed by the Mail.dat Job ID. For others the identifiers will be the CustomerGroupID and the MailingGroupID.

2.2.9 Grouping Message Types

Mail.XML messages may be grouped into six message types. The XSDs are modularized based on these types:

- Transportation Messages
- Mailing Group Transactions
- Data Distribution Messages
- Identification Messages
- Supply Chain Messages
- System Messages

2.2.9.1 Transportation Messages

AllApptCloseoutResponse
AllDeliveryApptCloseoutRequest
AllPickupApptCloseoutRequest
ApptInductionCloseoutRequest
ApptInductionCloseoutResponse
ConsigneeGoodsReceipt
ConsigneeGoodsReceiptDelivery
ConsigneeGoodsReceiptNotification
ContainerStatusQueryRequest
ContainerStatusQueryResponse
ContainerUpdateRequest
ContainerUpdateResponse
CustomerSupplierAgreementQueryRequest
CustomerSupplierAgreementQueryResponse
DeliveryApptCancelCreateRequest

DeliveryApptCancelCreateResponse
DeliveryApptCancelRequest
DeliveryApptCancelResponse
DeliveryApptContentRemoveRequest
DeliveryApptContentRemoveResponse
DeliveryApptCreateRequest
DeliveryApptCreateResponse
DeliveryApptMultiStopUpdateRequest
DeliveryApptQueryRequest
DeliveryApptQueryResponse
DeliveryApptShellCancelRequest
DeliveryApptShellCancelResponse
DeliveryApptShellCreateRequest
DeliveryApptShellCreateResponse
DeliveryApptShellUpdateRequest

DeliveryApptShellUpdateResponse
DeliveryApptStatusQueryRequest
DeliveryApptStatusQueryResponse
DeliveryApptTransportationUpdateRequest
DeliveryApptTransportationUpdateResponse
DeliveryApptUpdate
DeliveryApptUpdateRequest
DeliveryApptUpdateResponse
DeliveryContentCancelRequest
DeliveryContentCancelResponse
DeliveryContentCreateRequest
DeliveryContentCreateResponse
DeliveryContentQueryRequest
DeliveryContentQueryResponse
DeliveryContentUpdateRequest
DeliveryContentUpdateResponse
DeliveryReApptRequest
GoodsReceiptResponse
OpenApptQueryRequest
OpenApptQueryResponse

PartnerApptQueryRequest
PartnerApptQueryResponse
PartnerContentAssignmentRequest
PartnerContentAssignmentResponse
PickupApptCancelRequest
PickupApptCancelResponse
PickupApptCreateRequest
PickupApptCreateResponse
PickupApptUpdateRequest
PickupApptUpdateResponse
RecurringApptQueryRequest
RecurringApptQueryResponse
StaleContentDelivery
StaleContentNotification
StaleContentQueryRequest
StaleContentQueryResponse
UnscheduledConsigneeGoodsReceipt
UnscheduledConsigneeGoodsReceiptDelivery
UnscheduledConsigneeGoodsReceiptNotification

2.2.9.2 Mailing Messages

BeginCombinedMailingRequest
BeginCombinedMailingResponse
BundleDetailCancelRequest
BundleDetailCancelResponse
BundleDetailCreateRequest
BundleDetailCreateResponse
CloseMailingGroupRequest
CloseMailingGroupResponse
ConsolidatedPeriodicalStatementCreateRequest
ConsolidatedPeriodicalStatementCreateResponse
ContainerBundleReportCancelRequest
ContainerBundleReportCancelResponse
ContainerBundleReportCreateRequest
ContainerBundleReportCreateResponse
ContainerBundleReportQueryRequest
ContainerBundleReportQueryResponse

CustomerMailReportCreateRequest
CustomerMailReportCreateResponse
EndCombinedMailingRequest
EndCombinedMailingResponse
MailingGroupQueryRequest
MailingGroupQueryResponse
MailPieceCancelRequest
MailPieceCancelResponse
MailPieceCreateRequest
MailPieceCreateResponse
MailPieceUpdateRequest
MailPieceUpdateResponse
OpenMailingGroupRequest
OpenMailingGroupResponse
PaymentMessageQueryRequest
PaymentMessageQueryResponse

PeriodicalStatementCreateRequest	PostageStatementStatusQueryResponse
PeriodicalStatementCreateResponse	QualificationReportCreateRequest
PeriodicalStatementQueryRequest	QualificationReportCreateResponse
PeriodicalStatementQueryResponse	QualificationReportQueryRequest
PostageStatementCancelRequest	QualificationReportQueryResponse
PostageStatementCancelResponse	ReconciliationReportQueryRequest
PostageStatementCreateRequest	ReconciliationReportQueryResponse
PostageStatementCreateResponse	SummaryZipDestinationReportCreateRequest
PostageStatementQueryRequest	SummaryZipDestinationReportCreateResponse
PostageStatementQueryResponse	
PostageStatementStatusQueryRequest	

2.2.9.3 Data Distribution Messages

ByForConflictDelivery	FullServiceContainerVisibilityQueryResponse
ByForConflictNotification	FullServiceDataQualityVerificationReportDelivery
ByForConflictQueryRequest	FullServiceDataQualityVerificationReportNotification
ByForConflictQueryResponse	FullServiceDataQualityVerificationReportQueryRequest
DeliveryPointValidationDelivery	FullServiceDataQualityVerificationReportQueryResponse
DeliveryPointValidationNotification	FullServiceNixieDetailDelivery
DeliveryPointValidationQueryRequest	FullServiceNixieDetailNotification
DeliveryPointValidationQueryResponse	FullServiceNixieDetailQueryRequest
FullServiceAddressCorrectionDelivery	FullServiceNixieDetailQueryResponse
FullServiceAddressCorrectionNotification	FullServiceStartTheClockDelivery
FullServiceAddressCorrectionQueryRequest	FullServiceStartTheClockNotification
FullServiceAddressCorrectionQueryResponse	FullServiceStartTheClockQueryRequest
FullServiceContainerVisibilityDelivery	FullServiceStartTheClockQueryResponse
FullServiceContainerVisibilityNotification	
FullServiceContainerVisibilityQueryRequest	

2.2.9.4 Identification Messages

CastofCharactersCancelRequest	CustomerRelationIdentifyUpdateResponse
CastofCharactersCancelResponse	USPSCRIDCreateRequest
CastofCharactersCreateRequest	USPSCRIDCreateResponse
CastofCharactersCreateResponse	USPSCRIDQueryRequest
CastofCharactersUpdateRequest	USPSCRIDQueryResponse
CastofCharactersUpdateResponse	USPSMIDCreateRequest
CustomerRelationIdentifyQueryRequest	USPSMIDCreateResponse
CustomerRelationIdentifyQueryResponse	USPSMIDQueryRequest
CustomerRelationIdentifyUpdateRequest	USPSMIDQueryResponse

2.2.9.5 Supply Chain Messages

OriginalContainerLinkageCancelRequest
OriginalContainerLinkageCancelResponse
OriginalContainerLinkageCreateRequest
OriginalContainerLinkageCreateResponse
PostageAdjustmentCreateRequest
PostageAdjustmentCreateResponse
SiblingContainerCancelRequest
SiblingContainerCancelResponse
SiblingContainerCreateRequest
SiblingContainerCreateResponse

2.2.9.6 System Messages

Fault
LargeTransactionDividerResult
MessageResponseRetrievalRequest
MessageResponseRetrievalResponse
DeliveryResponse
NotificationResponse
SystemVersionQueryRequest
SystemVersionQueryResponse

2.2.9.7 Messages Deleted in MailXML 9.0

USPSForecastCancelRequest
USPSForecastCancelResponse
USPSForecastCreateRequest
USPSForecastCreateResponse
USPSForecastUpdateRequest
USPSForecastUpdateResponse

2.3 Mapping Mail.XML to SOAP

For Mail.XML an explicit binding to SOAP is provided. This binding can be found in the following (partial) WSDL script:

```
<binding name="Mail.XML -binding" type="tns:tm-portType"/>
```

Mail.XML 4.1 was specifically designed to function as a Web service and to take advantage of SOAP as a messaging protocol. The entire Mail.XML message was designed to be carried in the SOAP body.

Mail.XML uses XML as the format for its message header, delivery and subscription elements. All Mail.XML message elements **MUST** be formatted in accordance with the XML 1.1 specification. Furthermore, Mail.XML message elements **MUST** be well formed and **MUST** be valid according to the Mail.XML XML Schema Definitions. This document does not repeat the general rules for proper XML encoding; readers are expected to refer to the XML specification.

To understand how Mail.XML works with SOAP, see Figure 2.1. Mail.XML messages are carried within the Body of the SOAP Envelope. Faults are carried within the SOAP Fault.

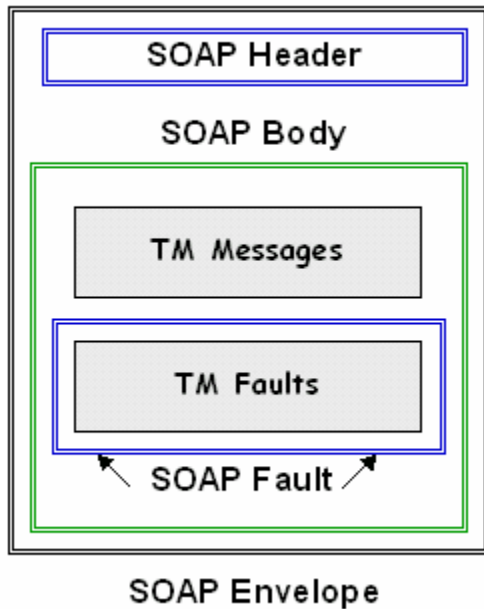


Figure 2.1 Mail.XML carried by SOAP

Mail.XML Messaging is encoded in a Mail.XML/SOAP format. This is simply an XML file where Mail.XML messages are wrapped in a SOAP envelope and carried in the SOAP Body. Note that Mail.XML system faults are carried in <Fault within SOAP faults. Business level return information is part of the Mail.XML message and encode as <ReturnInfo.

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env='http://www.w3.org/2002/12/soap-
envelope'>
```

```

<env:Header/>
<env:Body>
  <mailxmlTM:DeliveryApptRequest
...
...
...
  </mailxmlTM:DeliveryApptRequest>
</env:Body><env:Envelope>

```

2.3.1 Mail.XML Outside SOAP

Because Mail.XML Messaging is defined in XML, the messages can be viewed as small XML files. If necessary these message files can be sent and received by means other than SOAP through normal file transfer protocols such as FTP.

2.4 Mail.XML Identifiers

Mail.XML defines a number of identifiers that control the access to content and enable content management throughout the messaging process. Identifier attributes include:

Attribute	Datatype	Role
AdvanceMailingID	String (9)	A unique ID for identifying an advance mailing
BarcodeID	Numeric String (02)	Unique ID for Barcode
BusinessEntitySchedulerCorpID"	String (12)	Unique ID for the for the corporation a business partner requesting closeout information.
BusinessEntitySchedulerID	String (12)	Unique ID for a business partner requesting closeout information.
CancelConfirmID	String (12)	Unique confirmation ID for appointment cancellations. Generated by consignee.
CapsID	String (400)	Unique CAPS ID
CarrierID	String (15)	Unique Carrier ID
CloseoutResponseID	String (12)	Unique identifier, PO! will put this information on the closeout response given to the mailer.
CombinedMailingID	NonNegativeInteger	Unique ID for a combined mailing
ComponentID	String(8)	Unique ID that identifies the component.
ConsigneeApptID	String (12)	ID assigned to the shipment by the consignee, Should be unique for a period of 1 year.
ConsigneeContainerGroupingID	String (16)	An optional identification that the consignee may attach to a shipment detail block.
ConsigneeContentID	String (12)	A unique ID that the consignee uses to identify mail content
ConsigneeMultiStopID	String (12)	An ID assigned by the consignee that links multiple shipments onto a single truck. Value should be

Attribute	Datatype	Role
		unique for a period of one year.
ContactID	String (30)	A unique ID for the contact at a consignee facility. May be a driver's license number or an employer ID
ContainerID	String(06)	ID assigned by WWS to uniquely identify a container
ContainerBundleReportID	String (12)	ID for the Container Bundle Report
CQTDatabaseID	NonNegativeInteger (8 digits)	Unique ID for CQT database
CoPaletizationCustomerContainerID	String(50)	ID assigned by WWS to uniquely identify a copalletization container
CreatorSchedulerID	String (12)	Unique ID for the creator of standalone content.
CreatorSchedulerCorpID	String (12)	Unique ID for the corporation of creator of standalone content.
CRID	String (10)	UniqueCustomer Registration ID
CRIDRequestID	String (12)	Unique ID for the CRID requested
CRIDCreateRequestID	String (12)	Unique ID for the CRIDs created.
CustomerBundleID	String (12)	Unique ID to identify the customer bundle.
CustomerContainerID	String(50)	ID assigned by WWS to uniquely identify a customer container
CustomerGroupID	String(25)	ID assigned by WWS to uniquely identify the customer group
CustomerQualificationReportID	String (12)	ID for qualification report that was submitted
CSAID	String (10)	Customer Supplier Agreement ID
DatabaseContainerID	Number (6)	Mail.dat Serial Number. Used to link mail.dat files, must be mutually exclusive across all segments and container types of a job ID. CSM file: 14-19
EndConfirmID	String (12)	Unique ID to indicate End Combined Mailing
FASTApptID	String (9)	Unique FAST appointment ID
FASTContentID	String(9)	Unique FAST Content ID
FASTSchedulerID	String(12)	Unique FAST Scheduler ID
FASTLogisticsID	String(12)	Unique FAST Logistics ID
FobID	String (8)	Unique Freight on Board ID
LinktoMaildatSummaryJobID	String(8)	Link to job ID
LinktoMaildatSummaryContainerID	Number (6)	Link to Database Container ID
LinkTransactionID	String(12)	ID that identifies the copalletization container linkage
LogisticsCorpID	String(12)	ID that identifies the logistics corporation
LogisticsSchedulerID	String(12)	ID that identifies the logistics scheduler
MachineID	String (04)	A unique ID for the barcode machine
MaildatJobID	String (8)	Mail.dat Job ID as given by originator of file. Should be unique compared to all other job IDs

Attribute	Datatype	Role
		supplied by same source.
Mailer6ID	String (06)	Unique ID for Mailer (6)
Mailer9ID	String (09)	Unique ID for Mailer (9)
MailingGroupID	NonNegativeInteger	ID assigned by WWS to a mailing group
MailPreparerID	String (12)	A unique ID for the mail preparer
MailPieceID	Complex Model	This is a complex model consisting of IMB or PlanetCode and Delivery Point Zip followed or Delivery Container ID and Delivery Point Zip
MailPieceConfirmID	String (12)	A unique ID for the mail piece that was created
MailWeightConfirmID	String (12)	A unique ID for the mail weight report
MID	Numeric String (6) or Numeric String (9)	A unique Mailer ID
MIDRequestID	String (12)	Unique ID for the MID requested
MIDCreateRequestID	String (12)	Unique ID for the MIDs created.
MPUID	Number(5)	A unique ID for the MPU
OriginalSenderMsgID	String (12)	A unique ID for original sender when there are a cast of characters
ParentBundleID	String (12)	Unique ID to identify the parent bundle.
ParentContainerID	String	
ParentContainerRefID	String(6)	A unique ID for the parent container
PeriodicalVersionID	NonNegativeInteger	A unique ID for a periodical version
PieceID	String (22)	A unique ID for a mail piece
PostageGroupingID	String(8)	A unique DI from Mail.dat for the postage grouping
PostageAdjustmentConfirmID	String(12)	A unique ID assigned when a postage statement is created/updated
ReceiverMailPieceGroupID	String(12)	A unique ID assigned by the receiver for the mail piece group
ReceiverMsgID	String (12)	A unique ID for the receiver when there are a cast of characters
RecurringApptID	String(8)	A unique ID for a recurring appointment.
RemoveConfirmID	String (12)	ID indicating content has been removed from an appointment
RequestorSchedulerID	String (12)	A unique ID for the requestor of information about standalone content.
RequestorSchedulerCorpID	String (12)	A unique ID for the corporation of the requestor of information about standalone content
SchedulerCorpID	String (12)	An agreed upon ID that identifies the scheduler corporation to the consignee. Value agreed to by

Attribute	Datatype	Role
		both the shipper and consignee.
SchedulerID	String (12)	An agreed upon ID used to segment the scheduler role. Value agreed to by both the shipper and consignee.
SchedulerContentID	String (12)	A unique ID that the scheduler uses to identify mail content
SegmentID	String(4)	A unique ID that identifies the Segment
SenderMailPieceGroupID	String(12)	ID assigned to a senders mail piece group
SenderRequestTrackingID	String (12)	ID assigned for a sender request
ShipperApptRequestID	String (12)	ID assigned to the shipment by the shipper. Should be unique for a period of 1 year.
ShipperContainerGroupingID	String (16)	An optional identification that the shipper may attach to a shipment detail block.
ShipperMultiStopID	String (12)	An ID assigned by the shipper that links multiple shipments onto a single truck. Value should be unique for period of 1 year.
StatementID	NonNegativeInteger	Statement ID
TrackingID	String (12)	Unique tracking ID for responses. Generated by consignee.
UniqueContainerBarcode	String (24)	Mail.dat unique barcode identifier among containers within user license code for 3 month period. This ID is 24 characters to allow for the USPS 24 character bar code
UserID	String (12)	An identifier of the user in the Appointment Closeout
UserPostageStatementSequenceID	String (10)	Unique ID for a postage statement by sequence
UserPostageStatementID	String (20)	A unique ID for the USPS Postage Statement
USPSPostageStatementSequenceID	String (10)	The USPS Postage Statement Sequence ID
USPSPublicationID	String (12)	A unique ID for the publication
VersionID	String (12)	A unique ID for the version of bundle, container, document, etc.
VersionStatementID	NonNegativeInteger	A unique version statement ID
WeightAdConfirmID	String (12)	A unique ID for the weight/ad percent that was submitted
WWSJobID	String (8)	Unique WWS ID for a job

2.5 Mail.XML Datatyping

One of the strengths of defining Mail.XML with an XML Schema is to benefit from the ability to specify simple datatypes that XSD offers. Mail.XML defines datatypes for elements and attributes within the Mail.XML messages in a “simpleTypes” definition

area of the Mail.XML schema. See the example below:

```
<xs:simpleType name="phoneNumber">
  <xs:restriction base="xs:string">
    <xs:maxLength value="25"/>
    <xs:minLength value="1"/>
    <xs:whiteSpace value="preserve"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name = "mailClassType">
  <xs:restriction base = "xs:string">
    <xs:enumeration value = "BPM"/>
    <xs:enumeration value = "Media"/>
    <xs:enumeration value = "Library"/>
  </xs:restriction>
</xs:simpleType>
```

Simple datatypes are defined within a `mailxml_base:` namespace. Complex datatypes used within Mail.XML, are defined within the `mailxml:` namespace.

This document does not repeat the general rules for XML datatyping; readers are expected to refer to the XSD specification to understand the datatypes utilized by Mail.XML.

2.6 Mail.XML Namespaces

XML namespaces provide a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references. XML Namespaces enable us to define a set of unique element names within a given context. Namespaces prevent element collisions and enable computers to unequivocally determine exact points of reference. Such unique addressing is critical to reliable messaging between Web services. In the Mail.XML Messaging Specification 4.1 all elements are moved into the `mailxml.XML` namespace to enable Mail.XML to function as a Web service and utilize SOAP messaging. Simple datatypes that are shared by Mail.XML and PDR, are defined within the `mailxml:` namespace.

Mail.XML 9.0 namespaces are:

- http://idealliance.org/maildat/Specs/md091/mailxml9.0/mailxml_defs
- <http://idealliance.org/maildat/Specs/md091/mailxml9.0/base>
- http://idealliance.org/maildat/Specs/md091/mailxml9.0/mailxml_tm
- http://idealliance.org/maildat/Specs/md091/mailxml9.0/mailxml_dd
- http://idealliance.org/maildat/Specs/md091/mailxml9.0/mailxml_sc
- http://idealliance.org/maildat/Specs/md091/mailxml9.0/mailxml_mm
- http://idealliance.org/maildat/Specs/md091/mailxml9.0/mailxml_id
- <http://idealliance.org/maildat/Specs/md091/mailxml9.0/mailxml>

NOTE: The namespace definition is not a URL that can be directly resolved. Rather it is simple an identifier for the namespace. The prefixes used in this specification are examples. Anyone is free to assign their own namespace prefixes. Implementers should honor the namespace declarations rather than matching the prefix strings used here.

2.7 Fault and Status Codes

Faults, such as a message timeout or invalid XML are to be communicated using the `<fault>` element and returned within the detail section of the SOAP fault. The fault is made up of a tracking ID and one or more fault codes and optional fault descriptions.

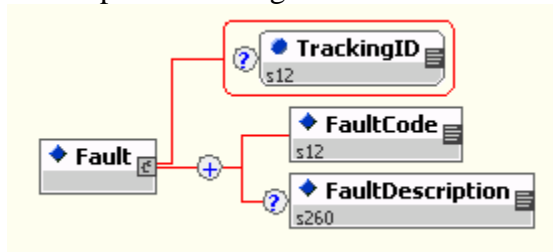


Figure 2.2 Fault Structure

2.7.1 Tracking ID

This is an identifier provided to facilitate message tracking in case a fault condition interrupts the messages. If messages “time out” the tracking ID can be used to retrieve messages that could not be transmitted for some reason.

2.7.2 Fault Codes

Fault codes contain the three-digit code `positiveInteger` value, the corresponding defined fault messages to indicate faults or status.

NOTE: The *Mail.XML: Fault and Return Information Specification* documents all fault and return codes.

2.7.3 Fault Description

This field allows for 260 characters that can be used to provide a natural language description of the Fault Code. It can be used to clarify or qualify the fault code.

NOTE: While this Specification defines the fault description, that field may be enhanced with information more specific to the systems and implementation of the Responder.

2.8 Return Information

Some Messages contain a <ReturnInfo element that is made up of Consignee Facility information and a Return Code and Return Description.. See Figure 2.3. Return Information is only used to return business information and not to be used for operational faults. Return codes contain the four-digit code `positiveInteger` value; the corresponding defined return messages to indicate business information or status. Return codes may be errors or they may be warnings/informational.

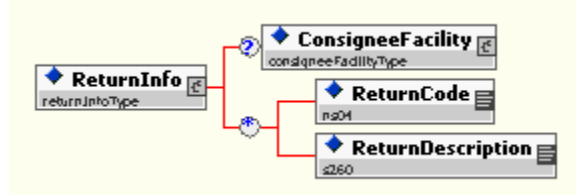


Figure 2.3 Return Information Structure

2.8.1 Consignee Facility

A return information block information may document the Consignee Facility. See the structure in Figure 2.4. At a minimum the Company Name, and Facility Number, Facility Address or both are required.

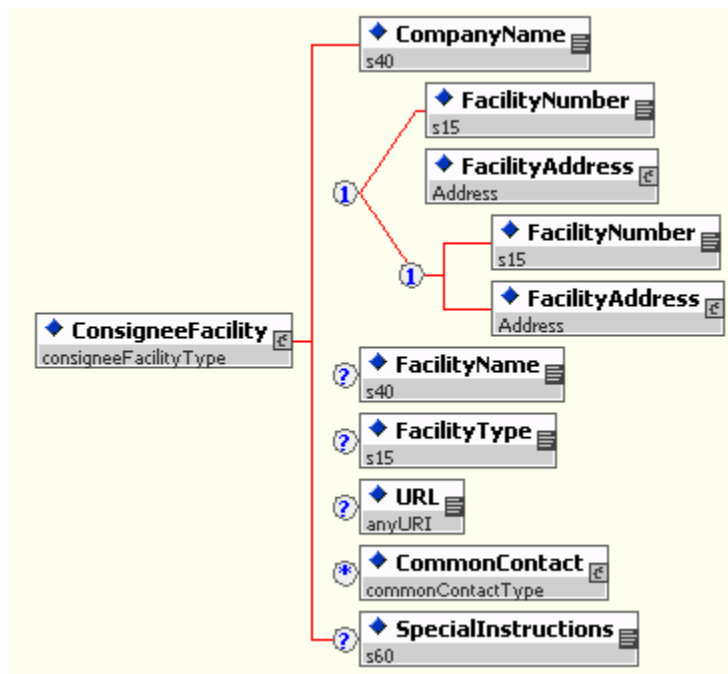


Figure 2.4 Consignee Facility Structure

2.8.2 Return Codes

Return codes contains the four-digit code `positiveInteger` value, the corresponding defined return messages. The format of the four-digit code determines whether this is a Mail.XML error code (1xxx) or whether it is a user defined error (3xxx) or a user defined warning (4xxx)

NOTE: The *Mail.XML Mail.XML: Fault and Return Information Specification* documents all Mail.XML fault and return codes.

2.8.3 Return Description

This field allows for 260 characters that can be used to provide a natural language description of the Return Code. It can be used to clarify or qualify the code.

NOTE: While this Specification defines the Return Description, that field may be enhanced with information more specific to the systems and implementation of the Responder.

3 Building Custom Profiles of MailXML

Custom profiles of MailXML are built using the mailxml.xsd module. There are three steps to build a custom profile:

3.1 Step 1: Specify Module Namespaces

Each module of MailXML can now be versioned independently. So, for example, the Mailing and Data Distribution messages may have updates and a new version, while the Base and Transportation Messages are held at their current version. The versioning is controlled by the schema declaration at the top of mailxml.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:mailxml_defs="http://idealliance.org/maildat/Specs/md091/
mailxml90/mailxml_defs"
  xmlns:mailxml_base="http://idealliance.org/maildat/Specs/md091/
mailxml90/base"
  xmlns:mailxml_tm="http://idealliance.org/maildat/Specs/md091/
mailxml90A/mailxml_tm"
  xmlns:mailxml_dd="http://idealliance.org/maildat/Specs/md091/
mailxml90/mailxml_dd"
  xmlns:mailxml_sc="http://idealliance.org/maildat/Specs/md091/
mailxml90/mailxml_sc"
  xmlns:mailxml_mm="http://idealliance.org/maildat/Specs/md091/
mailxml90/mailxml_mm"
  xmlns:mailxml_id="http://idealliance.org/maildat/Specs/md091/
mailxml90A/mailxml_id"
  xmlns:mailxml="http://idealliance.org/maildat/Specs/md091/
mailxml90/mailxml"
  targetNamespace="http://idealliance.org/maildat/Specs/md091/
mailxml90A/mailxml"
  elementFormDefault="qualified" attributeFormDefault="qualified"
  version="mailxml90060710">
```

In this figure we are using modules from the MailXML 9.0A release for Transportation Messaging and Identification Messaging. Modules for MailXML 9.0 are employed for the remainder.

Note: MailXML 9.0 will be the first modularized version of MailXML. All modules will be baselined as MailXML 9.0. As modules are updated, the versioning of individual modules will increment.

3.2 Step 2: Import Modules

The second step in developing a custom profile is to import the modules that will make up your implementation. These import statements are near the top of the mailxml.xsd. Simply use XML comments to ignore those modules that you do not want to use.

```
<!-- Transportation Messaging Module -->
<xs:import namespace="http://idealliance.org/maildat/Specs/md091/
  aillxml90/mailxml_tm" schemaLocation="mailxml_tm_121209.xsd"/>
```

```

<!-- Data Distribution Module -->
<!--<xs:import namespace="http://idealliance.org/maildat/Specs/md091/
mailxml90/mailxml_dd" schemaLocation="mailxml_dd_121209.xsd"/>-->

<!-- Supply Chain Messaging Module -->
<xs:import namespace="http://idealliance.org/maildat/Specs/md091
/mailxml90/mailxml_sc" schemaLocation="mailxml_sc_121209.xsd"/>

<!-- Mailing Messaging Module -->
<!--<xs:import namespace="http://idealliance.org/maildat/Specs/md091/
mailxml90A/mailxml_mm" schemaLocation="mailxml_mm_121209.xsd"/>-->

<!-- Identification Module -->
<!--<xs:import namespace="http://idealliance.org/maildat/Specs/md091/
mailxml90/mailxml_id" schemaLocation="mailxml_id_121209.xsd"/>-->

```

In the example above only the Transportation Messages and Supply Chain messages are included in the custom profile of MailXML.

Note: The mailxml_base and mailxml_defs modules must be included as they are used by every module.

3.3 Step 3: Response Messages

To complete the customization of your MailXML Profile, you must comment out all the response messages that are not included in your profile. The response messages are organized by message set so it should be easy to use XML comments to exclude the response messages that do not fit your profile.

```

<!--=== Message Response Retrieval Response Type ===-->
<!-- Customize your mailXML profile by including only the message
responses for the message modules you are implementing -->
<xs:complexType name="messageResponseRetrievalResponseType">
  <xs:sequence>
    <xs:choice>
      <xs:element name="QueryResults">
        <xs:complexType><xs:sequence>
          <xs:choice>
            <!-- ++++++ Include Transportation Responses ++++++ -->
            <xs:element ref="mailxml_tm:AllApptCloseoutResponse"/>
            <xs:element ref="mailxml_tm:ApptInductionCloseoutResponse"/>
            <xs:element ref="mailxml_tm:ContainerStatusQueryResponse"/>
            <xs:element ref="mailxml_tm:ContainerUpdateResponse"/>
            <xs:element ref="mailxml_tm:CustomerSupplierAgreementQueryResponse"/>
            <xs:element ref="mailxml_tm:DeliveryApptCancelCreateResponse"/>
            <xs:element ref="mailxml_tm:DeliveryApptCancelResponse"/>
            <xs:element ref="mailxml_tm:DeliveryApptContentRemoveResponse"/>
            <xs:element ref="mailxml_tm:DeliveryApptCreateResponse"/>
            <xs:element ref="mailxml_tm:DeliveryApptQueryResponse"/>
            <xs:element ref="mailxml_tm:DeliveryApptShellCancelResponse"/>
            <xs:element ref="mailxml_tm:DeliveryApptShellCreateResponse"/>
            <xs:element ref="mailxml_tm:DeliveryApptShellUpdateResponse"/>
            <xs:element ref="mailxml_tm:DeliveryApptStatusQueryResponse"/>
            <xs:element
ref="mailxml_tm:DeliveryApptTransportationUpdateResponse"/>
            <xs:element ref="mailxml_tm:DeliveryApptUpdateResponse"/>
            <xs:element ref="mailxml_tm:DeliveryContentCancelResponse"/>
            <xs:element ref="mailxml_tm:DeliveryContentCreateResponse"/>
            <xs:element ref="mailxml_tm:DeliveryContentQueryResponse"/>

```

```

<xs:element ref="mailxml_tm:DeliveryContentUpdateResponse"/>
<xs:element ref="mailxml_tm:GoodsReceiptResponse"/>
<xs:element ref="mailxml_tm:OpenApptQueryResponse"/>
<xs:element ref="mailxml_tm:PartnerApptQueryResponse"/>
<xs:element ref="mailxml_tm:PickupApptCancelResponse"/>
<xs:element ref="mailxml_tm:PickupApptCreateResponse"/>
<xs:element ref="mailxml_tm:PickupApptUpdateResponse"/>
<xs:element ref="mailxml_tm:RecurringApptQueryResponse"/>

<!-- ++++++ Include Data Distribution Responses ++++++ -->
<!--<xs:element ref="mailxml_dd:ByForConflictQueryResponse"/>
<xs:element ref="mailxml_dd:DeliveryPointValidationQueryResponse"/>
<xs:element
ref="mailxml_dd:FullServiceAddressCorrectionQueryResponse"/>
<xs:element
ref="mailxml_dd:FullServiceContainerVisibilityQueryResponse"/>
<xs:element
ref="mailxml_dd:FullServiceDataQualityVerificationReportDelivery"/>
<xs:element
ref="mailxml_dd:FullServiceDataQualityVerificationReportQueryResponse"/>
<xs:element ref="mailxml_dd:FullServiceNixieDetailQueryResponse"/>
<xs:element ref="mailxml_dd:FullServiceStartTheClockQueryResponse"/>
-->

<!-- ++++++ Include Mailing Message Responses ++++++ -->
<!--<xs:element ref="mailxml_mm:BeginCombinedMailingResponse"/>
<xs:element ref="mailxml_mm:BundleDetailCancelResponse"/>
<xs:element ref="mailxml_mm:BundleDetailCreateResponse"/>
<xs:element ref="mailxml_mm:CloseMailingGroupResponse"/>
<xs:element
ref="mailxml_mm:ConsolidatedPeriodicalStatementCreateResponse"/>
<xs:element ref="mailxml_mm:ContainerBundleReportCancelResponse"/>
<xs:element ref="mailxml_mm:ContainerBundleReportCreateResponse"/>
<xs:element ref="mailxml_mm:ContainerBundleReportQueryResponse"/>
<xs:element ref="mailxml_mm:CustomerMailReportCreateResponse"/>
<xs:element ref="mailxml_mm:EndCombinedMailingResponse"/>
<xs:element ref="mailxml_mm:MailPieceCancelResponse"/>
<xs:element ref="mailxml_mm:MailPieceCreateResponse"/>
<xs:element ref="mailxml_mm:MailPieceUpdateResponse"/>
<xs:element ref="mailxml_mm:MailingGroupQueryResponse"/>
<xs:element ref="mailxml_mm:OpenMailingGroupResponse"/>
<xs:element ref="mailxml_mm:PaymentMessageQueryResponse"/>
<xs:element ref="mailxml_mm:PeriodicalStatementCreateResponse"/>
<xs:element ref="mailxml_mm:PeriodicalStatementQueryResponse"/>
<xs:element ref="mailxml_mm:PostageStatementCancelResponse"/>
<xs:element ref="mailxml_mm:PostageStatementCreateResponse"/>
<xs:element ref="mailxml_mm:PostageStatementQueryResponse"/>
<xs:element ref="mailxml_mm:PostageStatementStatusQueryResponse"/>
<xs:element ref="mailxml_mm:QualificationReportCreateResponse"/>
<xs:element ref="mailxml_mm:QualificationReportQueryResponse"/>
<xs:element ref="mailxml_mm:ReconciliationReportQueryResponse"/>
<xs:element
ref="mailxml_mm:SummaryZipDestinationReportCreateResponse"/>
-->

<!-- ++++++ Include ID Message Responses ++++++ -->
<!--<xs:element ref="mailxml_id:CastofCharactersCancelResponse"/>
<xs:element ref="mailxml_id:CastofCharactersCreateResponse"/>
<xs:element ref="mailxml_id:CastofCharactersUpdateResponse"/>
<xs:element ref="mailxml_id:CustomerRelationIdentifyQueryResponse"/>
<xs:element ref="mailxml_id:CustomerRelationIdentifyUpdateResponse"/>
<xs:element ref="mailxml_id:USPSCRIDCreateResponse"/>
<xs:element ref="mailxml_id:USPSCRIDQueryResponse"/>
<xs:element ref="mailxml_id:USPSMIDCreateResponse"/>
<xs:element ref="mailxml_id:USPSMIDQueryResponse"/>
-->

```

```

<!-- ++++++++ Include Supply Chain Message Responses ++++++++ -->
<xs:element ref="mailxml_sc:OriginalContainerLinkageCancelRequest"/>
<xs:element ref="mailxml_sc:OriginalContainerLinkageCancelResponse"/>
<xs:element ref="mailxml_sc:OriginalContainerLinkageCreateRequest"/>
<xs:element ref="mailxml_sc:OriginalContainerLinkageCreateResponse"/>
<xs:element ref="mailxml_sc:PostageAdjustmentCreateRequest"/>
<xs:element ref="mailxml_sc:PostageAdjustmentCreateResponse"/>
<xs:element ref="mailxml_sc:SiblingContainerCancelRequest"/>
<xs:element ref="mailxml_sc:SiblingContainerCancelResponse"/>
<xs:element ref="mailxml_sc:SiblingContainerCreateRequest"/>
<xs:element ref="mailxml_sc:SiblingContainerCreateResponse"/>

<!-- These System responses are required by all implementations -->
<xs:element ref="mailxml:DeliveryResponse"/>
<xs:element ref="mailxml:NotificationResponse"/>
<xs:element ref="mailxml:SystemVersionQueryResponse"/>

    </xs:choice>
    <xs:element name="ReturnInfo"
type="mailxml_defs:basicReturnInfoType" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:element ref="mailxml_defs:QueryError"/>
</xs:choice>
</xs:sequence>
    <xs:attribute name="TrackingID" type="mailxml_base:s12"
use="required"/>
</xs:complexType>

```

Responses for Data Distribution, Mailing Messages and Supply Chain messages have been commented out (or excluded) from the profile that only implements Transportation Messaging and Supply Chain Messages.

Note: If you do not comment out the responses from modules you are not including in your profile, you will get an error message that the response is undefined. This is a clue for you to comment out the responses that do not apply.

4 The Fault Recovery Model

Mail.XML Messaging is Web Services based. For every request, there will be a paired response. Sometimes, due to system problems, a response may “timeout” or fail for some other reason. This business scenario outlines the business rules for fault and fault recovery:

1. Each response message is given a unique internal ID for tracking purposes when it is sent.
2. If the response message fails to get through for any reason a <Fault> is sent within the detail information of the SOAP Fault. The Mail.XML fault carries the TrackingID as a mechanism for the Receiver to poll for the responses that have failed.

```
<Fault TrackingID="55441">
  <FaultCode>501</FaultCode>
  <FaultDescription>Temporary Responder
Error</FaultDescription>
</Fault>
```

3. The Receiver (Shipper) sends a <ApptResponseRetrievalRequest> to the Sender (Consignee) listing all TrackingIDs that have come through in the SOAP Fault messages.

```
<ApptResponseRetrievalRequest TrackingID="55441" />
```

4. The Sender (Consignee) responds with a <ApptResponseRetrievalResponse>. If the response can now be sent it will be:

```
<ApptResponseRetrievalResponse>
  <ApptResponse TrackingID="55441">
    <ApptCancelResponse ShipperApptRequestID="55667712345522"
ConsigneeApptID="123456789012"
SchedulerCorpID="789012345678" SchedulerID="901234567890">
      <CancelConfirmID>444444444444</CancelConfirmID>
    </ApptCancelResponse>
  </ApptResponse>
</ApptResponseRetrievalResponse>
```

If the response is not available, the response will say not to try back and will provide a fault:

```
<ApptResponseRetrievalResponse>
  <ApptResponse TrackingID="55441" ComeBackLater="No">
    <Fault><FaultCode>551</FaultCode></Fault>
  </ApptResponse>
</ApptResponseRetrievalResponse>
nse>
</ApptResponseRetrievalResponse>
```